JOHANNES KEPLER
UNIVERSITÄT LINZ

Author
Stefan Neuhuber

Submission

Institut of Networks and Security

Thesis Supervisor

Univ.-Prof. Priv.-Doz. DI Dr. René Mayrhofer

February, 2017

802.1x for home users and guest networks

Master's Thesis
to confer the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Contents

1. Introduction . . . **1**

1.1. Motivation . . . 1

1.2. Threat Model and security concerns . . . 2

1.3. Structure . . . 2

2. State of the art . . . **3**

3. Employed Technologies and Tools . . . **8**

3.1. OpenWrt . . . 8

3.1.1. Architecture . . . 9

3.1.2. Build System . . . 9

3.1.3. Flash Memory Layout . . . 10

3.1.4. The Unified Configuration Interface (UCI) System . . . 13

3.2. Port-Based Authentication - 802.1x . . . 15

3.2.1. Primary Components . . . 17

3.2.2. Authentication Process . . . 19

3.2.2.1. Communication between Supplicant and Authenticator (802.1x) . 20

3.2.2.2. Communication between Authenticator and Authentication Server (RADIUS) . . . 22

3.2.2.3. Communication between Supplicant and Authentication Server (EAP-Method) . . . 23

3.3. EAPoL/EAPoW Protocol . . . 24

3.3.1. Packet Structure . . . 25

Version . . . 25

Type . . . 25

Length . . . 26

Packet Body . . . 26

3.4. Extensible Authentication Protocol (EAP) . . . 26

3.4.1. Packet Structure . . . 27

Code . . . 28

Identifier . . . 28

Length . . . 29

Data . . . 29
3.4.2. Packet Types . . . 29
EAP-Request/Response . . . 29
EAP-Success/Failure . . . 30
3.4.3. EAP-Methods . . . 30
3.4.3.1. Packet Structure . . . 31
Type . . . 31
Data . . . 31
3.4.3.2. EAP-MD5 . . . 31
3.4.3.3. EAP-TLS . . . 32
3.4.3.4. EAP-TTLS . . . 34
3.4.3.5. PEAP . . . 35
3.5. RADIUS . . . 37
3.5.1. Packet Structure . . . 37
3.6. Dynamic VLANs . . . 41

4. Implementation . . . 42
4.1. Dynamic VLAN and Guest network configuration in OpenWrt . . . 42
4.1.1. Network configuration . . . 43
4.1.2. DHCP configuration . . . 48
4.1.3. Firewall configuration . . . 48
4.1.4. Wireless configuration . . . 49
4.2. Authentication Server configuration using FreeRADIUS . . . 50
4.2.1. Basic configuration . . . 50
4.2.1.1. radiusd.conf . . . 51
4.2.1.2. eap.conf . . . 53
4.2.1.3. clients.conf . . . 54
4.2.1.4. Enabling additional modules . . . 55
4.2.1.5. SQL configuration . . . 55
4.2.2. Populating the User database . . . 56
4.2.3. Issues with FreeRADIUS on OpenWrt . . . 59
4.3. Web interface . . . 60
4.3.1. User management . . . 60
4.3.2. Group management . . . 64

4.3.3. Network/VLAN overview . . . 65
4.3.4. Portability and Modifiability . . . 66
4.4. Credential distribution . . . 67
4.4.1. Distribution via QRCodes . . . 68
4.4.2. Mobile application for administrators . . . 69
4.5. Hard- and Software requirements . . . 72

5. Future work . . . 73

6. Conclusion . . . 74

7. Appendix . . . 75
7.1. EAP-Method Types . . . 75
7.2. RADIUS Attribute Types . . . 77

Abstract

The vast majority of wireless home networks is secured by pre-shared key authentication models, such as WPA2-PSK, which results in a number of security issues rooted in the wireless protocol itself as well as the environmental conditions of home networks. Apart from the security aspects, a user-based authentication scheme also provides better control over by whom and when the network is accessed rather than which device is connecting to the network. A user-based authentication scheme also allows for a more granular control of the network and its users by enforcing certain restrictions such as access times or account expiration. This thesis implements and discusses a WPA2-Enterprise (WPA2-802.1x EAP) solution for wireless home networks while running all necessary components and services on a single OpenWrt based consumer device. The implemented solution provides administrators of wireless home networks with all the key features needed to administrate this user-based authentication system as well as comfortable ways of distributing credentials to the user or guest.

Abstract

Der größte Teil von drahtlosen Heim-Netzwerken werden durch pre-shared key Authentifikationsmodelle abgesichert, wie z.B. WPA2-PSK, welche eine gewisse Menge an Sicherheitsproblemen mit sich bringen, die einerseits dem Wireless Protokoll und andererseits den Umgebungsbedingungen von Heim-Netzwerken zugrunde liegen. Abgesehen von den Sicherheitsaspekten, bietet ein Benutzer-basierendes Authentifikationsschema bessere Kontrolle über von wem und wann auf das Netzwerk zugegriffen wird, als nur welches Gerät sich zum Netzwerk verbindet. Ein Benutzer-basierendes Authentifikationsschema erlaubt ebenfalls eine genauere Kontrolle des Netzwerks und dessen Benutzer durch die Einführung gewisser Restriktionen wie z.B. Zugriffszeiten oder das Ablaufen eines Kontos. Diese Arbeit implementiert und diskutiert eine WPA2-Enterprise (WPA2-802.1x EAP) Lösung für drahtlose Heim-Netzwerke, welche alle benötigten Komponenten und Dienste auf einem einzigen OpenWrt basierenden Router betreibt. Die implementierte Lösung bietet Administratoren von drahtlosen Heim-Netzwerken alle nötigen Funktionen zur Verwaltung eines Benutzer-basierenden Authentifikationssystems als auch komfortable Möglichkeiten zur Verteilung der Anmeldedaten von Benutzern und Gästen.

1. Introduction

Nowadays WPA2-PSK (Wi-Fi Protected Access - Pre-Shared Key) is the default wireless security scheme for most home networks. Although WPA2-PSK provides a good basic protection against unwanted access to a private network this authentication model has a series of well known weaknesses which may be exploited rather easily. For example, a connected device or malicious user may sniff the traffic [1][2] of other connected devices or spoof another device MAC address and therefore impersonate said device. Apart from WPA2-PSK's weaknesses the human factor has to be considered as well. Many home network administrators may most likely choose an easy to guess password and disclose it to a considerable large amount of users such as guests or neighbors without ever changing it. This may become a rather high risk factor to a network if we consider the amount of foreign mobile devices roaming our private and public networks.
This thesis will address the above mentioned security issues by introducing a user-based authentication scheme (using WPA2-Enterprise/WPA2-802.1x) rather than a device based authentication (WPA2-PSK). In addition, one of the main goals of the proposed solution should be ease-of-use, to enable guests to connect to the network with a minimal amount of effort by the administrator of the network and the guest himself. By using WPA2-Enterprise also a more detailed and granular administration will be possible. This may enable the network's owner/administrator to grant every user individual access rights to portions of the network, facilitate a sort of parental control by limiting access to certain times of day and generate temporary access for guest users. The credentials themselves can then be generated so they expire after a certain amount of time or they are managed by the administrator and revoked if necessary.

1.1. Motivation

The goal of this work is to provide users and administrators of home networks with the possibility of an easy way to secure their wireless network with an 802.1x port-based authentication scheme rather than device-based schemes like WPA2-PSK. This will introduce a more secure wireless home environment with an additional set of features that wouldn't be possible in a WPA2-Personal (WPA/WPA2-PSK) based network. For example the home network's administrator will be able to grant individual users access to certain parts of the network as well as a self contained guest network, generate temporary guest users and introduce a sort of parental control by limiting access times of specific users (e.g. access to the internet is only granted between 9AM and 8PM). The

main focus will therefore be on creating an 802.1x/RADIUS authentication solution that is as easy as possible to install, administrate and use by home users and their respective guest users. Further this solution will illustrate approaches to easily distribute user's credentials to their corresponding devices.
To this end all services needed to realise this 802.1x port-based authentication solution will be run on one consumer router running OpenWrt, instead of being spread across multiple devices and systems like this is usually the case with setups of this kind (see Section 3.2.1.). Also this solution should provide a centralised management interface to control various aspects of the system, which are usually also handled by a series of different standalone software solutions.

1.2. Threat Model and security concerns

WPA-Personal (WPA2-PSK) represents the most popular authentication scheme for home environments due to it's simpleness in administration and distribution to users. Although it provides a good basic protection against unwanted access to a wireless network there are a series of well known weaknesses which can rather easily be exploited [1][2]. Considering many private wireless networks are in range of multiple households and therefore potentially malicious users this might impose a serious security risk. Also allowing guest users to join your main network might introduce corrupted and malicious devices to the network.
Apart from those concerns the human element is also a huge factor, considering that former guests and potentially neighbours might disclose the wireless password to unwanted users. This becomes even more of an issue due to the fact that most administrators of such networks choose weak passwords (which are easy to guess or crack with dictionary attacks) and rarely change them over time.

1.3. Structure

This work offers an overview of currently existing solutions in the area of home user guest networks as well as technologies used by the proposed and implemented 802.1x solution and also give an outlook on how the current implementation could be improved. These topics will be discussed in the following chapters:

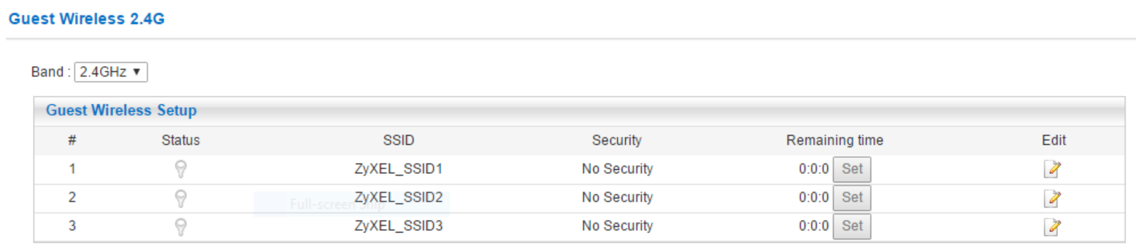
- Chapter “State of the Art”: An overview of currently existing solutions in the sector of wireless guest networks in general and 802.1x port-based authentication solutions in particular.
- Chapter “Employed Technologies and Tools”: This chapter represents a comprehensive

collection of technologies used by the implementation of this project. It discusses OpenWrt, port-based authentication as well as the protocols involved in realising such an authentication scheme. This includes Extensible Authentication Protocol (EAP), Extensible Authentication Protocol over Lan/Wireless (EAPoL/EAPoW), RADIUS (these represent the foundation of port-based authentication) and a short overview of dynamic VLAN assignment which is used to segment the network accordingly.

- Chapter "Implementation": Describes and discusses the necessary steps of configuring the router and operating system to provide the needed infrastructure for hosting the guest network, as well as the configuration of the authentication server (FreeRadius) and all further involved components. Further this chapter discusses the implementation of an user interface, which allows the management of the system's key features as well as a proposed mobile application as an addition or alternative to the former mentioned user interface.
- Chapter "Future Work": Provides an outlook on what could be improved or added to the system's current implementation.

2. State of the art

Nowadays wireless guest networks are mainly accomplished by segmenting the network via separate SSIDs, which in turn are secured using WPA-Personal (WPA/WPA2-PSK) and potentially be distributed to the guest user using Wi-Fi Protected Setup (WPS). Figures 1, 2 and 3 show such an example configuration and its possibilities using a ZyXEL NBG6815 wireless router:



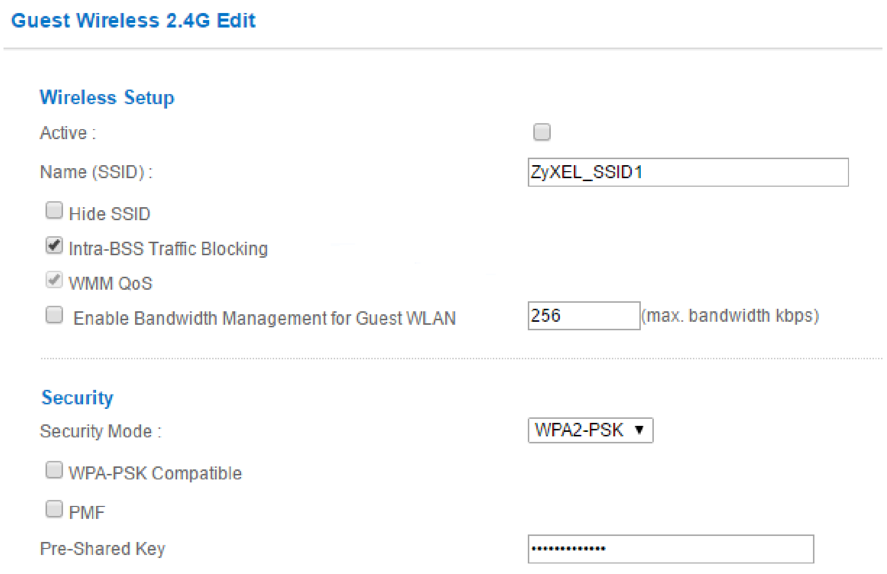
Guest Wireless 2.4G

Band : 2.4GHz ▾

Guest Wireless Setup

#	Status	SSID	Security	Remaining time	Edit
1	○	ZyXEL_SSID1	No Security	0:0:0 Set	✎
2	○	ZyXEL_SSID2	No Security	0:0:0 Set	✎
3	○	ZyXEL_SSID3	No Security	0:0:0 Set	✎

Figure 1: ZyXEL SSID based guest networks [3]



Guest Wireless 2.4G Edit

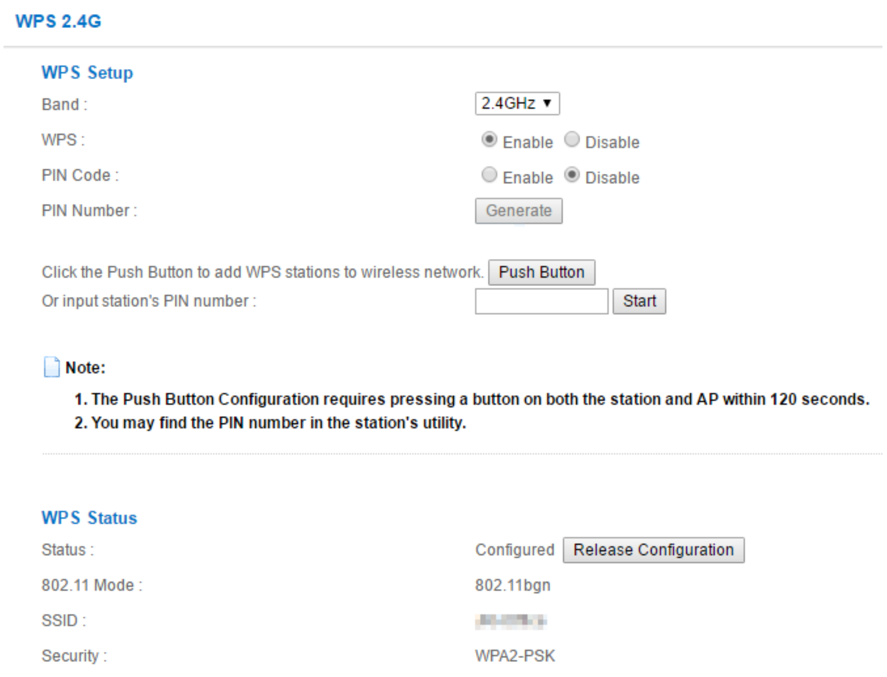
Wireless Setup

Active :	☐
Name (SSID) :	ZyXEL_SSID1
☐ Hide SSID	
✓ Intra-BSS Traffic Blocking	
✓ WMM QoS	
☐ Enable Bandwidth Management for Guest WLAN	256 (max. bandwidth kbps)

Security

Security Mode :	WPA2-PSK
☐ WPA-PSK Compatible	
☐ PMF	
Pre-Shared Key	•••••••••••••

Figure 2: ZyXEL guest network options [3]



WPS 2.4G

WPS Setup

Band :	2.4GHz
WPS :	[✓ Enable] Disable
PIN Code :	Enable [✓ Disable]
PIN Number :	Generate

Click the Push Button to add WPS stations to wireless network. Push Button
Or input station's PIN number : Start

Note:
1. The Push Button Configuration requires pressing a button on both the station and AP within 120 seconds.
2. You may find the PIN number in the station's utility.

WPS Status

Status :	Configured Release Configuration
802.11 Mode :	802.11bgn
SSID :	[illegible]
Security :	WPA2-PSK

Figure 3: ZyXEL WPS configuration [3]

Depending on manufacturer and device these solutions might offer different feature sets. In the example above the device offers possibilities for activating the guest network for a set amount of time (Figure 1) as well as restricting the SSID's bandwidth (Figure 2). Further it allows the registration of devices to **one** specific SSID using WPS (Figure 3).

Other approaches apart WPA2-PSK involve the usage of WPA2-Enterprise (802.1x port-based authentication). Currently existing software projects and solutions in this context can basically be categorised into three classes: Captive Portals, Radius Management and Credential/Certificate Distribution solutions.

Captive Portals

Captive Portals are probably the best known of those three categories, since these are solutions most users come into contact with. They are mainly used in public places wherever an open WiFi hotspot is available. These solutions allow the redirection of a connecting HTTP client to a sort of gateway web-page, where users are prompted access conditions, terms of use or simply asked to log-in with a before acquired username and password. Some of the best known Captive portals[1] are probably:

- HotspotSystem,
- WiFiDog and
- CoovaChilli/Chillispot,

since these are available out of the box on devices running DD-WRT and are also part of the OpenWrt repository, they are therefore easy to implement in a home environment.
Contrary to the proposed solution of this thesis, solutions of this kind neither provide the full extent of security provided by a full fledged 802.1x authentication system, nor a simple enough user and/or guest management to accommodate home users, or access to extended features provided by RADIUS. Although some proprietary hotspot solutions like HotspotSystem Basic+[2] or Socialwave[3] provide similar features for guest user support they are still not viable solutions for simple and small home networks.

[1]List of Captive Portals: `https://goo.gl/OyIgbb`
[2]HotspotSystem: `http://www.hotspotsystem.com`
[3]Socialwave: `http://www.socialwave.at`

RADIUS Management

RADIUS represents the server backend and central point for authenticating and authorising users in an 802.1x secured network. Therefore it would be desirable to manage it's features as simple as possible. Management interfaces like FreeRADIUS Dialup Admin[4], phpRADmin[5] or daloRADIUS[6] are foremost created for experienced administrators and simply provide the ability to manage various aspects of a RADIUS server. This can reach from only being able to manage users, accounting (logging) and server preferences (FreeRADIUS Dialup Admin), to providing more advanced features like group management, billing information and more granular access control (daloRADIUS).
Taking into account that solutions of this kind are rather complicated to manage and only provide a fraction of the features that this work wants to offer to a home user, this kind of software is also not a viable choice for the proposed area of application.

Credential/Certificate Distribution

This category of software provides the end-user with an automated configuration of certificates and user credentials for joining and configuring access to 802.1x and WPA2-Enterprise networks. Products like SU1X[7] or Cloudpath Enrollment System[8] accomplish this by installing their own deployment tools or mobile apps on the respective clients. Although these solutions offer a rather easy way to access 802.1x networks from a wide range of devices and operating systems, they are mostly proprietary and designed for larger company or educational networks and are simply not designed to run on consumer hardware. Also the fact that a third party application has to be installed to join these networks makes it not applicable for this work's target audience.

Apart from those three categories of software solutions, there are a few research projects which tried to unify most of the above mentioned aspects and extend them with additional functionality. The paper "A Secure Wireless LAN Access Technique for Home Network" for example suggests the creation of a custom protocol, based on EAP, to transport several authentication methods using passwords which are changed randomly and periodically based on a counter for each authentication method and password [4]. Some also propose the extension of EAP to support the implementation

[4]Dialup Admin: `http://freeradius.org/dialupadmin.html`
[5]phpRADmin: `https://sourceforge.net/projects/phpradmin/`
[6]daloRADIUS: `http://www.daloradius.com/`
[7]SU1X: `https://sourceforge.net/projects/su1x/`, `https://goo.gl/pyH0xz`
[8]Cloudpath ES: `https://goo.gl/PSScVN`

of an Intrusion Detection System (IDS), which is not only applicable for home networks but 802.1x secured wireless networks in general [5]. Finally “A Secure Wireless LAN Access Technique for Home Network” suggests a rather consumer friendly 802.1x authentication method using username and password on a dedicated SSID. This solution involves the use of a captive portal and the execution of an applet (based on SU1X mentioned above) to distribute credentials and enrol new users [6].

3. Employed Technologies and Tools

This part of the thesis will focus on discussing details about the basis technologies and tools used to implement the in Chapter 1.1. proposed 802.1x/RADIUS authentication solution. It represents a comprehensive collection of technologies used by the implementation of this project, which wasn't found this way in any literature and therefore will be collected and discussed in this chapter. This involves the obvious components such as OpenWrt and Radius as well as the Extensible Authentication Protocol (EAP) which provides the basis of port-based authentication.

3.1. OpenWrt

OpenWrt is a Linux based operating system for embedded devices, usually wireless routers, which is stripped to the bare minimum to realise distribution on a wide range of systems and hardware with a very limited amount of resources regarding CPU speed, memory and disk space (flash memory)[9]. Although it is such a minimalistic system it provides the user with full access to the filesystem and an optional package management, which allows a free customization of the system to ones individual needs. Since the project is entirely open-source and heavily community driven OpenWrt, at this point, supports 50 platforms (with the instruction sets AVR32, ARM, CRIS, m68k, MIPS, PowerPC, SPARC, SuperH, Ubicom32, x86, x86-64) [8] and offers over 3500 software packages through the `opkg` (Open Package Management) package management system. To allow the user an as easy as possible configuration of the system's most important settings OpenWrt provides the so called UCI system [9] (see section 3.1.4.). This represents the command-line variant of the subsystem, as well as a web user interface (as of Version 8.09) which is based on LuCI [10], a framework written in Lua. Apart from this OpenWrt also provides an extensive build system to enable developers to easily cross-compile their software, generate cross-compile toolchains and build customized firmware images.

[9]As of the release of OpenWrt 12.09 Attitude Adjustment devices with less than 16MB RAM are no longer supported [7].

3.1.1. Architecture

To provide a better understanding of an Embedded Linux System architecture and OpenWrt's architecture in specific, this section provides an overview of such a system's primary components. This kind of systems basically consist of the following elements (also refer to Table 1):

- build system
- bootloaders
- "current" mainline Linux Kernel[10]
- C Libraries
- userspace (Busybox[11], package manager, UCI, ...)

UCI	OPKG	User Programs
BusyBox		
C Libraries		
Linux Kernel		
Bootloader		

Table 1: OpenWrt's architecture and system components

3.1.2. Build System

OpenWrt's Build system is a set of Makefiles that allow users to generate both a cross-compilation toolchain and a root filesystem ("firmware images") for embedded systems [11]. The goal of the build system is to automate the process of configuring and compiling your own software for different target systems by providing the instruction set architectures of several embedded systems. To

[10]kernel version 3.18.23 in OpenWrt Chaos Calmer 15.05
[11]Busybox Website: `https://busybox.net/`

accomplish this OpenWrt provides a Makefile system with it's own syntax and therefore differs from the traditional Linux Makefiles[12]. In the context of OpenWrt a Makefile contains meta information about a package, how to compile, where to install binaries and copy files, and several other options [11].
The compilation toolchain is a set of tools used to compile code and consists of:

- a compiler (usually *gcc*)
- binary utilities like assembler and linker (usually *binutils*)
- a standard C library (most Linux systems use *gLibc*, but *musl-libc* and *uClibc* are also a possibility)

The buildroot itself provides four directories (*tools, toolchain, package and target*) to realise the system's build sequence as follows [11]:

tools	automake, autoconf, cmake, ...
toolchain/binutils	as (assembler), ld (linker), ...
toolchain/gcc	gcc, g++, cpp, ...
target/linux	kernel modules
package	core and additional "feed" packages
target/linux	kernel image
target/linux/image	firmware image generation

Above mentioned "feed packages" provide the user with the ability to supply his own set of packages using Subversion, Git or directly within the according buildroot directory. By doing so they will already appear as a package when running *make menuconfig* (Linux Kernel menuconfig). The user may then choose to compile and build them directly into a firmware image or as a separate package for later installation with `opkg` [12].
For further information about installation and usage regarding the build system please refer to the official documentation[13].

3.1.3. Flash Memory Layout

Since almost all embedded systems or in this case routers don't have any mass storage, those systems rely on flash memory to store the operating system and additional data. Generally two

[12]These Makefiles can still be executed using `make` but provide additional flags and functionality for the OpenWrt buildroot and toolchain, such as meta data about packages.
[13]OpenWrt Build System HowTo: `https://wiki.openwrt.org/doc/howto/build`

kinds of flash memory are employed to realise this storage. The typically smaller raw NOR flash (4MB - 16MB used in older routers) and the larger raw NAND flash memories (32MB - 256MB used in newer devices). This raw flash storage isn't partitioned in the traditional sense, where information about the partitions is stored within the Master Boot Record (MBR), but rather done by the Linux Kernel or the bootloader (for OpenWrt's generic flash layout see Figure 4). Partitions are simply addressed by offsets within the raw flash, but can also be named to avoid the repeated use of start offsets [13].

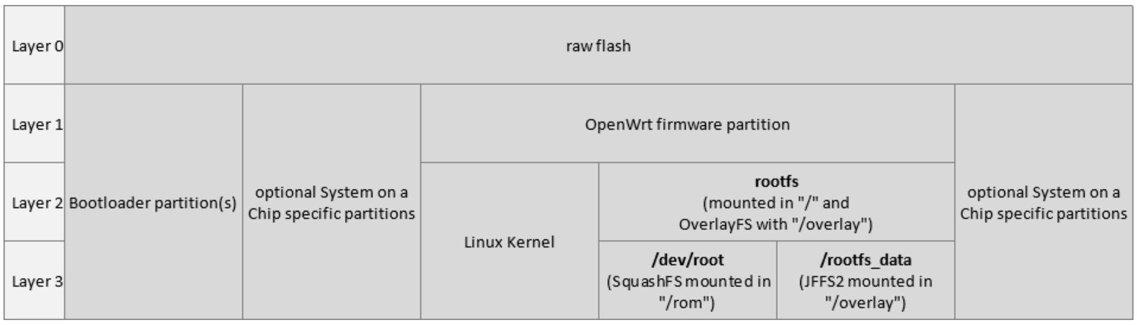

Layer 0
raw flash
Layer 1
OpenWrt firmware partition
Layer 2
Bootloader partition(s)
optional System on a Chip specific partitions
rootfs (mounted in "/" and OverlayFS with "/overlay")
optional System on a Chip specific partitions
Layer 3
Linux Kernel
/dev/root (SquashFS mounted in "/rom")
/rootfs_data (JFFS2 mounted in "/overlay")

Figure 4: OpenWrt's generic flash layout [13]

Most newer routers share this generic partition scheme, which slightly differs depending on the U-Boot implementation and System on a Chip (SoC) specific images.
As shown in Figure 4 the partitions are nested and sectioned into four layers:

Layer 0 describes the raw flash chip which is connected to the SoC [13].

Layer 1 partitions the flash space into several parts for [13]:

1. one or more partitions for U-Boot depending on implementation (usually `u-boot` and `u-boot-env`)
2. an OpenWrt `firmware` partition
3. optional SoC partitions for specific firmware

Layer 2 further divides the `firmware` partition into [13]:

1. `kernel` space, which contains a twice packed Kernel binary (first with LZMA and then again with `gzip` to save as much space as possible) that is written directly to the raw flash and not part of any filesystem.
   This binary is decompressed and loaded into RAM at boot time.
2. the `rootfs` which contains the filesystem mounted on "/"

Layer 3 again divides `rootfs` into two partitions [13]:

1. Read only memory partition ("`/dev/root`") using SquashFS which is mounted into "`/rom`"
2. writable `rootfs_data` partition using JFFS2 which holds space for additional data and changes to existing files mounted into "`/overlay`"

The entire root filesystem is mounted into "`/`" and is comprised of "`/rom`" and "`/overlay`", which can therefore be ignored and only "`/`" should be used for changes to the system. The ROM partition mounted in "`/rom`" contains all basic files depending on the selected packages when building the firmware image. It also provides default configuration files for FailSafe booting[14]. Since this partition is using SquashFS[15] none of the data in this mount point can be deleted or changed in any way. Due to the use of OverlayFS [15] all changes to existing files and additional files which are added to the device after installation are written to the writable JFFS2 partition mounted in "`/overlay`" which is merged with "`/rom`" to form the before mentioned "`/`". Therefore whenever the system is looking for a file, "`/overlay`" is checked first and only if the file doesn't exist in this partition it checks in "`/rom`". This way "`/overlay`" simply overrides "`/rom`" and creates the illusion of a writable "`/`" (see Figure 5) [13].

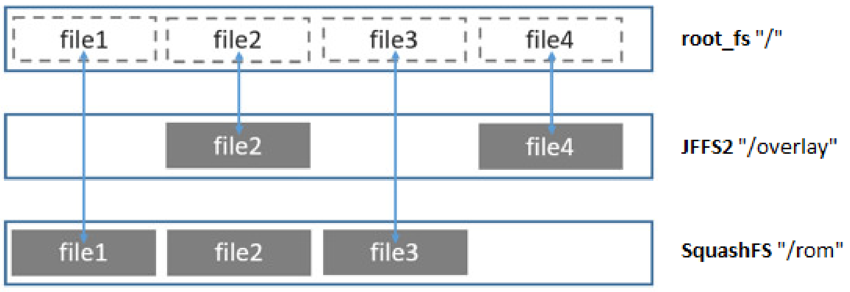


Figure 5: OverlayFS construct [16]

When a file is to be deleted that is actually within "`/rom`" a corresponding entry in "`/overlay`" is created instead, these entries are so called whiteouts which are symlinks that behave like the file doesn't exist [13].

[14]Booting the system with a necessary set of default settings to recover from failure.

[15]SquashFS is a read only compressed filesystem, which uses LZMA for the compression. Since SquashFS is a read only filesystem, it doesn't need to align the data, allowing it to pack the files tighter thus taking up significantly less space than JFFS2 (saving 20-30% over a JFFS2 filesystem) [14].

3.1.4. The Unified Configuration Interface (UCI) System

The UCI system is intended to provide a centralized and unified interface for configuring OpenWrt's basic and most important system settings. Typically these are crucial settings for the functioning of the device such as the main network interface, wireless, remote access, firewall and logging settings. In addition to the basic system settings some third party programs such as `samba` or the system's default web server `uhttpd` have been made compatible with this system to facilitate an easier way of configuring those services. The central configuration is split into several files which are located within "`/etc/config`". Each of the files contained in this directory relates to a portion of the system which it configures. OpenWrt offers a variety of comfortable ways to modify these files like the command line utility *uci* and various APIs (e.g. shell, lua, C), which is how the system's web interface LuCI parses and alters UCI configuration files [9].
UCI configuration files typically consist of `config` statements, so called sections, and several option and/or list statements defining the actual values (see example below).

```
config 'foo' 'bar'
        option   'string_var'  'value'
        option   'boolean_var' '1'
        list     'collection'  'item_1'
        list     'collection'  'item_2'
```

Listing 1: UCI configuration file example

- The Statement `config 'foo' 'bar'` defines a new section of the type `'foo'` and the name `'bar'`. These sections can be anonymous as well by only providing a type and no identifier. Although the identifier being optional a section's type is mandatory for processing programs to decide how to handle the enclosed options.
- The lines `option 'string_var' 'value'` and `option 'boolean_var' '1'` simply define values within the defined section.
- Statements starting with the keyword `list` define options with multiple values. Therefore all values of `list` statements with the same identifier will be merged into a single list of values.

Command Line Utility (uci)

The command line utility `uci` provides a simple and comfortable way of parsing and modifying UCI configuration files for developers and scripting purposes. UCI allows several operations to the configuration files by simply addressing the corresponding file, section and option (`uci <command> <config>.<section>.<option>`). After committing any changes to a configuration file the affected service has to be restarted for the changes to take effect.
The following example shows retrieving and setting values to a configuration file:

```
//show all wireless interfaces
root@OpenWrt:~# uci show wireless | grep wifi-iface
wireless.@wifi-iface[0]=wifi-iface
wireless.@wifi-iface[0].device='radio0'
wireless.@wifi-iface[0].network='lan'
wireless.@wifi-iface[0].mode='ap'
wireless.@wifi-iface[0].ssid='OpenWrt5Ghz'
wireless.@wifi-iface[0].encryption='none'
wireless.@wifi-iface[1]=wifi-iface
wireless.@wifi-iface[1].device='radio1'
wireless.@wifi-iface[1].network='lan'
wireless.@wifi-iface[1].mode='ap'
wireless.@wifi-iface[1].encryption='psk2'
wireless.@wifi-iface[1].key='some_password'
wireless.@wifi-iface[1].ssid='OpenWrt'

//set ssid of wireless interface 2
root@OpenWrt:~# uci set wireless.@wifi-iface[1].ssid=test
root@OpenWrt:~# uci commit wireless

//restart wireless devices
root@OpenWrt:~# wifi

//retrieve the newly set ssid
root@OpenWrt:~# uci get wireless.@wifi-iface[1].ssid
test
```

Listing 2: UCI example

If there are multiple anonymous rules (e.g multiple wireless interfaces as shown in Listing 2), UCI allows them to be referenced in an array-like manner. Whereas [0] returns the first and the usage of [-1] the last one and so on. For a full set of commands please refer to the official `uci` documentation and the documentation of the corresponding configuration files [9].

3.2. Port-Based Authentication - 802.1x

The following sections will give an overview of how an 802.1x port-based authentication system is comprised, which standards and protocols are employed and how all those parts play together. The complexity of such a system is the result of interoperability issues between all the components used and the fact that no single standard defines all of those components. The standards and specifications used by 802.1x are written by two different standardisation organisations, the Institute of Electrical and Electronic Engineers (IEEE) which supplies a standard for EAPoL [17] (Extensible Authentication Protocol over LAN) and the Internet Engineering Task Force (IETF) who provide the RFCs for EAP [18] (and it's methods) as well as Radius [19].
In the context of port-based authentication the term "port" has to be understood, which represents a Layer 2 (Data Link Layer) connection to a network. In a wired network this refers to a physical port on a switch as shown in Figure 6. While the physical connection is provided by an Ethernet cable connected to a switch's Ethernet port (Layer 1: Physical Layer), port-based authentication attempts to verify the identity of the connected device via Layer 2 [20].

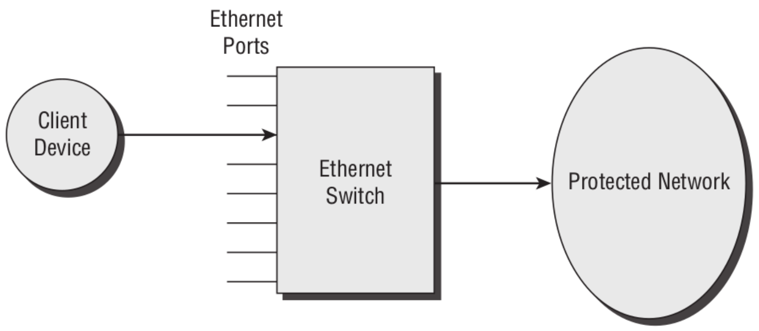


Figure 6: Port-based Authentication via physical link [20]

Ports and therefore the IEEE standard for EAPoL also apply to wireless networks, but in a wireless LAN the "port" is an association of a client with an access point rather than a physical connection (see Figure 7). Every access point within a wireless network periodically broadcasts a so called 802.11 beacon frame. When a wireless client boots up the surrounding area is scanned for active access points using those beacon frames. The client then tries to associate with an access point which involves a series of 802.11 frame transmissions between the client and the access point [21] as shown in Figure 8.

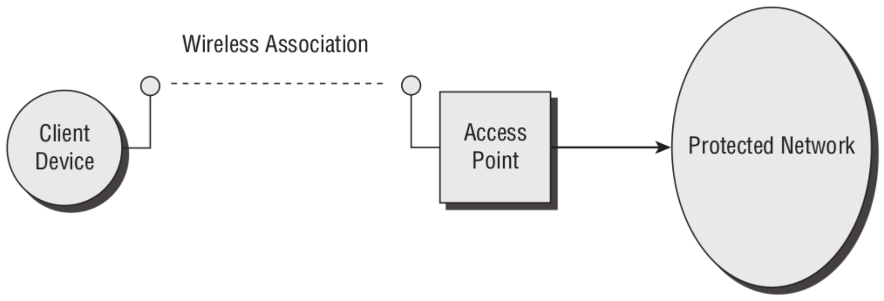


Figure 7: Wireless LAN virtual link [20]

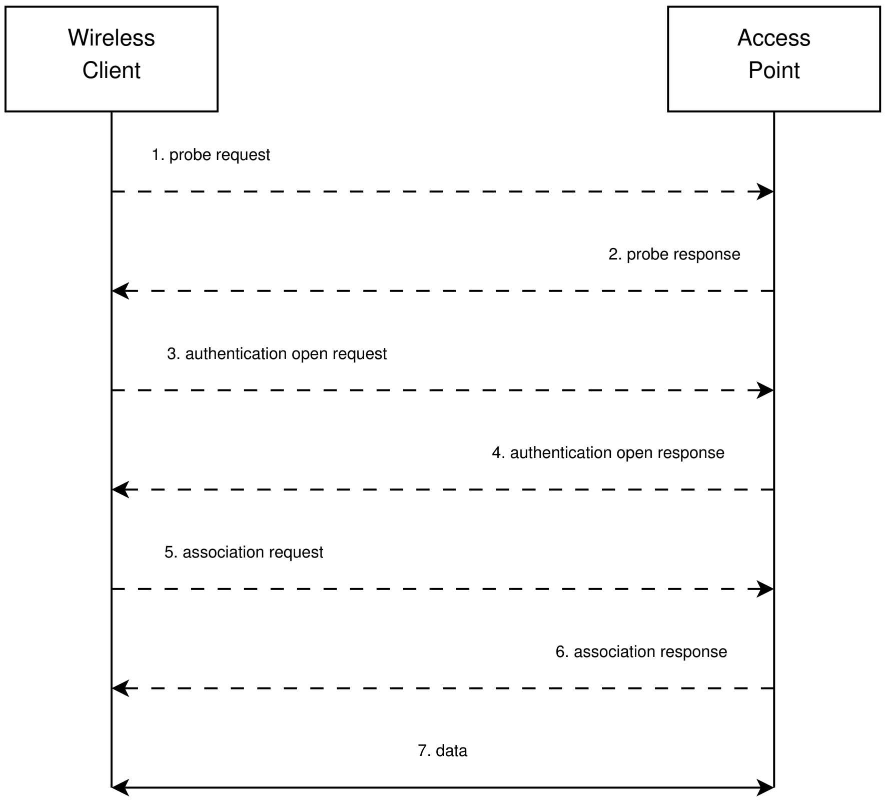


Figure 8: 802.11 association process

After successful connection and authentication to the network, the authentication server grants the device access and certain privileges (depending on his credentials) to the protected network (authorization). Although introducing port-based authentication to a wired or wireless network is a big step towards securing these networks, methods such as data encryption, intrusion detection and access control should still be employed in order to prevent possible attacks against the network. In addition to preventing unauthorized users from accessing the network, a 802.1x port-based authentication system also supports the following features [20]:

- **User location:** Applications can easily track the whereabouts of a user or device based on the switch or access point where the corresponding client was authenticated.
- **Billing and accounting:** Combining port-based authentication with billing and accounting mechanisms enables the implementation of fee-based network access (mostly used by hotel solutions or Internet service providers).
- **Personalized access:** Based on the credentials the system grants the user access to certain parts of the network or applications.

3.2.1. Primary Components

Until now port-based authentication systems were shown from a rather generic perspective, but in fact there are a number of components and protocols found in the 802.1x specifications which are involved in realising such systems. As Figure 9 constitutes, this kind of authentication system primarily consists of supplicants, authenticators and authentication servers.

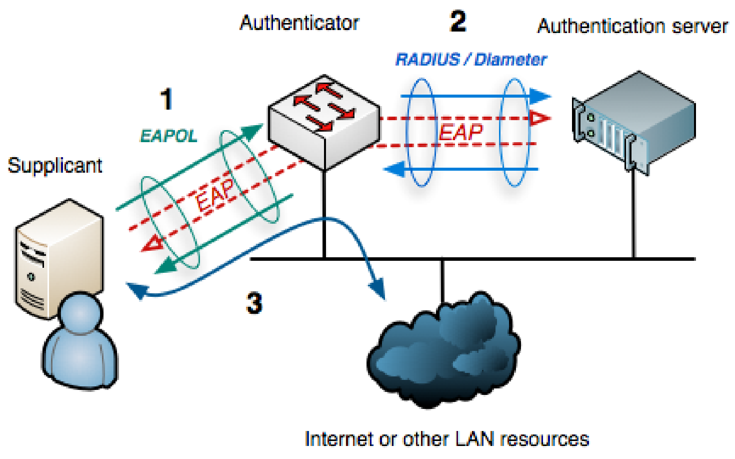


Figure 9: 802.1x primary components

Supplicant is a client device that requests access to a network and needs to be authenticated. To be recognized as a valid supplicant the client has to implement 802.1x and a certain EAP-Method (further details on EAP and EAP-Methods can be found in Chapter 3.4.). The communication between supplicant and authentication server is accomplished via EAP as transport protocol and a specific EAP-Method to facilitate the actual authentication mechanism (see Section 3.2.2.3.).
Whereas the supplicant and authenticator are communicating via 802.1x EAPoL which encapsulates the EAP-Method frames as data (see Section 3.2.2.1.).

Authenticator The authenticator is a Layer 2 device, usually a switch or access point. It functions as a "security gate" between the supplicant and the protected network, blocking all traffic except EAPoL encapsulated frames until the authentication process is complete (Figure 10[16]). This device is also responsible for the communication between supplicant and authentication server by taking the EAP-Method data from the EAPoL frames and encapsulating it into RADIUS frames (see Section 3.2.2.2.).

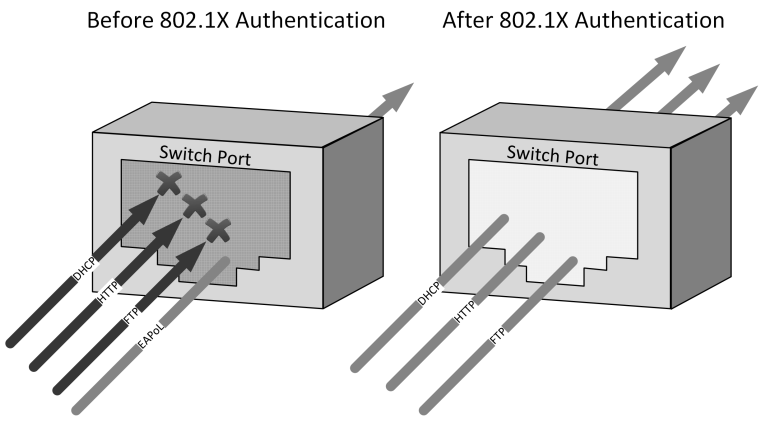


Figure 10: 802.1x port control [22]

[16]AAA and 802.1X Authentication:
`https://networklessons.com/security/aaa-802-1x-authentication/`

Authentication Server	As described above the supplicant and authenticator communicate about the authentication, where at some point the authentication server will request credentials from the supplicant. According to the credentials offered by the supplicant the authentication server then either grant or deny the client access to the protected network (see Section 3.2.2.). The 802.1x standards and specifications don't demand any particular type of authentication server, but nearly all solutions use RADIUS.

3.2.2. Authentication Process

A typical authentication progression passes through the following steps (also see Figure 11):

1. **Initialization:** On connection of a new client/supplicant the physical port on the switch is set active and put into an unauthorized state. As long as the port is in this state only 802.1x traffic is allowed and passed on, all other traffic will be dropped.

2. **Initiation:** During this phase the authenticator periodically sends EAP Identity request frames to a multicast MAC address, which the supplicants listen to. Upon receiving said frame the client responds with an EAP Identity response frame which contains an identifier for the supplicant (e.g. User ID). The authenticator then further encapsulates this EAP Identity response frame in a RADIUS Access-Request packet which is forwarded to the authentication server.
   This process can also be initiated by the supplicant with the sending of an EAPoL-Start frame to the corresponding authenticator, which in turn responds with an EAP Identity request frame.

3. **Negotiation:** After receiving former mentioned RADIUS Access-Request packet the authentication server sends a RADIUS Access-Challenge encapsulated reply to the authenticator, which contains an EAP authentication request specifying a certain EAP-Method (e.g. EAP-TLS). The authenticator then encapsulates the EAP authentication request in an EAPoL frame and forwards it to the supplicant, which can now use the specified EAP-Method.

4. **Authentication:** Once the authentication server and supplicant have agreed on an EAP-Method a series of EAP requests and responses are sent between them (relayed by the

authenticator), until the authentication server responds with either EAP-Success or EAP-Failure (encapsulated in the according RADIUS Accept-Access or RADIUS Accept-Reject packet). In case of successful authentication the switch port will be put into an “authorized” state and therefore allow normal traffic. This state persists until an EAP-Logoff message is sent by the supplicant and the port is put back into an “unauthorized” state.

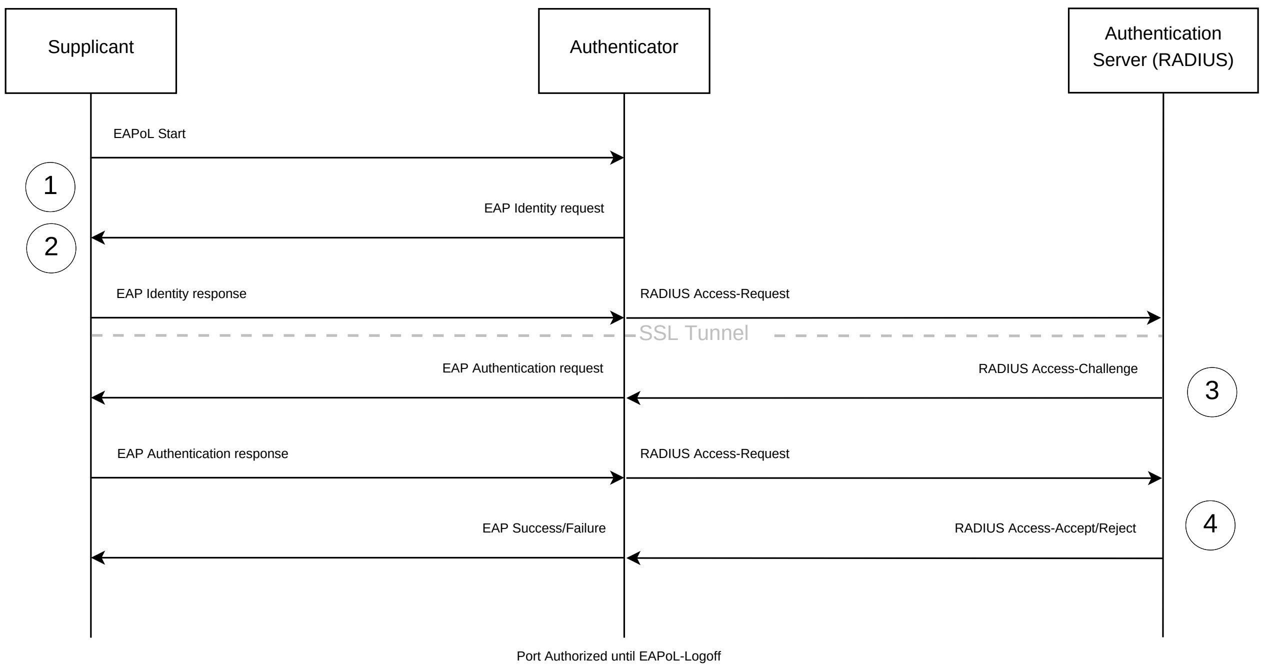


Figure 11: 802.1x authentication process

The following sections will discuss the communication between every single component (supplicant, authenticator and authentication server) in further detail.

3.2.2.1. Communication between Supplicant and Authenticator (802.1x)

The 802.1x standard (EAPoL) only applies to the communication between supplicant and authenticator and therefore is only a part of a 802.1x port-based authentication system, which also makes use of other protocols such as RADIUS. As defined in the 802.1x standard, EAPoL encapsulates EAP frames as data to facilitate the communication between supplicant and authenticator by adding several fields (Version, Type and Length) to EAP. Chapter 3.3. will explain this encapsulation in further detail. Figure 12 shows the communication explained in section 3.2.2. in further detail to illustrate the process and encapsulation of EAP frames.

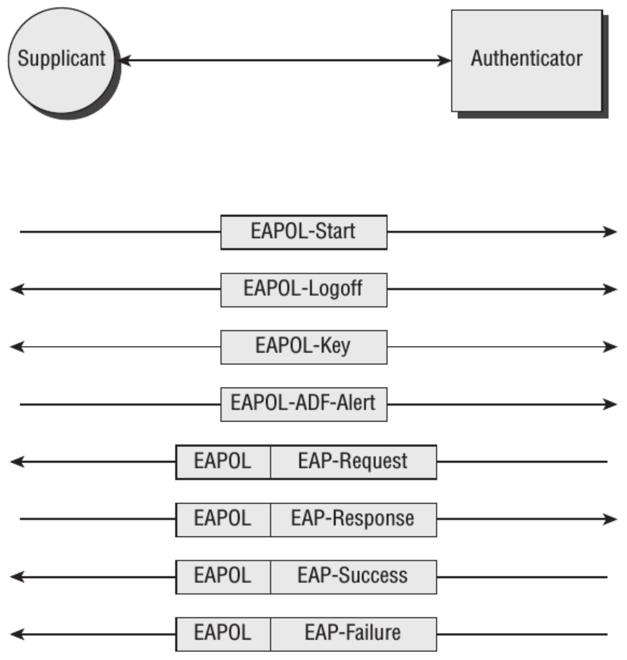


Figure 12: Communication between supplicant and authenticator[20]

By sending EAPoL frames of type "0" the involved devices signal that this frame is carrying an EAP frame as data. Whether the recipient is the supplicant or the authenticator the receiving device simply removes the EAPoL header and processes the encapsulated EAP frame according to its type which may be one of the following:

- Request
- Response
- Success
- Failure

Whereby the supplicant can only issue response frames and the authenticator may only perform Request, Success and Failure frames.
If a link between a supplicant and the authenticator becomes active (after initial EAPoL communication is completed), the authenticator sends an EAP Identity Request to the supplicant. At this point only EAP traffic will be forwarded to the protected part of the network (see Figure 11). After confirming the supplicants identity, the supplicant and the authentication server will start communicating based on their negotiated EAP-Method. From here on out the authenticator merely acts as a translator between supplicant and authentication server until the authentication

server authorises or rejects the supplicant. This far only the communication involving EAPoL Type "0" frames for carrying EAP data was discussed. For establishing and maintaining the connection between supplicant and authenticator serveral other EAPoL frames outside the scope of EAP are necessary, including the following (as shown in Figure 12):

- EAPoL-Start
- EAPoL-Logoff
- EAPoL-Key
- EAPoL-ADF-Alert

To initiate the EAPoL connection the supplicant sends an EAPoL-Start frame, which causes the authenticator to immediately respond with an EAP Identity request. This has to be done by the supplicant since the authenticator has no means of detecting when the supplicant comes online. As mentioned earlier 802.1x (i.e. EAPoL) applies to Layer 2 in order to prevent the supplicant from connecting to the protected network [20]. To accomplish integration on Layer 2 level, 802.1x takes advantage of access control mechanisms offered by 802.1D, which defines MAC bridges (including Spanning Tree and Bridging) [23]. Since 802.1D is required in all 802 LANs (including Ethernet and Wi-Fi), 802.1x works with any kind of LAN. In addition 802.1x makes use of the addressing reserved for the 802.1D Spanning-Tree Protocol, in particular the group address 01:80:C2:00:00:03 is used to facilitate 802.1x communications[17]. This address is also referred to as 802.1x Port Access Entry (PAE) address. As a consequence all 802 based devices (NICs, switches, access points, ...) are equipped to receive and process frames sent to this group address[20].

3.2.2.2. Communication between Authenticator and Authentication Server (RADIUS)

The communication between authenticator and authentication server is achieved by using RADIUS (see Figure 11). This involves the following RADIUS frame types:

- Access-Request
- Access-Accept

[17]By using a packet sniffer and filtering for the MAC address 01:80:C2:00:00:03 all 802.1x communications can easily be traced.

- Access-Reject
- Access-Challenge

optional (if Accounting is configured):

- Accounting-Request
- Accounting-Response

To establish communication the authenticator takes the EAP-Method data from an EAPoL frame (sent by the supplicant), encapsulates it into a RADIUS Access-Request frame and passes it on to the authentication server. Upon receiving an Access-Request the authentication server checks if the authenticator's IP address and shared secret match the server's expected values. If those values match, the RADIUS Access-Request will be processed accordingly, if not the server simply remains silent and doesn't respond in any way. After processing the RADIUS Access-Request the authentication server sends EAP-Method data to the authenticator (encapsulated into a RADIUS Access-Challenge frame), which again the authenticator will relay to the supplicant via EAPoL. According to the outcome of the EAP-Method exchange between supplicant and authentication server, the server transmits the corresponding RADIUS Access-Accept or Access-Reject frame. Which in turn is converted into an EAPoL Success or Failure frame by the authenticator and forwarded to the supplicant.

3.2.2.3. Communication between Supplicant and Authentication Server (EAP-Method)

The communication of supplicant and authentication server realises the actual authentication process. As Figure 13 shows this is accomplished by exchanging EAP-Method data, which contain various elements, such as the supplicants credentials. Depending on the chosen credential type (username/password, encryption keys, ...), certain EAP-Method implementations are required, such as:

- MD5 challenge
- One-Time Passwords

Additionally there is a vast variety of proprietary and RFC-based Methods available (e.g. EAP-TLS, EAP-TTLS, LEAP, EAP-FAST, ...). Chapter 3.4. will discuss the most commonly used methods in further detail.

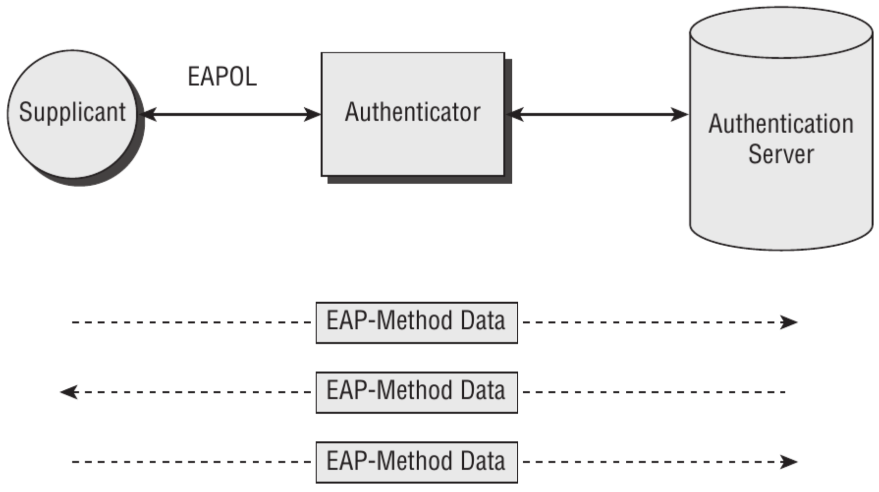


Figure 13: Communication between supplicant and authentication server[20]

3.3. EAPoL/EAPoW Protocol

The EAPoL protocol provides a Layer 2 communication between a supplicant and an authenticator to prevent the supplicant from connecting to a network protected by a port-based authentication system. As defined in the 802.1x standard [17] EAPoL provides mechanisms to realise EAP communications over LANs. To accomplish this, EAPoL adds additional headers to EAP packets in order to create specialised packet types, which transport these EAP packets as data. Figure 14 illustrates this encapsulation process within an 802.1x port-based authentication system.

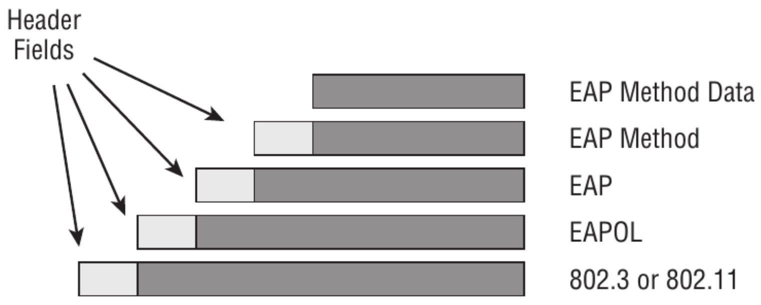


Figure 14: EAPoL encapsulation[20]

The primary objective of 802.1x communications is to transport EAP-Method data, which facilitates the actual authentication process. The by the supplicant and authentication server negotiated EAP-Method defines the actual EAP-Method data. EAP packets (typically Request and Response packets) in turn transport the EAP-Method protocol and data. EAPoL then carries the EAP packets, and 802.3 (Ethernet) or 802.11 (Wi-Fi) frames again transport the EAPoL packets [20]. Depending on the corresponding transport protocol we then either talk about:

- EAPoL (Extensible Authentication Protocol over LAN) or
- EAPoW (Extensible Authentication Protocol over Wireless)

3.3.1. Packet Structure

As already mentioned in Section 3.2.2.1., EAPoL adds three additional fields to an EAP packet as a header (see Figure 15). These header fields are essential for integrating with LANs and transporting EAP packets. The subsequent sections explain these fields in further detail.

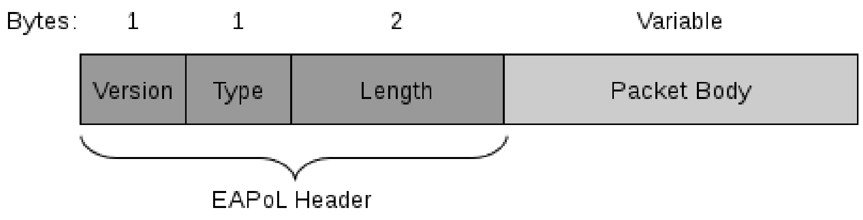


Figure 15: EAPoL packet structure

Version

The Version field identifies the EAPoL protocol version supported by the EAPoL packet's sender. The length of this field is one byte and for 802.1x implementations this field always contains the value "0000 0010"[18] (Hex "02").

Type

The Type field, with a length of one byte, describes the type of packet being sent by either the supplicant or authenticator. Table 2 shows the various types and their values. EAPoL doesn't define any other values at this point which allows for future extension.

[18]All values are unsigned.

Packet Type	Value (Hex)
EAP Packet	00
EAPoL-Start	01
EAPoL-Logoff	02
EAPoL-Key	03
EAPoL-ASF-Alert	04

Table 2: EAPoL Type field values

Length

The Length field provides two bytes for storing the length of the Packet Body. The value stored in this field represents the length of the payload in number of bytes. For example a value of "0000 0000 1010 1010" defines a Packet Body field size of 170 bytes. A zero value in this field signals that the according EAPoL packet doesn't contain any payload, which is the case with EAPoL-Start and Logoff packets. With the length of two bytes EAPoL could theoretically provide a payload size of 65535 bytes, which is restricted by the limitations of the according link transport protocol, i.e. Ethernet (IEEE 802.3) or Wi-Fi (802.11).

Packet Body

The Packet Body represents the payload portion of an EAPoL packet. This payload is only present in the following types of EAPoL packet types:

- EAP-Packet
- EAP-Key
- EAP-ASF-Alert

The Packet Body contains **exactly one** EAP packet if the type is EAP-Packet. For the EAPoL-Key type a **single** Key Descriptor and in the case of an EAP-ASF-Alert type, **exactly one** ASF-Alert packet.

3.4. Extensible Authentication Protocol (EAP)

The Extensible Authentication Protocol, or EAP, was originally designed for PPP (Point-to-Point Protocol)[24] to provide an additional authentication phase after the link was established. It is also a general-purpose authentication protocol which supports a number of authentication methods such as Kerberos, token/smart card, one-time passwords (OTP) and public key authentication [25]. Figure 16 shows a layered structure of other protocols involved with EAP and some of the most commonly used EAP-Methods in the Authentication Layer, including EAP-MD5, EAP-TLS, EAP-TTLS. For further details about these methods please refer to Section 3.4.3.

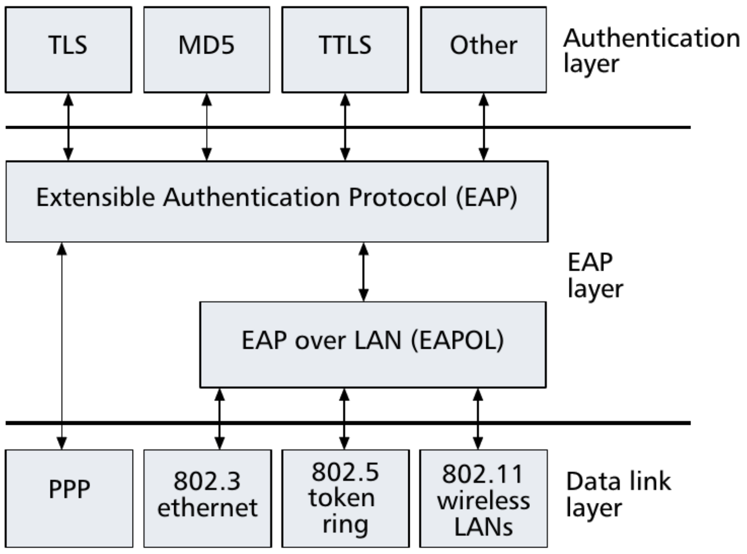


Figure 16: EAP layers[25]

3.4.1. Packet Structure

An EAP packet header (as shown in Figure 17) includes the following fields:

- Code
- Identifier
- Length
- Data

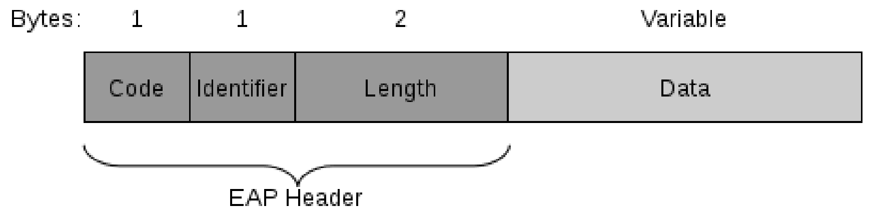


Figure 17: EAP packet header

The subsequent sections will further describe these field types and their values.

Code

Analog to EAPoL Type fields, the EAP Code field describes a one byte long type of an EAP packet. Table 3 shows the various types and their corresponding values. At this point EAP doesn't specify the use of any other values, which may be used for future extension. Refer to Section 3.4.2. for descriptions of the packet types.

Packet Type	Value (Hex)
EAP-Request[18]	01
EAP-Response[18]	02
EAP-Success[18]	03
EAP-Failure[18]	04
EAP-Initiate[26]	03
EAP-Finish[26]	04

Table 3: EAP Code field values

Identifier

The Identifier field (one byte length) facilitates a mechanism to match EAP-Response packets to their corresponding EAP-Requests. If the EAP-Request is sent by the authentication server to the supplicant the identifier value may be set to “0000 1010”, then the supplicant will in turn respond with an EAP-Response packet with it's identifier set to the same value. Additionally the authenticator uses the same identifier values when forwarding EAP packets. Therefore each transmission uses a new identifier value.

Length

The two byte long Length field contains the length of the entire EAP packet (including Code, Identifier, Length and Data fields). For example a length value of “0000 1000 0110 1101” indicates a total length of 2157 bytes, resulting in an EAP Data field of the length of 2153 bytes (2157 bytes minus 4 header bytes). Since the EAP Length field describes the length of the entire packet it's value is equal to the value of the EAPoL Length field, which in comparison only contains the length of its data field.

Data

Just as the EAPoL Data field this field has a variable length and may also be none existent depending on the packet's type. The value of the Code field (described above) defines how the Data field's value is to be interpreted.

3.4.2. Packet Types

EAP-Request/Response

Figure 18 illustrates the structure of EAP-Request and EAP-Response packets. Request packets are used by the authenticator to communicate with the supplicant to request the Identity of the supplicant or deliver EAP-Method data (see Figure 11). Response packets on the other hand serve the supplicant's communication with the authenticator to send EAP-Method data or credentials requested by the authentication server.

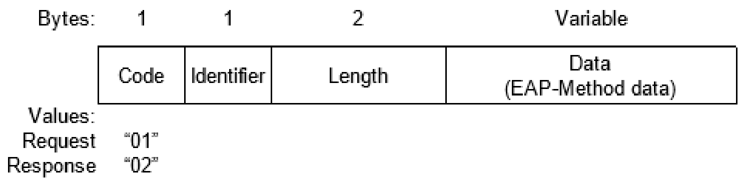


Figure 18: EAP-Request/Response packet format

EAP specifies a set of EAP types to define the structure of EAP-Request and EAP-Response packets. This EAP type dictates what data the EAP packet carries and is represented in the

EAP-Method packet's Type field (see Section 3.4.3.). Table 4 shows a set of standards-based EAP types which must be supported by **all** EAP implementations.

EAP Type	Value (Decimal)
Identity	1
Notification	2
NAK	3
MD5-Challenge	4

Table 4: Standards-based EAP types

For a full list of valid type values and their RFC references please refer to Table 7 in Appendix Section 7.1.

EAP-Success/Failure

Based on the outcome of the EAP-Method communication between supplicant and authentication server the authenticator my issue either EAP-Success or EAP-Failure packets to the supplicant. Figure 19 shows the packet format for Success and Failure packets. Since those packets simply function as notification packets the Length field in these packets is set to "00" (Data field length is zero bytes) and therefore don't contain an EAP Data field.

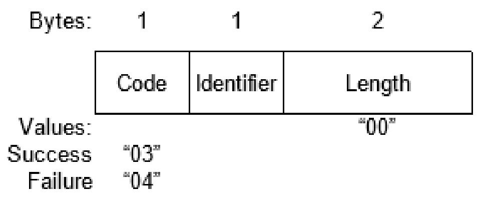


Figure 19: EAP-Success/Failure packet format

3.4.3. EAP-Methods

EAP-Methods communication represents the primary communication mechanism between a supplicant and an authentication server within a 802.1x port-based authentication system. Meaning that the EAP-Method realises the actual authentication process, whereas EAPoL and RADIUS merely serve the transport of EAP-Method data between the involved parties. The subsequent sections will discuss the packet structure as well as the most commonly used EAP-Method types, such as EAP-MD5, EAP-TLS, EAP-TTLS and PEAP.

3.4.3.1. Packet Structure

All EAP-Method packets consist of a two fields, a Type and Data field (see Figure 20).

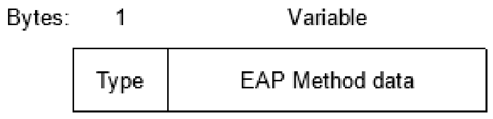


Figure 20: EAP Method packet format

Type

The one byte EAP-Method Type field describes a specific EAP-Method, some of which are defined by RFC 3748[18] (EAP specification) and have to be implemented by all EAP implementations (see Table 4). Apart from this standards-based EAP Types there are many other types that are proprietary and/or optional (see Table 7). The value of this Type field determines what the encapsulating EAP packet will carry. For Example a value of “21” indicates that the EAP-Method data is relevant to EAP-TTLS authentication processes.

Data

The EAP Data field contains the data for the corresponding EAP-Method type, such as credentials or certificates. For instance an EAP Data field of the Identity type (indicated by a value of “1” in the type field, see Tables 4 and 7) would carry credentials to implement an authentication process between supplicant and authenticator.

3.4.3.2. EAP-MD5

EAP-MD5 uses challenge handshake authentication protocol (CHAP) which represents a challenge-response process for the user authentication between supplicant and authentication server. This one-way authentication method allows for a simple authentication through the use of a user and password, where the password's MD5 hash is stored by the authentication server. The authentication process is implemented as a three-way handshake between supplicant and authentication server (see Figure 21) after receiving the initial RADIUS Access-Request and EAP Identity exchange between supplicant and authenticator (see Figure 11).

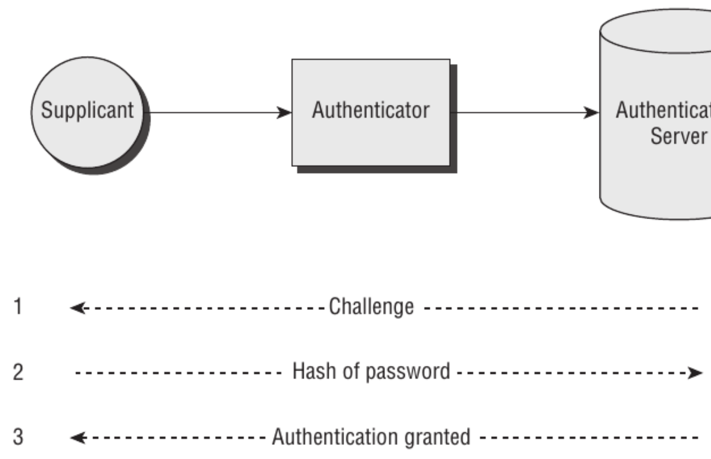


Figure 21: EAP-MD5 three-way handshake [20]

The authentication server requests the users password by sending a RADIUS Access-Challenge to the supplicant, which responds with the password hash using an EAP Authentication response which is further encapsulated into a Radius Access-Request. Upon receiving this request the authentication server either grants or denies the supplicant access to the protected part of the network.
Finding: From a security standpoint MD5 is not suitable for public and wireless networks since the password hashes can easily be sniffed, which allows for a derivation of the original password and therefore compromises the networks security [27]. Also this method doesn't provide neither server nor mutual authentication and supplicant authentication only via password hash (see Table 5). Therefore this method was deemed as unsuitable for the proposed scenario.

3.4.3.3. EAP-TLS

EAP-TLS (Transport Layer Security) provides a mutual authentication (opposing to EAP-MD5 which is only one-way), whereby both the supplicant and authentication server have to prove their identities. For this purpose EAP-TLS uses public key cryptography, which may involve smart cards, key tokens or digital certificates. The following and Figure 22 describe the EAP-TLS authentication process in more detail:

1. The authenticator sends an EAP Identity request packet to the supplicant.
2. in turn the supplicant responds with an EAP Identity response packet which contains the connecting client's user ID.
3. The authenticator then forwards this identity response to the authentication server by encapsulating it into a RADIUS Access request packet.
4. The server then sends an EAP-TLS start packet to the supplicant, which initiates a TLS Records client-server handshake and certificate exchange.
5. The supplicant responds with EAP-Response packets of the EAP-TLS type, with the data field containing one or more TLS records such as:
   - Client Hello (to initiate handshake)
   - Client certificate (optional)
   - Client Key exchange
   - Change cipher
   - Finished
6. According to the supplicants messages the authentication server sends EAP-Request packets to the client, containing a set of the following TLS records:
   - Server-Hello
   - TLS certificate
   - Client certificate request (optional)
   - Server done
   - Change cipher
   - Finished
7. After the server has sent it's last EAP request, the supplicant sends one more EAP-TLS message, which is empty, to signal the end of the handshake.
8. To conclude the handshake the authentication server finally sends an EAP-Success packet. Should any of the steps fail or an error occur, the server would have sent an EAP-Failure packet at the time the problem was detected.

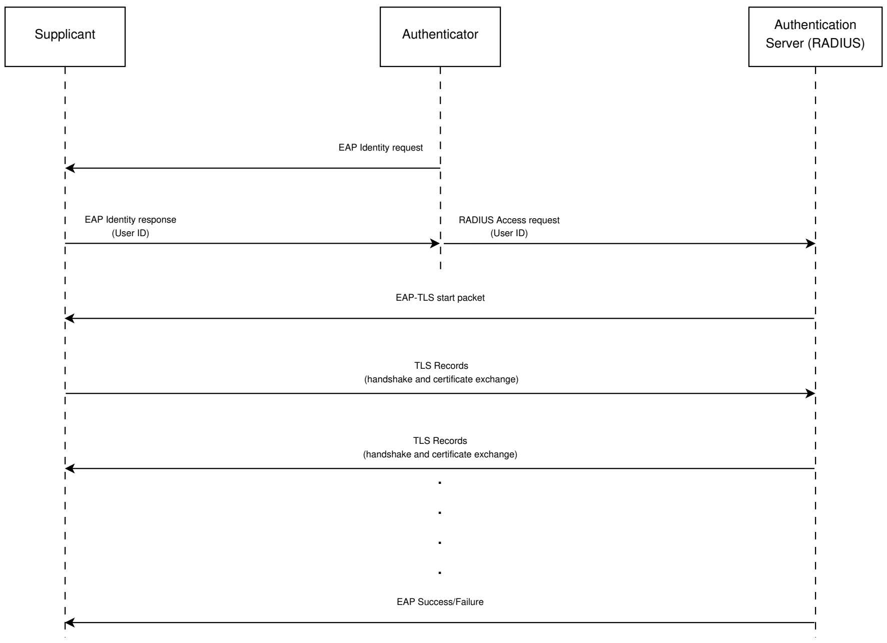

Supplicant
Authenticator
Authentication Server (RADIUS)
EAP Identity request
EAP Identity response (User ID)
RADIUS Access request (User ID)
EAP-TLS start packet
TLS Records (handshake and certificate exchange)
TLS Records (handshake and certificate exchange)
EAP Success/Failure

Figure 22: EAP-TLS authentication process

For further information about EAP-TLS and it's authentication process please refer to RFC 2716 (PPP EAP TLS Authentication Protocol)[28].
Finding: As a consequence of certificate usage on both the supplicants and the authentication server the administrative effort is very high in larger networks, since the certificates have to be deployed on every new device that needs to be integrated into the network. Also for a home network environment, the distribution of certificates for each device and especially guest users represents an unacceptable amount of effort needed by both the administrator and guest user alike to temporary register devices.

3.4.3.4. EAP-TTLS

EAP-Tunneled Transport Layer Security (EAP-TTLS) is similar to EAP-TLS, in the sense that both are certificate-based mutual authentication systems. However EAP-TTLS only requires a certificate on the server side. Supplicants can still be authenticated by using certificates (as with EAP-TLS) but usually authenticate themselves through a password, which greatly reduces the administrative effort needed to manage a port-based authentication system.
EAP-TTLS negotiations are comprised of two phases:

TLS handshake phase	authenticate the supplicant with the authentication server by using certificates (as described in Section 3.4.3.3.)

TLS tunnel phase	authentication of the supplicant using any non-EAP protocol [25]

Finding: Although this method already offers the possibility to authenticate the supplicant with username and password without the need for a client side certificate, it was dismissed due to the fact that EAP-TTLS wasn't supported natively by Microsoft until Windows 8.1. Devices running Windows XP, Vista, 7 and Windows Phone 8 do not support this authentication method without third-party software [29]. Therefore this method would greatly restrict the range of supported devices. Apart from these compatibility issues EAP-TTLS also has some significant security issues which allow for a number of attacks [30][31].
Further information about EAP-TTLS and it's versions can be found in RFC 5281 (EAP-TTLSv0)[32] and the Internet draft of EAP-TTLSv1 [33].

3.4.3.5. PEAP

Protected EAP is similar to EAP-TTLS and doesn't require certificates on the supplicants, only on the authentication server. Same as EAP-TTLS, it also comprised of two phases. In phase one a TLS session is negotiated and established between supplicant and authentication server (as described before). In the second phase (tunnel phase) all EAP messages are encrypted using the key negotiated in phase one. Hence the basic idea of EAP-TTLS and PEAP is identical, with the difference that PEAP is only capable of using EAP protocols (e.g. EAP-MS-CHAP-V2[19] [34]) whereas EAP-TTLS allows the usage of both EAP and non-EAP protocols [25]. When using this authentication method within an wireless LAN, typically the the authentication server will be authenticated by a supplicant based on the servers certificate and a secure TLS tunnel established. The supplicant itself is then authenticated by using username and password, which at this point is protected by the TLS tunnel [25].
Finding: This method will be employed in the implementation portion of this thesis, to provide the user with the capability of authentication via username and password (see Section 4.2.1.). PEAP is also supported by all established Microsoft operating systems and therefore offers the widest range of devices access to networks secured by this method.

[19]Encapsulates MS-CHAP-V2 within EAP

To conclude this section about EAP and it's various methods, Table 5 again provides an overview and comparison of the discussed authentication mechanisms.

	EAP-MD5	EAP-TLS	EAP-TTLS	PEAP
Server authentication	No	Public key (certificate)	Public key (certificate)	Public key (certificate)
Supplicant authentication	Password hash	Public key (certificate or token/smart card)	Certificate, EAP, non-EAP protocols	Certificate or EAP protocols
Mutual authentication	No	Yes	Yes	Yes
Dynamic key delivery	No	Yes	Yes	Yes
Basic protocol architecture	challenge / response	Establish TLS session and validate certificates for both client and server	1. Establish TLS between client and TTLS server 2. Exchange attribute - value pairs between client and server	1. Establish TLS between client and server 2. Run EAP exchanges over TLS tunnel
Server certificate	No	Required	Required	Required
Client certificate	No	Required	Optional	Optional
Protection of user identity	No	No	Yes, protected by TLS	Yes, protected by TLS

Table 5: Comparison of authentication mechanisms [25]

3.5. RADIUS

As already discussed in Chapter 3.2., RADIUS protocols present the primary communication mechanism between authenticator and authentication server by transporting EAP-Method data (see Chapter 3.4.) in a certain encrypted format. 802.1x and EAP don't necessarily require RADIUS as an authentication server, but since RADIUS is the most popular and is also used in the implementation part, this thesis will only focus on RADIUS as an authentication server for the port-based authentication system.

3.5.1. Packet Structure

All RADIUS packets share the same structure, which is made up of the five fields: Code, Identifier Length, Authenticator and RADIUS Attribure (see Figure 23).

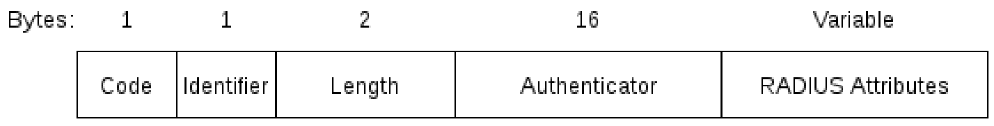


Figure 23: RADIUS packet structure

The subsequent sections describe those fields and their values in further detail.

Code

The one byte RADIUS code field describes the type of the RADIUS packet. Table 6 shows the various packet types and their corresponding values. At which "Accounting" packets are optional and are only sent if accounting is active/configured and "Status" packets are still in an experimental state [19].

Packet Type	Code
RADIUS Access-Request	1
RADIUS Access-Accept	2
RADIUS Access-Reject	3
RADIUS Accounting-Request	4
RADIUS Accounting-Response	5
RADIUS Access-Challenge	11
Status-Server	12
Status-Client	13
reserved	255

Table 6: RADIUS Code field values

Identifier

The RADIUS Identifier field is similar to the EAP Identifier field, which makes it possible to match RADIUS Access-Challenge packets to their corresponding Access-Request packets. The Identifier field of a RADIUS Access-Request packet sent by the authenticator may contain the value "0000 1010", as a result the authentication server's corresponding RADIUS Access-Challenge packet also holds this value within it's Identifier field. In addition the authenticator uses the same field value if the same Access-Request packet has to be retransmitted.

Length

The two byte long RADIUS Length field describes the length of the entire RADIUS packet including the fields Code, Identifier, Length, Authenticator and RADIUS Attribute. Just like EAP and EAPoL before, the total possible length of a RADIUS packet is restricted by the link transport protocol, i.e. Ethernet (IEEE 802.3) or Wi-Fi (802.11).

Authenticator

The Authenticator field contains a value according to the RADIUS packet type that is being sent, i.e. request (RADIUS Access-Request) or response (RADIUS Access-Accept/Reject, Access-Challenge) packets.

Authenticator Request	Contains a random 16 byte string, which is combined with the shared secret configured in the authenticator and authentication server and then put through a MD5 hash to again create a 16 byte value. This value is then XORed with the user's password. This result is then carried by the RADIUS Access-Request packet within the RADIUS Attribute field (User-Password attribute). Additionally the Request Authenticator value changes if the Identifier field is changed.
Authenticator Response	Is part of RADIUS Access-Accept/Reject and Access-Challenge packets. The Response Authenticator field is simply a MD5 hash of the entire corresponding RADIUS Access-Request packet plus the shared secret (see Figure 24).

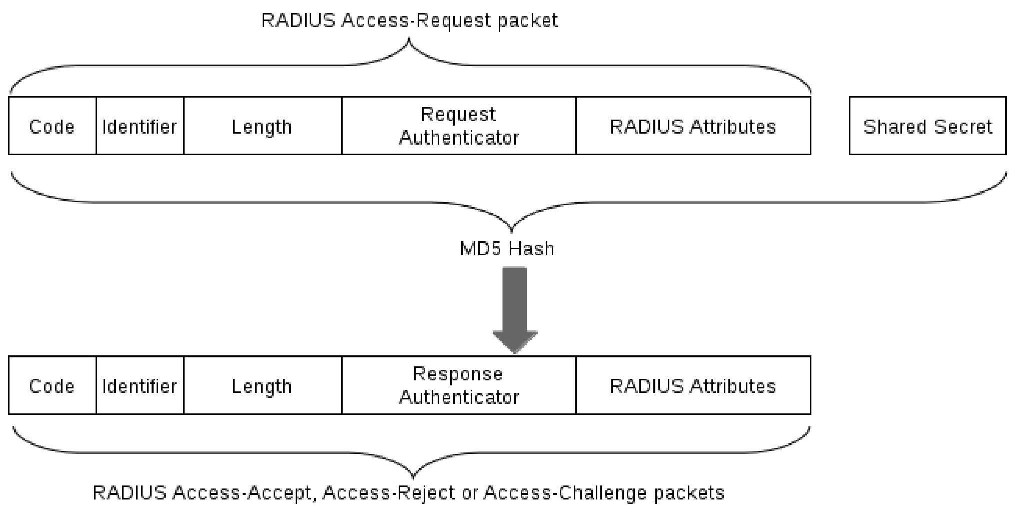


Figure 24: Authenticator field structure and content

Finding: Taking into account that Access-Requests encrypted using the configured shared secret and the connection is secured by TLS the password can be stored as cleartext on the server side. Further information about user and password storage can be found in Chapter 4..

Attribute

The RADIUS Attribute field contains the actual data communicated between the authentication server and authenticator. The RADIUS specification itself [19] and vendor documentation define a vast number of attributes used by RADIUS (see Table 8 in Appendix Section 7.2. and

RFC 2865 [19]). As shown in Figure 25 each Attribute field is again divided into three sub-fields: Type, Length and Value.

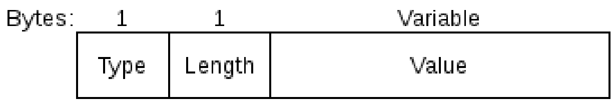


Figure 25: RADIUS Attribute field structure

Type Defines the type of content transported by the RADIUS Attribute field. As mentioned before RFC 2865 [19] defines a basic set of values for the Type field (see Table 8). Further types of attributes like the "EAP-Message" are defined by RFC 2284 [35] (PPP Extensible Authentication Protocol), which encapsulates EAP packets (and therefore EAP-Method data) sent between authentication server and authenticator.

Length The Length field contains the full length of the RADIUS Attribure field, including the length of the Type, Length and Value sub-fields.

Value Depending on the defined type the Value field may hold different kinds of data types, such as:

- Text (UTF-8 encoded characters)
- String (binary data)
- IP Address (32 bit value)
- Integer (32 bit unsigned value)
- Time (Unix time[20])

For further details about RADIUS protocols please refer to RFC 2865 (Remote Authentication Dial In User Service, RADIUS) [19].

[20]Number of seconds since 00:00:00 UTC, January 1, 1970

3.6. Dynamic VLANs

Dynamic VLANs or rather Dynamic VLAN Assignment allows the RADIUS server to dynamically assign a VLAN to a supplicant that requests 802.1x authentication through that server. The corresponding VLANs have to be configured on the RADIUS server via RADIUS attributes as well as on the authenticator (switch) to guarantee successful authentication of a supplicant. The RADIUS server attributes required to accomplish Dynamic VLAN Assignment are defined in RFC 2868 [36] (RADIUS Attributes for Tunnel Protocol Support), of which the following three attributes have to be configured by the authentication server to guarantee a functioning setup:

Tunnel-Type	RADIUS Attribute field type "64". Should be set to **VLAN**.
Tunnel-Medium-Type	RADIUS Attribute field type "65". Should be set to **802**[21].
Tunnel-Private-Group-Id	RADIUS Attribute field type "81". This value will be set to the corresponding **VLAN ID** or **VLAN name**.

Apart from letting RADIUS assign users dynamically to networks, the usage of VLANs would also allow the introduction of Quality of Service (QoS) for each VLAN through OpenWrt itself. This can be done by using OpenWrt's `qos scripts`. This package only represents one possibility of accomplishing QoS in OpenWrt, although `qos scripts` is the most comfortable one since it offers direct LuCI support (`luci-app-qos`) [37].
For more information about the configuration of Dynamic VLANs please refer to the documentation of the specific implementation of the RADIUS server, which in context of this work will be FreeRADIUS[22].

[21]On FreeRADIUS configurations this value is **IEEE-802**
[22]FreeRADIUS Documentation: `http://freeradius.org/doc/`

4. Implementation

This part of the thesis will focus on the implementation and the configuration within OpenWrt necessary to accommodate the in Section 1.1. proposed solution. Further this chapter discusses several problems encountered during the configuration and implementation as well as their impact on the final result. For this project it was decided to segment the network via VLANs rather than separate SSIDs for the individual networks to provide a more consistent administration and avoid user confusion by offering multiple SSIDs. This solution will therefore benefit from additional features offered by RADIUS as well as the security aspects of VLANs.

4.1. Dynamic VLAN and Guest network configuration in OpenWrt

One of the main features required for this solution is the ability to dynamically assign VLANs to users depending on their credentials. To this end the router operating system had to support dynamic VLAN assignment, which led to some restrictions on the hardware as well as of the software side of things. Although it was technically possible to enable dynamic VLAN support on OpenWrt, it wasn't natively included until the release of version 15.05 Chaos Claimer. Up to this release modifications to several system files regarding hostapd (authenticator for IEEE 802.11 networks) had to be made to enable this feature, which resulted in having to manually compile firmware images for the users corresponding hardware. For the full set of necessary changes please refer to OpenWrt changesets r43473[23], r42787[24], and r41872[25]. Due to the fact that neither a manual compilation of the firmware image nor the installation of a custom image would be a viable solution for the consumer, the minimum requirement of OpenWrt was simply set to version 15.05 to guarantee the easiest and most stable installation and configuration of the system. The configuration necessary to create a working dynamic VLAN and therefore guest network is spread across four different files within `/etc/config` provided by OpenWrt's Unified Configuration Interface System (see Section 3.1.4.):

- `network`
- `dhcp`
- `firewall`

[23]Changeset r43473: `https://dev.openwrt.org/changeset/43473/`
[24]Changeset r42787: `https://dev.openwrt.org/changeset/42787/`
[25]Changeset r41872: `https://dev.openwrt.org/changeset/41872/`

- `wireless`

The subsequent sections will describe the purpose of these files as well as the configuration needed to create an initial wireless guest network.

4.1.1. Network configuration

OpenWrt's network configuration can be found in `/etc/config/network`. It provides the ability of defining interface configurations, switch VLANs as well as network routes [9]. In order to generate a working dynamic VLAN environment, two types of sections provided by this file are needed. The `switch` or rather the `switch_vlan` subsections allow for partitioning the switch into VLANs and the `interface` section, which allows us to create logical networks that serve as containers for IP addresses, physical and virtual interfaces, aliases for firewall rules and so on.
The first step towards ensuring a working configuration of this kind is to modify the existing default VLAN (VLAN ID 1) called "`lan`" to tag the traffic going to the CPU port, which means, changing the default configuration of the corresponding `switch_vlan` section. This is necessary due to the fact that the virtual switch and therefore all switch ports are connected to the CPU through one physical port (see Figure 26).

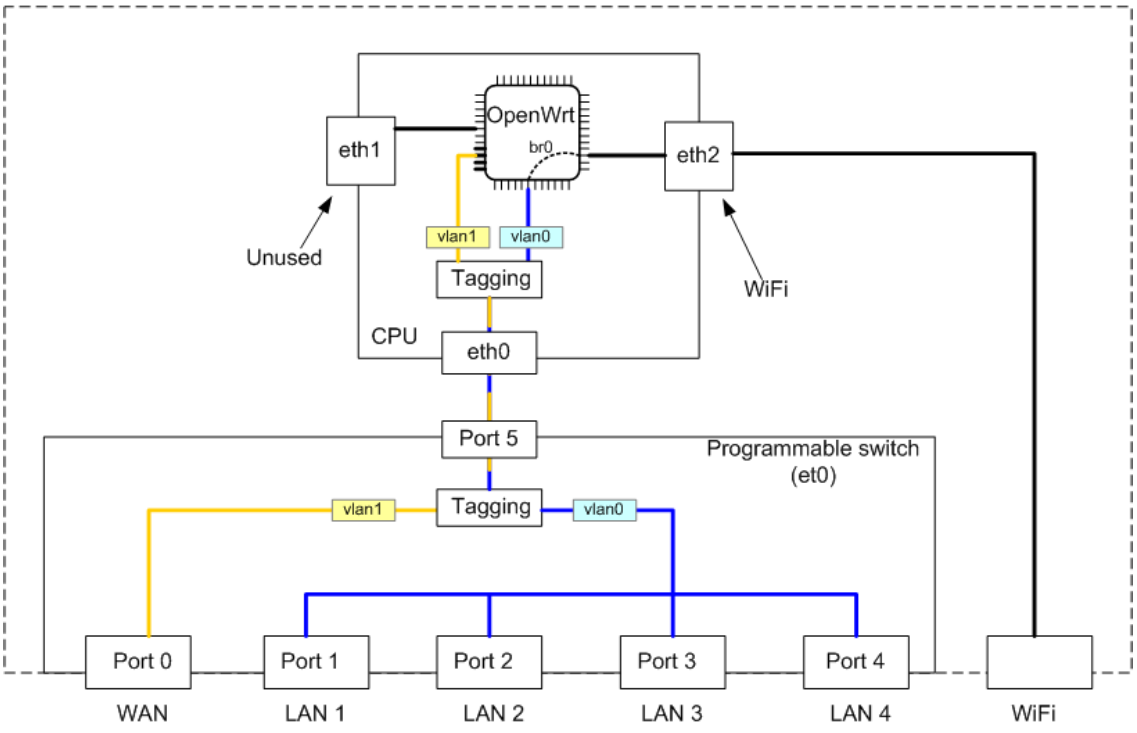


Figure 26: Router and switch structure [38]

The default network configuration file regarding this section looks like:

```
config switch_vlan
        option device 'switch0'     #corresponding virtual switch
        option vlan '1'             #VLAN ID
        option vid '1'              #VLAN ID (overrides "option vlan")
        option ports '0 1 2 3 4'    #ports to be (un)tagged, disabled
```

To tag the traffic towards the CPU port (in case of the development router[26] this is port 0[27]) this section needs to be changed to:

```
config switch_vlan
        option device 'switch0'     #corresponding virtual switch
        option vlan '1'             #VLAN ID
        option vid '1'              #VLAN ID (overrides "option vlan")
        option ports '0t 1 2 3 4'   #ports to be (un)tagged, disabled
```

NOTE: This port might already be tagged on some devices and therefore could be skipped if this is the case. On the test router (Zyxel NBG6716) this wasn't the case and port 0 had to be `'tagged'`.

Next the default interface itself needs to be changed to accommodate the now tagged VLAN traffic, therefore the physical interface (in this case eth0) needs to be converted into a virtual interface, which means changing the interface section with the default alias "lan" from:

```
config interface 'lan'
        option ifname 'eth0'            #physical or virtual interface
        option force_link '1'           #Specifies whether ip address,
                                        #route, and optionally gateway
                                        #are assigned to the interface
                                        #regardless of the link
                                        #being active
        option type 'bridge'            #If set to "bridge", a bridge
                                        #containing the given ifnames
```

[26] Zyxel NBG6716
[27] The number of the of the CPU port on the switch can vary between different routers but should usually be port 0.

```
                                               #is created
         option proto 'static'                 #Enables static adresses
         option netmask '255.255.255.0'        #Netmask
         option ipaddr '192.168.1.1'           #IP address
```

To a version where the interface represents the corresponding VLAN ID:

```
config interface 'vlan1'
         option ifname 'eth0.1'                #physical or virtual interface
         option force_link '1'                 #Specifies whether ip address,
                                               #route, and optionally gateway
                                               #are assigned to the interface
                                               #regardless of the link
                                               #being active
         option type 'bridge'                  #If set to "bridge", a bridge
                                               #containing the given ifnames
                                               #is created
         option proto 'static'                 #Enables static adresses
         option netmask '255.255.255.0'        #Netmask
         option ipaddr '192.168.1.1'           #IP address
```

Here two important changes have to be made to ensure that the 802.11 authenticator (hostapd) can match the default vlan and attach users to it. First the alias of the interface has to be changed from `lan` to `vlan1` and second the physical interface `eth0` is converted into a virtual interface to represent the VLAN ID, in this case `eth0.1`. Additionally the IP addresses for all VLANs were chosen in a way that the third octet represents the ID of the corresponding VLAN. For example in the case of a VLAN with ID 30, "`option ipaddr`" would be '`192.168.30.1`'.

NOTE: Due to the fact that the primary network interface was changed from `lan` to `vlan1` the DHCP (`/etc/config/dhcp` , see Listing 3) as well as the FIREWALL (`/etc/config/firewall`, see Listing 4) confgiuration have also be updated accordingly to avoid locking the user out of the router.

```
config dhcp 'vlan1'
        option interface 'vlan1'   #List of interfaces to listen on,
                                   #configured in network
                                   #configuration
        ...
```

Listing 3: Default DHCP configuration change

```
config zone
        option name 'vlan1'    #Unique zone name
        option network 'vlan1' #List of interfaces attached to zone
        ...

config forwarding
        option src 'vlan10'     #Specifies the traffic source zone
        option dest 'wan'       #Specifies the destination zone

...
```

Listing 4: Default Fireweall configuration change

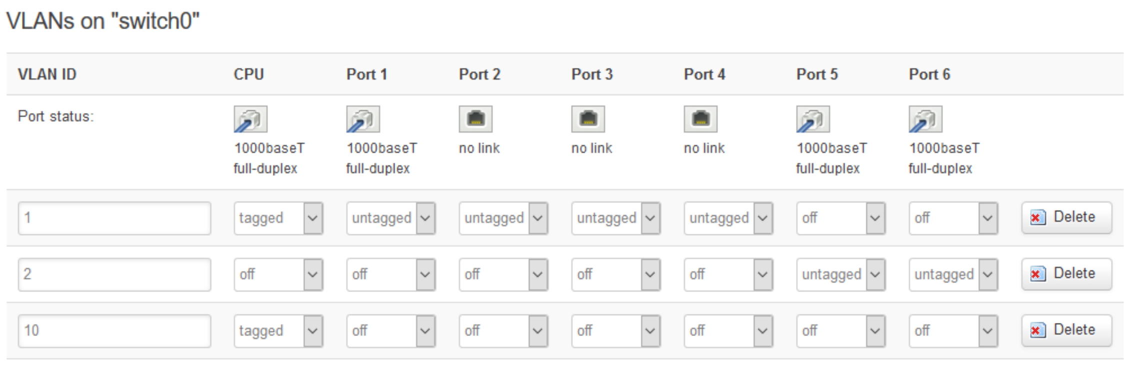
Now that the main interface and default VLAN are reconfigured, the guest VLAN and its corresponding virtual interface can be added. Therefore new switch_vlan and interface sections for VLAN ID 10 are added:

```
config switch_vlan
        option device 'switch0' #corresponding virtual switch
        option vlan '10'        #VLAN ID
        option vid '10'         #VLAN ID (overrides "option vlan")
        option ports '0t'       #ports to be (un)tagged, disabled
                                #(port 0 is tagged, rest disabled)

config interface 'vlan10'
        option ifname 'eth0.10'        #physical or virtual interface
        option type 'bridge'           #If set to "bridge", a bridge
                                       #containing the given ifnames
                                       #is created
        option proto 'static'         #Enables static adresses
        option ipaddr '192.168.10.1'  #IP address
        option netmask '255.255.255.0' #Netmask
```

Listing 5: Guest network configuration

For this guest network only the CPU port (Port 0) is tagged and all other ports are disabled (see Figure 27) to only grant access to the internet and no other portions of the network for this VLAN.


VLANs on "switch0"

VLAN ID	CPU	Port 1	Port 2	Port 3	Port 4	Port 5	Port 6	
Port status:	1000baseT full-duplex	1000baseT full-duplex	no link	no link	no link	1000baseT full-duplex	1000baseT full-duplex	
1	tagged	untagged	untagged	untagged	untagged	off	off	Delete
2	off	off	off	off	off	untagged	untagged	Delete
10	tagged	off	off	off	off	off	off	Delete

Figure 27: Default and Guest VLANs

NOTE: The number of the VLAN can be specified by two parameters, `option vlan` and `option vid` (see Listing 5). VID (`option vid`) is associated with a VLAN and is by default the same as the number of the VLAN (`option vlan`), but `vlan` can be overridden by `vid` so, for example VLAN 11 could be renamed to VLAN with ID 20 (see Listing 6) [38].

```
config switch_vlan
        option device 'switch0' #corresponding virtual switch
        option vlan '11'        #VLAN ID
        option vid '20'         #VLAN ID (overrides "option vlan")
```

Listing 6: Overridding VLAN ID

4.1.2. DHCP configuration

Now that the VLAN and the corresponding interface are configured, the guest network interface can be provided with options from the internal `dhcp` server. The configuration file responsible for this can be found at `/etc/config/dhcp`. To provide the before configured virtual interface with `dhcp` capabilities, a new `dhcp` section has to be added to the `dhcp` configuration file. Listing 7 shows an example for the previously configured guest network.

```
config dhcp 'vlan10'
        option interface 'vlan10'  #List of interfaces to listen on
        option start '50'          #Specifies the offset from the
                                   #network address
        option limit '100'         #Specifies the size of the
                                   #address pool
        option leasetime '12h'     #Specifies the lease time of
                                   #addresses handed out to clients
```

Listing 7: DHCP configuration

In this example the value of `option interface` refers to the alias of the virtual interface created in Listing 5 (`config interface` **`’vlan10’`**), `option start` indicates the starting IP address within the IP range provided by the virtual interface (in this case the starting IP would be `192.168.10.50`) and `option limit ’100’` restricts the maximum number of clients being permitted to connect to this particular interface or VLAN.

4.1.3. Firewall configuration

To ensure that the now created guest network is routed properly, a so called firewall zone and the appropriate forwarding rule for this zone need to be added to the firewall configuration (`/etc/config/firewall`). Listing 8 shows the configuration for our example VLAN with ID 10.

```
config zone
        option name 'vlan10'    #Unique zone name
        option input 'ACCEPT'   #policy for incoming zone traffic
        option output 'ACCEPT'  #policy for outgoing zone traffic
        option forward 'REJECT' #policy for forwarded zone traffic
        option network 'vlan10' #List of interfaces attached to zone
```

```

config forwarding
        option src 'vlan10'     #Specifies the traffic source zone
        option dest 'wan'       #Specifies the destination zone
```

Listing 8: Firewall guest network configuration

This example creates a firewall zone with name `vlan10` and connects it to the interface with the same name which was created before during the network configuration (see Section 4.1.1.). Furthermore all traffic from the source network `vlan10` is forwarded to the `wan` firewall zone and therefore to the router's WAN interface.

4.1.4. Wireless configuration

Finally to provide users with the ability to actually access the guest network, new SSIDs have to be created on the according wireless interfaces in OpenWrt's wireless configuration (`/etc/config/wireless`). Listing 9 provides an example wireless interface configuration with dynamic VLAN support and WPA2-Enterprise (WPA2-EAP) enabled.

```
config wifi-iface
        option device 'radio1'
        option mode 'ap'
        option ssid 'SomeSSID'
        option encryption 'wpa2'
        option auth_server '127.0.0.1'
        option auth_port '1812'
        option auth_secret 'testing123'
        option dynamic_vlan '2'
        option vlan_tagged_interface 'eth0'
        option vlan_bridge 'br-vlan'
        option vlan_naming '0'
```

Listing 9: Wireless configuration

To enable these features a few specific options have to be set in the according wireless interface section. WPA2-Enterprise is configured by setting `encryption 'wpa2'`, `auth_server '127.0.0.1'` (authentication server will be running on the device itself), `auth_port '1812'` (RADIUS default port) and `auth_secret 'testing123'` (default RADIUS shared secret, this should be changed accordingly). The configuration regarding WPA2-EAP can also be done via OpenWrt's own webinterface LuCI. Lastly the actual dynamic VLAN capability needs to be enabled by setting `dynamic_vlan '2'` (value '2' indicates that a VLAN ID is required and authentication is rejected by the RADIUS server if it is not supplied through the according RADIUS attributes, see Sections 3.6. and 4.2.2.), `vlan_tagged_interface 'eth0'` (Interface on which the CPU port was tagged, see Section 4.1.1.), `vlan_bridge 'br-vlan'` (defines the VLAN Bridge naming scheme) and `vlan_naming '0'` (defines how a vlan id is checked, in this case **vlan**<**ID**>).

This concludes the basic network and guest network configuration of the router, which now allows the authentication server to be configured and enabled for the corresponding wireless radios and SSIDs.

4.2. Authentication Server configuration using FreeRADIUS

This section of the thesis will cover the basic configuration of the authentication server, in this case FreeRADIUS, as well as the necessary steps to enable features needed to realise the requirements to the final system discussed in Chapter 1.1. Furthermore the configuration of the SQL portion will be discussed as well as the initial population of the FreeRADIUS specific MySQL database. Lastly this section will cover issues encountered with FreeRADIUS specific to OpenWrt.

4.2.1. Basic configuration

The configuration of the FreeRADIUS authentication server is split into five main files:

radiusd.conf	handles the configuration of the RADIUS daemon, i.e. the connection to the authentication server itself (listening ports, trusted foreign IP addresses or Ethernet ports), logging and including additional configuration files if needed.
eap.conf	provides the ability to configure several EAP-Method types (see Section 3.4.3.) for the authentication process as well as the required certificates.
clients.conf	governs a list of authenticators which are eligible to communicate with the authentication server.
sites/default	contains the default policies for the authentication process. It also provides the ability to enable modules for additional features.

sql.conf realises the configuration of the SQL module and holds the SQL server's connection data.

The subsequent sections will illustrate the necessary changes to these files to accomplish a working configuration with MySQL support and the features asked for in Chapter 1.1.

4.2.1.1. radiusd.conf

This represents the server's main configuration file. Here three sections need to be changed. First the "`listen`" section needs to be changed to only listen to the localhost for authentication packets. Listing 10 shows the parameters needed to accomplish this.

```
listen {
        #  Type of packets to listen for.
        #  Allowed values are:
        #       auth    listen for authentication packets
        #       acct    listen for accounting packets
        #       proxy   IP to use for sending proxied packets
        #       detail  Read from the detail file.  For examples, see
        #               raddb/sites-available/copy-acct-to-home-server
        #       status  listen for Status-Server packets.
        #               For examples,
        #               see raddb/sites-available/status
        #       coa     listen for CoA-Request and Disconnect-Request
        #               packets.  For examples, see the file
        #               raddb/sites-available/coa
        #
        type = auth
        ipaddr = 127.0.0.1
        port = 0
        #       interface = br-vlan1
        interface = lo  #optional
        ...
        }
```

Listing 10: `radiusd.conf` listen section

Setting "`port = 0`" causes the server to listen to it's default port which is provided by "`/etc/services`". Also the parameter "`interface = br-vlan1` needs to be made inactive, else the

RADIUS daemon won't respond to requests. Further due the fact that the RADIUS server is only needed locally it doesn't need to be bound to any interface at all, but can optionally be bound to the loopback interface (`interface = lo`).

Secondly the "`log`" section will be modified to provide more information about the authentication process. This is mainly done to provide this information to the webinterface later on and for debugging purposes (see Listing 11).

```
log {
	...
	auth = yes
	# rest remains unchganged
	...
	}
```

Listing 11: `radiusd.conf` log section

Lastly the "`modules`" section has to be changed so that the eap, `sql` and any additional modules are loaded (see Listing 12).

```
modules {
		#  Modules are initialized ONLY if they are
		#  referenced in a processing section, such as authorize,
		#  authenticate, accounting, pre/post-proxy, etc.
		$INCLUDE ${confdir}/modules/

		#  Extensible Authentication Protocol
		#  For all EAP related authentications.
		$INCLUDE eap.conf

		#  Include another file that has the SQL-related configuration
			.
		$INCLUDE sql.conf
		...
		}
```

Listing 12: `radiusd.conf` modules section

4.2.1.2. eap.conf

As already mentioned this file handles the configuration of the preferred EAP-Method which in this case will be PEAP (see Section 3.4.3.3.). To enable this authentication method a view changes have to be made to the "tls" and "peap" subsections of the eap section. Listing 13 shows the changes for providing the server with the necessary certificate files, which were created beforehand.

```
eap {
     ...

   tls {
         #       Configuration Directory
         confdir = /etc/<some directory>/CA
         #       Certificates
         certdir = ${confdir}
         #       Certificate Authority
         cadir = ${confdir}
         #       Private Key data
         private_key_password = <password>
         private_key_file = ${certdir}/server.pem
         #       Certificate file
         certificate_file = ${certdir}/server.pem
         #       Certificate Authority file
         CA_file = ${cadir}/ca.pem
         #       DiffieŰHellman cipher file
         dh_file = ${certdir}/dh
         random_file = ${certdir}/random
         #       Certificate Authority path
         CA_path = ${cadir}

         ...
        }

      peap {
         #  The tunneled EAP session needs a default
         #  EAP type which is separate from the one for
         #  the non-tunneled EAP module.
         #  MS-CHAPv2 is recommended as this is the
         #  default type supported by Windows clients.
         default_eap_type = mschapv2
         ...
         }
     ...
     }
```

Listing 13: `eap.conf` eap section

4.2.1.3. clients.conf

To provide the authenticator with the ability to actually communicate with the authentication server, it needs to exist within `clients.conf`. Since in this case the authenticator will be the device itself it is only necessary to configure the client/authenticator labelled "`localhost`". Listing 14 shows the required parameters to be set.

```
client localhost {
        ipaddr = 127.0.0.1
        secret = <shared secret>

        # Older Clients my not include a Message-Authenticator in an
        # Access-Request packet. Therefore to also support older
        # clients this should be set to "no"
        require_message_authenticator = no

        nastype = other
        ...
    }
```

Listing 14: `client.conf` localhost section

After setting these parameters the hostapd process and the wireless interface configured in Section 4.1.4. can finally communicate with the authentication server.

4.2.1.4. Enabling additional modules

Additional FreeRADIUS modules can be enabled in `/etc/freeradius2/sites/default` by uncommenting the appropriate lines within the `authorize` section. To facilitate the features described in Section 1.1., namely "Account Expiration" and a "Login Time Restriction", the modules `expiration` and `logintime` have to be installed and activated (see Listing 15).

```
authorize {
        ...
        expiration
        logintime
        ...
    }
```

Listing 15: `sites/default` authorize section

4.2.1.5. SQL configuration

The MySQL configuration for FreeRADIUS is spread across two files, `radiusd.conf` (already handled in Section 4.2.1.1.) and `/etc/freeradius2/sites/default`. The second part in "`sites/default`" concerns the sections `authorize`, `accounting` and `session` (see Listing 16).

```
authorize {
        ...
        #  Look in an SQL database.  The schema of the database
        #  is meant to mirror the "users" file.
        sql
        ...
    }

#  Accounting.  Log the accounting data.
accounting {
        #  Log traffic to an SQL database.
        sql
    }



```

```
#  Session database, used for checking Simultaneous-Use.
session {
        radutmp
        sql
    }
```

Listing 16: `sites/default` sql configuration

Enabling the `sql` parameter in the `accounting` section isn't necessarily required since Accounting won't be used but should be enabled nevertheless to avoid possible errors should Accounting be required in the future.

4.2.2. Populating the User database

Now that the authentication server is completely configured, the MySQL database holding the users information can be populated with the initial users, groups and other relevant data. The FreeRADIUS MySQL database consists of the following tables to represent the data which per default would be stored within the "`users`" file:

- radcheck
- radreply
- radgroupreply
- radusergroup
- radgroupcheck
- radacct
- radpostauth

Out of those seven tables only the first five are used for the purpose of this work. The tables `radacct` and `radpostauth` are handling, as the names suggest, Accounting and Post-Authentication RADIUS attributes which aren't used in this case.

To illustrate how these tables are used and what data they contain, a few groups and users are created. As a first step three different groups with their corresponding RADIUS attributes are inserted into the table radgroupreply, see Listing 17 below.

```
mysql> select * from radgroupreply;
+----+--------------------+------------------------+----+----------+
| id | groupname          | attribute              | op | value    |
+----+--------------------+------------------------+----+----------+
|  1 | guest              | Tunnel-Type            | =  | VLAN     |
|  2 | guest              | Tunnel-Medium-Type     | =  | IEEE-802 |
|  3 | guest              | Tunnel-Private-Group-ID | =  | 10       |
|  4 | FullAccess         | Tunnel-Type            | =  | VLAN     |
|  5 | FullAccess         | Tunnel-Medium-Type     | =  | IEEE-802 |
|  6 | FullAccess         | Tunnel-Private-Group-ID | =  | 1        |
|  7 | ParentalControl_1  | Tunnel-Medium-Type     | := | IEEE-802 |
|  8 | ParentalControl_1  | Tunnel-Type            | := | VLAN     |
|  9 | ParentalControl_1  | Tunnel-Private-Group-ID | := | 10       |
+----+--------------------+------------------------+----+----------+
```

Listing 17: Group definitions

Two of these groups, “FullAccess” and “guest”, will be part of the default installation of this project, whereas “ParentalControl_1” only exists for testing and illustration purposes later on. Each group is defined by three RADIUS attributes, Tunnel-Type, Tunnel-Medium-Type and Tunnel-Private-Group-ID which allow for a dynamic vlan assignment (for further information about these attributes please refer to Section 3.6. and RFC 2868 [36]). These groups are responsible for assigning their respective users the according vlan, i.e. the group “guest” will assign vlan 10 (which was configured in Section 4.1.) to all guest users and all “FullAccess” users will be granted access to the default vlan.
Next users are created within the table radcheck, which basically holds the username and all attributes that need to be checked before granting a user access to a network. These attributes include the password, expiration date as well as the login time (see Listing 18).

```
mysql> select * from radcheck;
+----+-----------+--------------------+----+------------------+
| id | username  | attribute          | op | value            |
+----+-----------+--------------------+----+------------------+
|  1 | admin     | Cleartext-Password | := | password         |
| 44 | g-82842   | Cleartext-Password | := | h7352            |
| 45 | g-82842   | Expiration         | := | 6 May 2016 12:28 |
+----+-----------+--------------------+----+------------------+
```

Listing 18: User definitions

NOTE: The passwords can be stored as "`Cleartext`" due to the reasons described in Section 3.5. Further the usage of the "MD5-Password" attribute would prohibit FreeRADIUS from using `MsChapv2` or any `CHAP` authentication for that matter [39].

Finally each of the users created above is assigned to a group within table `radusergroup`.

```
mysql> select * from radusergroup;
+-----------+--------------------+----------+
| username  | groupname          | priority |
+-----------+--------------------+----------+
| admin     | FullAccess         |        1 |
| g-82842   | guest              |        1 |
+-----------+--------------------+----------+
```

Listing 19: User to Group assignment

In some cases like the in Listing 17 defined group "ParentalControl_1" an additional table is needed, since this group requires a group wide check attribute to realise the login time restriction for it's users (see Listing 20).

```
mysql> select * from radgroupcheck;
+----+-------------------+------------+----+-------------+
| id | groupname         | attribute  | op | value       |
+----+-------------------+------------+----+-------------+
|  1 | ParentalControl_1 | Login-Time | := | Al0800-2000 |
+----+-------------------+------------+----+-------------+
```

Listing 20: Group wide attributes

In this example the value “Al0800-2000” indicates a daily access to the groups corresponding network between 08:00 and 20:00. For further information about the Login-Time attribute and it's values please refer to the FreeRADIUS documentation[28].
Additionally, if there should be user without any group affiliations, all data that is usually held in `radgroupreply` needs to be stored on a user instead of group basis, which is done by the table `radreply` (see Listing 21).

```
mysql> select * from radreply;
+----+----------+-------------------------+----+----------+
| id | username | attribute               | op | value    |
+----+----------+-------------------------+----+----------+
|  1 | test     | Tunnel-Type             | =  | VLAN     |
|  2 | test     | Tunnel-Medium-Type      | =  | IEEE-802 |
|  3 | test     | Tunnel-Private-Group-ID | =  | 10       |
+----+----------+-------------------------+----+----------+
```

Listing 21: Groupless user's reply section

NOTE: Users without a group are automatically assigned to the guest network (see Section 4.3.1.).

4.2.3. Issues with FreeRADIUS on OpenWrt

During the course of the installation and configuration of FreeRADIUS a few issues with the support of SQL came to light. First off FreeRADIUS or rather the sql package needed (`freeradius2 -mod-sql`) to enable SQL support from OpenWrt's repository is missing all files except the main SQL configuration file mentioned earlier. These missing files have been taken from an installation on a Debian system and will be included within the final OpenWrt package. Further FreeRADIUS2 is missing the support for SQLite, which would have saved resources on the device compared to using a MySQL database (also see section 4.5. for Hard- and Software requirements). This is not an issues specific to OpenWrt but in general. The support of SQLite is now part of FreeRADIUS3 which isn't part of any official repository yet and therefore not a viable solution for this project, since it would have to be compiled for each platform individually.

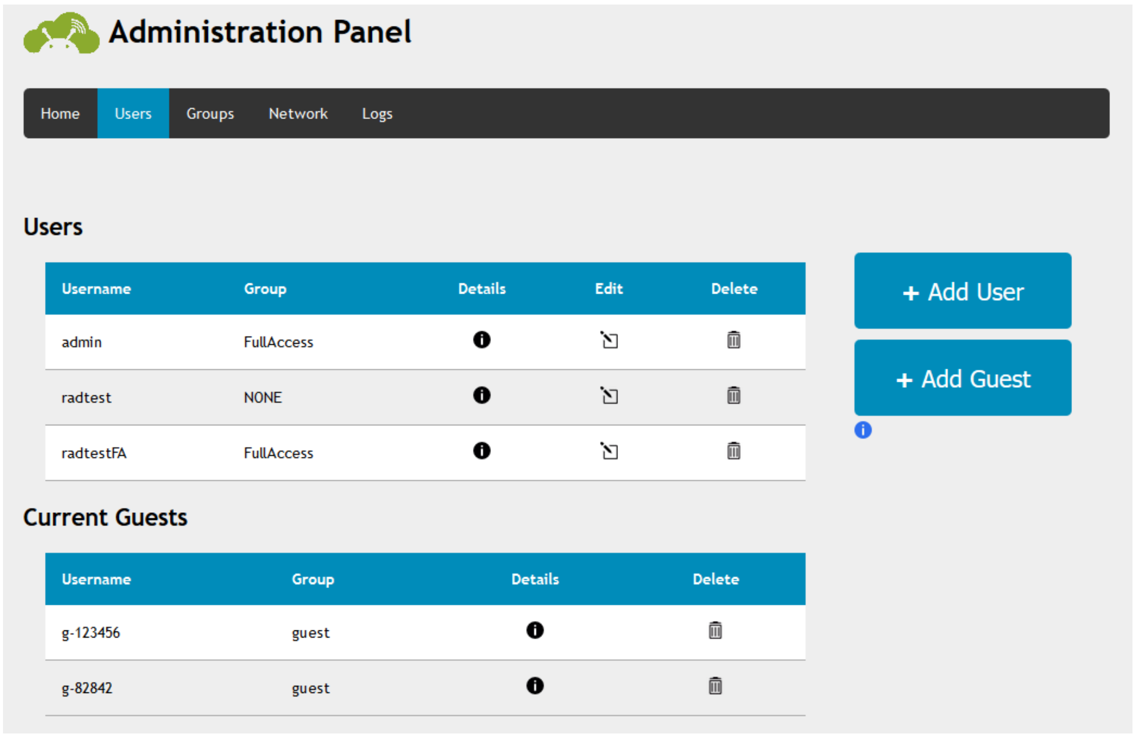
[28] FreeRADIUS Special attributes used in the users file: `http://wiki.freeradius.org/config/Users#special-attributes-used-in-the-users-file`

4.3. Web interface

To finally provide the administrator/owner of a wireless home network with the ability to easily manage the before configured system, a rather simple web based administration interface was implemented. Which allows for creating and managing users, guests and groups, provides an overview of all existing networks/vlans and an authentication and error log. The subsequent sections will elaborate on how the former mentioned functionality is implemented and how to actually operate this web interface.

4.3.1. User management

The user management page provides the administrator with an overview of all existing users and currently active guests as well as their current group affiliation. It also allows for the creation of additional users and guests, as well as the editing, deletion and displaying of detailed information of existing ones (see Figure 28).



Users

Username	Group	Details	Edit	Delete
admin	FullAccess			
radtest	NONE			
radtestFA	FullAccess			

Current Guests

Username	Group	Details	Delete
g-123456	guest		
g-82842	guest		

Figure 28: User management

Adding a new guest user will result in the creation of a new user affiliated with the group "guest" and an expiration time of four hours from the time of creation. Additionally a QRCode based on this users credentials will be generated to potentially allow an easier distribution of user data to the guest (for additional information about credential distribution please refer to Section 4.4.). The generation of guest credentials is handled by the the class "CGenerator" (see Listing 22), which creates a new username starting with "g-" and a suffix based on the current time (e.g. "g-103456", possible collisions are avoided when generating the user) as well as a five character long password starting with a random letter followed by four random digits. These username and password are chosen to be rather short and simple in order to minimise the guests effort when entering his credentials manually (which will be the default case). Additionally since the guests are only valid for four hours and are regularly deleted this shouldn't impose any security risks.

```
class CGenerator{

    private $username;
    private $password;

    function createGuest() {
        $this->username = "g-" . date("Gis");
        return $this->username;
    }

    function generatePasswd(){
        $letters = array_merge(range('a', 'z'), range('A', 'Z'));
        $this->password = $letters[$this->crand(0, 51)] . $this->crand
            (1000,9999);
        return $this->password;
    }

    function createQRCodeFrom($ssid, $uname, $passwd){
        $qrstr = 'TXT:WIFI:S:' . $ssid . ';T:802.1x EAP;U:'. $uname .'
            ;P:'. $passwd . ';;';
        exec("qrencode -s 8 -l H -c -m 0 -t SVG -o ./cache/qrcode.svg
            '" . $qrstr . "'");
    }

    function crand($min, $max){
        $range = $max - $min;
        if ($range < 1) return $min;
        $clog = ceil(log($range, 2));
        //number of bytes
        $byte_len = (int)($clog / 8) + 1;
        //length in bits
        $bit_len = (int)$clog + 1;
        //set all relevant bits to 1
        $cutoff = (int)(1 << $bit_len) - 1;
        do{
            $random = hexdec(bin2hex(openssl_random_pseudo_bytes(
               $byte_len)));
            //filter irrelevant bits
            $random = $random & $cutoff;
        } while ($random > $range);
        return $min + $random;
    }
}
```

Listing 22: Guest user credential generation

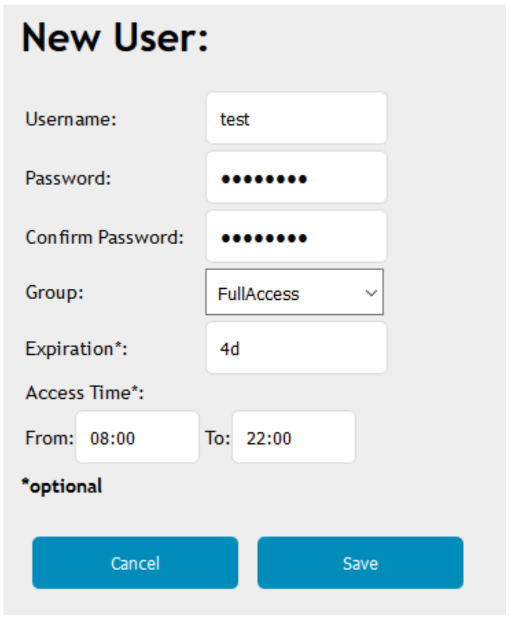
Further this class offers the generation of a QRCode appropriate to a given SSID (SSID of the 802.1x network) and credentials generated beforehand. This is accomplished by creating a string using the following scheme:

```
WIFI:S:<SSID>;T:802.1x EAP;U:<USERNAME>;P:<PASSWORD>;;
```

The substring `WIFI` informs the barcode scanner that this string contains network connection data and therefore should communicate with the devices network manager to add a new wireless connection with the security setting set to the value described by `T` (WEP, WPA, 802.1x EAP) within the QRCode string. Based on this string a QRCode is created using the tool "`qrencode`", which is provided by OpenWrt's repository and based on "`libqrencode`".

Figure 29 shows the creation of a new user, which allows creating users with a specific group affiliation as well as the two optional parameters "Expiration" and "Access Time". The expiration can be provided in either hours or days (e.g. 2h or 4d), if left blank the account won't expire until deleted or revoked by the administrator. Access Time has to be provided in a 24h format and will result in a daily login time restriction within the given timeframe.

NOTE: Users with no group affiliation (Group = `NONE`) are automatically assigned to the guest network by the Web Interface to avoid unwanted access to the main network.



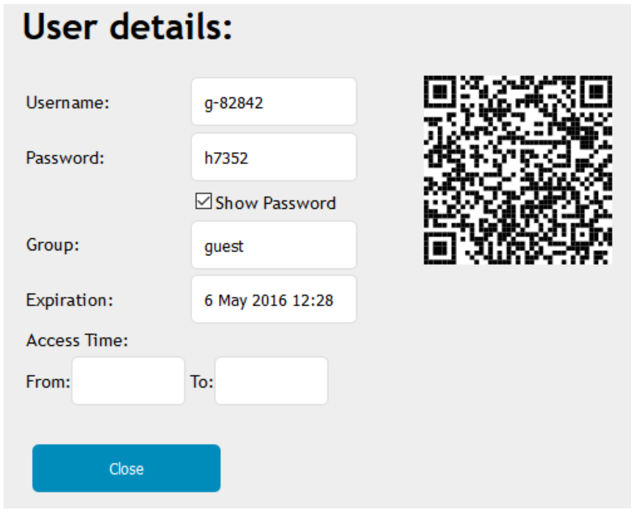
New User:

Username:	test
Password:	••••••••
Confirm Password:	••••••••
Group:	FullAccess
Expiration*:	4d
Access Time*:	
From: 08:00	To: 22:00

*optional

Figure 29: Create user

Additionally this page allows to show a users details, including the actual expiration date and time as well as the QRCode generated for this specific user (see Figure 30).



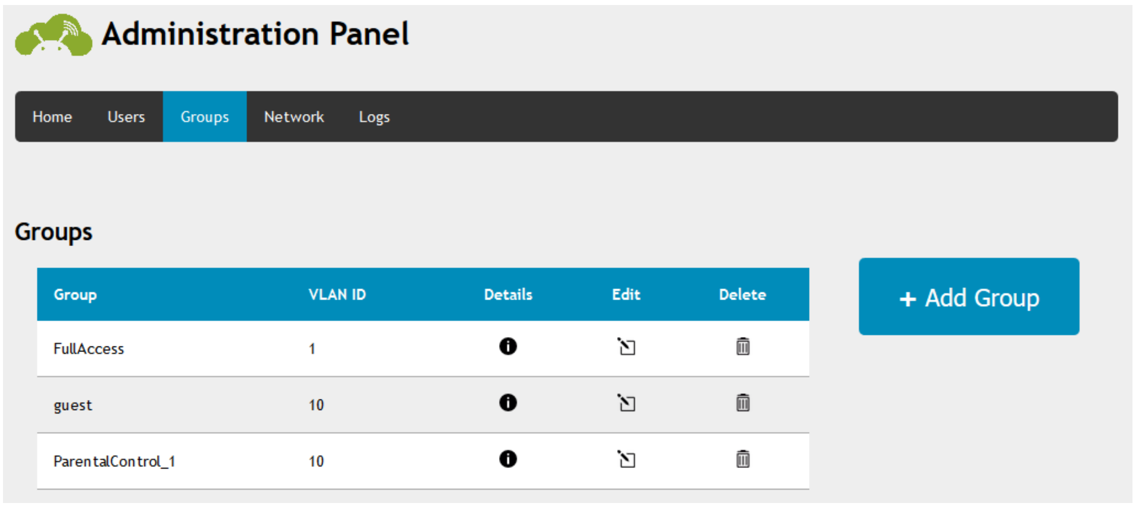
User details:

Username:	g-82842
Password:	h7352
	☑ Show Password
Group:	guest
Expiration:	6 May 2016 12:28
Access Time:	
From:	To:

Figure 30: User details

4.3.2. Group management

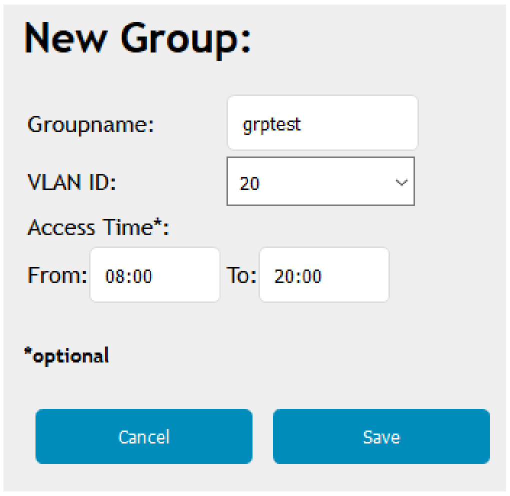
Similar to the user management page, the group management provides an overview of all existing groups and their currently configured VLAN, as well as the ability to add further groups and edit and delete existing ones (see Figure 31).



Group	VLAN ID	Details	Edit	Delete
FullAccess	1	●	✎	🗑
guest	10	●	✎	🗑
ParentalControl_1	10	●	✎	🗑

Figure 31: Group management

Figure 32 shows the group creation interface which allows the adding of additional groups with a specific VLAN configured on the device. Optionally groups, just like users, can be created with an "Access Time" parameter to realise a daily login time restriction on a group basis (e.g. for enforcing a sort of parental control).



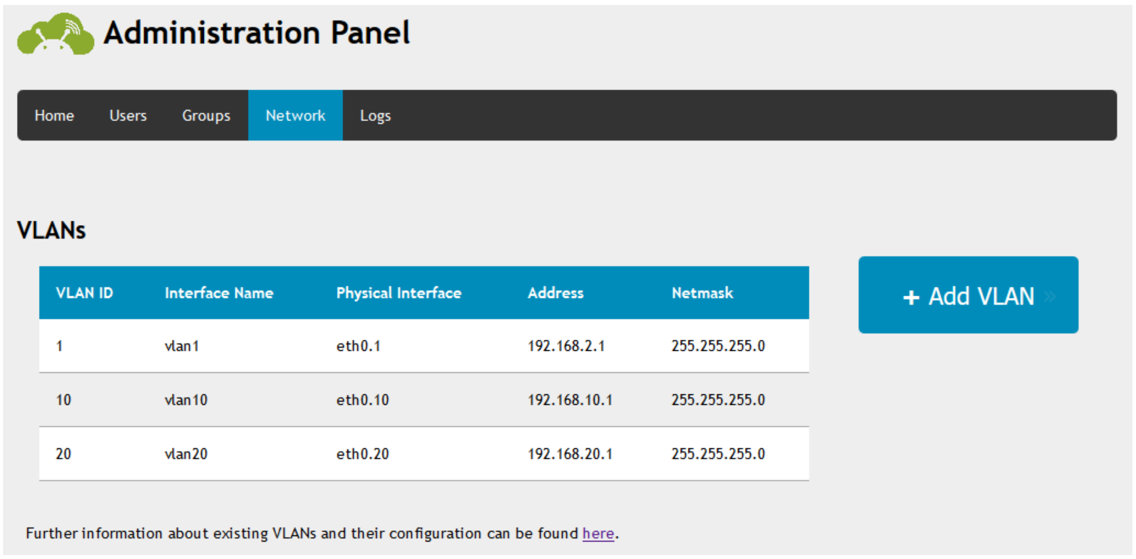
Groupname:	grptest
VLAN ID:	20
Access Time*:	
From: 08:00	To: 20:00

*optional

Figure 32: Create group

4.3.3. Network/VLAN overview

The Network/VLAN page simply provides an overview of all existing VLANs, as well as details about physical/virtual interface, IP address and netmask of each VLAN (see Figure 33). Additionally it provides a tutorial for adding further VLANs using OpenWrt's own web interface LuCI with links to the corresponding locations.



VLANs

VLAN ID	Interface Name	Physical Interface	Address	Netmask
1	vlan1	eth0.1	192.168.2.1	255.255.255.0
10	vlan10	eth0.10	192.168.10.1	255.255.255.0
20	vlan20	eth0.20	192.168.20.1	255.255.255.0

Further information about existing VLANs and their configuration can be found here.

Figure 33: Network/VLAN overview

The ability to manage VLANs wasn't implemented here to avoid possible inconsistencies with OpenWrt's own network configuration. Also the default installation will offer VLANs for most cases. Further a more granular VLAN configuration will require detailed knowledge about the network's configuration and infrastructure. Therefore implementing this functionality would unnecessarily complicate the interface for the average user.

Additionally this interface includes an error and authentication log to provide an overview of the most recent activity. This is done by simply parsing and filtering FreeRADIUS's log file (`radius.log`).

4.3.4. Portability and Modifiability

To guarantee a certain degree of portability and extensibility the entire web interface was written in PHP which should allow for the widest range of compatibility regarding host devices and operating systems and therefore allows for a rather uncomplicated port to other WRT derivatives such as dd-wrt[29]. The PHP interpreter also provides the least overhead in terms of flash memory, when considering alternatives such as `python` which provide a similar feature set (also see Section 4.5. for Hard- and Software requirements). To realise a port to dd-wrt the current implementation of the class `uci`, which handles the communication with OpenWrt's configuration interface (see Section 3.1.4.), would have to be replaced with an implementation for dd-wrt's equivalent NVRAM. Due to the fact that NVRAM isn't as uniform as UCI across different devices and dd-wrt at it's current state lacks the support for dynamic VLAN assignments, the implementation of this port was defined as out of scope of this thesis. In it's current state the implementation of the above mentioned `uci` class (Listing 23) only provides capabilities for retrieving network information using OpenWrt's command line configuration interface and could easily be extended to accommodate more functionality should this be required in the future.

```
class uci{

    ...
    /*
     * Return array of existing VLAN IDs
     */
    function getVlanIDs(){
        $str = shell_exec('uci show network');
        $pattern = "/.vid='([0-9]+)'/";
        if (preg_match_all($pattern,$str,$out)){
            $vlans = array_unique($out[1]);
            natsort($vlans);
            return array_values($vlans);
        }
        else return null;
    }

    /*
     * Returns the virtual interface name
```

[29]`http://www.dd-wrt.com/`

```
         */
        function getVlanIfaceName($vlanID){
            $str = shell_exec('uci show network');
            $pattern = "/\.(\w+)\.ifname='eth[0-9]+\." . $vlanID . "'/";
            if (preg_match_all($pattern,$str,$out)){
                return $out[1][0];
            }
            else return null;
        }

        ...
}
```

Listing 23: Class `uci`

Also the connection to the database is handled by a separate class which should make it easy to change to a different database backend, such as SQLite, in the future.

4.4. Credential distribution

The implemented solution is not only intended to make it as easy as possible for the administrator of a home network to generate and handle credentials for guest users but also to provide fast and easy ways to distribute them to the guest user. Further it has to be as easy as possible for the guest user to connect to the network to accept a different and maybe more complicated authentication method. Due to the fact that most wide spread authentication distribution methods, such as WPS, are incompatible with 802.1x authentication methods and the intention of not having the guest install a third party application (which is the case with some proprietary solutions) to communicate with the system it was decided to have the user enter his credentials manually or provide a QRCode containing said credentials and network information to deliver them to the target device. Since in many cases users will have to setup the connection and enter their credentials manually (e.g. on none mobile devices or depending on the operating system), the generated credentials have been chosen in a rather short manner (see Section 4.3.1.), which shouldn't impose any security risks since guest users are in their separate VLAN and are only temporary users with a rather short expiration time of four hours.

4.4.1. Distribution via QRCodes

The distribution of user credentials and network information via QRCodes is meant to further alleviate the process of connecting new users to the network. During the course of the implementation of this feature a few issues arose. First and foremost creating network connections through QRCodes is only possible on Android devices, since other mobile operating systems like iOS or Windows Phone don't allow third party applications access to the network manager, therefore Bar-/QRCode scanners on these systems will interpret a wifi QRCode pattern (see below) as plain text and display it accordingly.

```
WIFI:S:<SSID>;T:<Network type>;P:<PASSWORD;;
```

On Android devices the need for "`802.1x EAP`" as the network type and an additional field for the username (see proposed string below) this led to another set of problems on the client side.

```
WIFI:S:<SSID>;T:802.1x EAP;U:<USERNAME>;P:<PASSWORD>;;
```

Since most QRCode scanners for Android are based on ZXing's open source barcode scanner project[30] and ZXing's wifi pattern parser doesn't support "`802.1x EAP`" networks the proposed pattern will not work as intended. This is rooted in the fact that the 802.1x support within the Android's network manager changed between versions 4.0 and 4.3, therefore ZXing won't implement support for 802.1x networks in favour of supporting a wider range of devices and Android versions.
Considering these issues, it was decided to keep the proposed string in it's current state but to add a preamble (see Listing 24) so it would be interpreted as text across all platforms until barcode scanners would support the required network type.

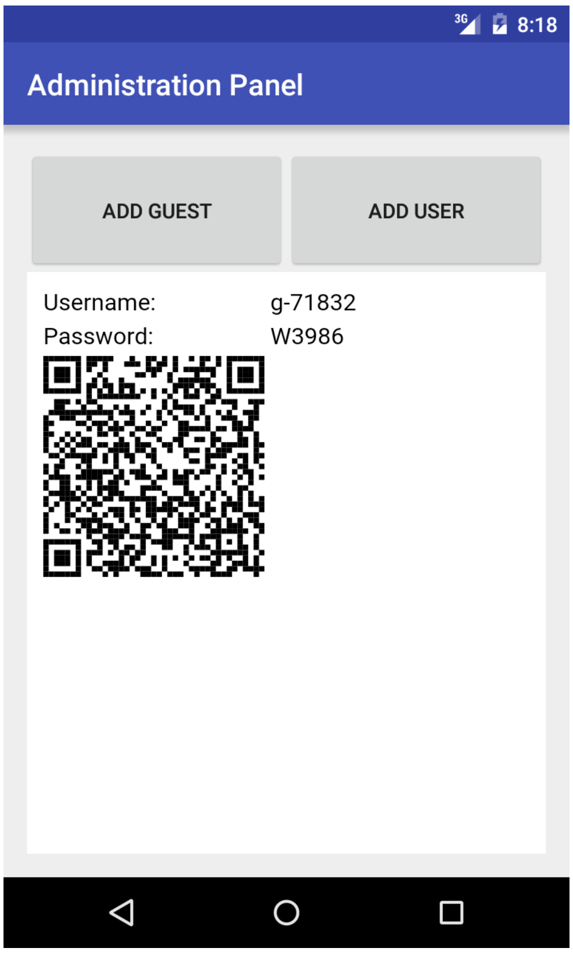
```
TXT:WIFI:S:<SSID>;T:802.1x EAP;U:<USERNAME>;P:<PASSWORD>;;
```

Listing 24: Final QRCode string

[30]Official ZXing project home: `https://github.com/zxing/zxing`

4.4.2. Mobile application for administrators

To provide an alternative to the web interface and therefore further ease the process of creating and distributing guest/user credentials this thesis proposes a mobile application for administrators using the implemented solution. Although a mobile application is out of scope of this thesis, a basic version was implemented as a prove of concept and to propose a possible form of communication between the application and the web interface.
Figure 34 shows a proposal of a minimalistic interface allowing the user to generate new guests and add further users to the system, as well as display the created users credentials and corresponding QRCode.



Username: g-71832
Password: W3986

Figure 34: Minimalistic application interface

To accomplish a communication with the web interface it was decided to encapsulate the user data into JSON objects and deliver them to the PHP implementation of the web interface via HTTP POST. Listing 25 shows the format of the JSON object as well as the necessary steps to create a readable string for the PHP implementation on the server side. For reasons of simplification at this point the application only generates HTML output and displays it in a simple WebView (see Figure 34).

```java
private String postRequest(String httpUrl) {

        ...

        /* Create and POST JSON object */
        URL url = new URL(httpUrl);
        JSONObject jo = new JSONObject();
        jo.put("uname", "testusr");
        jo.put("passwd", "atest123");
        jo.put("passwd_conf", "atest123");
        jo.put("group", "FullAccess");
        jo.put("expiration", "10h");
        jo.put("AT_from", "08:00");
        jo.put("AT_to", "16:00");
        String json = jo.toString();

        connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("POST");
        connection.setFixedLengthStreamingMode(json.getBytes().length);
        connection.setRequestProperty("Content-Type", "application/json;
            charset=utf-8");
        connection.setRequestProperty("X-Requested-With", "
            XMLHttpRequest");
        connection.connect();

        ...
        /* retrieve HTML response and output to WebView */
        ...
}
```

Listing 25: Client side

On the server side (Listing 26) the JSON string generated by the client is being read and decoded (`json_decode`) which in turn results in an array of key, value pairs. After decoding, the necessary information is extracted and the usual database operations are performed to create the user.

Subsequently an array (of key, value pairs) with the data relevant to the user is created and again encoded into a JSON string (json_encode) and delivered back to the client.

```
    /* create user data from JSON object */

    if ($_SERVER["REQUEST_METHOD"] == "POST"){
        $jsonIn = file_get_contents('php://input');
        $js = json_decode($jsonIn);

        if (check if decoded array is valid){
            $username = $js->{'uname'};
            $password = $js->{'passwd'};
            $password_conf = $js->{'passwd_conf'};
            $group = $js->{'group'};
            $expiration = $js->{'expiration'};
            $time_from = $js->{'AT_from'};
            $time_to = $js->{'AT_to'};


        /* database operations */

        ...

        /*
         * Responding JSON Object fields:
         * uname          Username
         * passwd         Password
         * group          Groupname
         * expiration     Expiration Time
         * AT_from        starting Access Time
         * AT_to          ending Access Time
         * error          Error message
         */

        /* create and encode responding JSON string */

            $response['uname']  = $username;
            $response['passwd'] = $password;
            $response['group']  = $group;
            $response['expiration']  = $date;
            $response['AT_from'] = $time_from;
            $response['AT_to'] = $time_to;
            $response['error'] = $error_msg;

            echo json_encode($response);
```

Listing 26: Server side

NOTE: This basic implementation is just to be viewed as a proposal for possible future work.

4.5. Hard- and Software requirements

The implemented solution has a few demands regarding software to be installed from OpenWrt's repository as well as specific features provided by the operating system itself. The following list provides an overview of the required software to run this implementation:

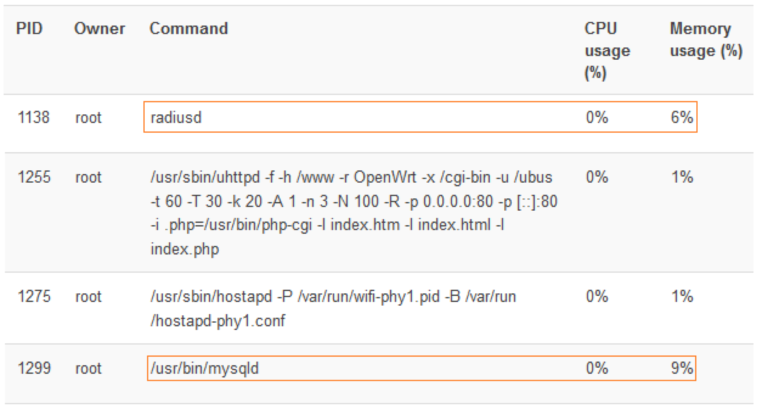
- OpenWrt 15.05 or higher
- FreeRADIUS2
- MySQL server
- PHP5
- Webserver (`uhttpd` preinstalled by OpenWrt)
- OpenSSL utilities
- A full featured 802.1x / WPA / EAP / RADIUS authenticator and Supplicant (`wpad`)
- QR-Encoder (`qrencode`)

A minimum of OpenWrt 15.05 is required due to need of dynamic VLAN assignment support (see Section 4.1.). Having the newest version of the operating system installed as well as some additional software and services, also resulted in a higher demand on the hardware side:

- OpenWrt 15.05 compatibility
- 16MB or more flash memory
- 64MB RAM (current implementation uses about 34MB)

- CPU depending on architecture 300 - 400MHz (OpenWrt's current minimum requirement)

The high demand on RAM results from the system having to run FreeRADIUS and MySQL on the device. Whereas FreeRADIUS requires 6% and MySQL about 9 to 10% of the above mentioned 34MB of RAM (see Figure 35).



PID	Owner	Command	CPU usage (%)	Memory usage (%)
1138	root	radiusd	0%	6%
1255	root	/usr/sbin/uhttpd -f -h /www -r OpenWrt -x /cgi-bin -u /ubus -t 60 -T 30 -k 20 -A 1 -n 3 -N 100 -R -p 0.0.0.0:80 -p [::]:80 -i .php=/usr/bin/php-cgi -I index.htm -I index.html -I index.php	0%	1%
1275	root	/usr/sbin/hostapd -P /var/run/wifi-phy1.pid -B /var/run /hostapd-phy1.conf	0%	1%
1299	root	/usr/bin/mysqld	0%	9%

Figure 35: Memory intensive processes

Therefore a switch to SQLite and FreeRADIUS3 might free up enough memory for the system to run on devices with only 32MB of RAM.

5. Future work

While this thesis already implements a rather lightweight and simple to use 802.1x authentication solution there is still room for improvement and additional features in the future. First and foremost an upgrade from FreeRADIUS2 to FreeRADIUS3, as it becomes available in OpenWrt's software repository, would lower the hardware requirements significantly due to the possibility of introducing SQLite as the database backend. Which in turn may result in the support of a wider range of devices.
Also the current string used to generate the QRCodes should be modified accordingly if future versions of barcode scanners should start to support 802.1x EAP network types.
By adding further FreeRADIUS modules and their attribute support it would also be easily possible to introduce further functionality such as bandwidth restrictions of groups or individual users[31]. Since especially bandwidth might be an issue in home environments with data rates often not exceeding 20 Mbps.

[31] FreeRadius Server How to: `http://itnetwork-infrastructure.blogspot.co.at/2012/01/freeradius-server-how-to-install-part-2.html?m=1`

Further porting the current implementation to other WRT derivatives would be an option as soon as the necessary features (e.g. dynamic VLAN assignment) become available on the target operating systems.
Lastly, since the implementation of the current mobile application is just to be seen as a proposal, implementing a version which covers the full set of features provided by the current web interface would be the biggest enrichment to the implemented solution.

6. Conclusion

This thesis designed and implemented an 802.1x authentication solution for home users which can be installed and run on a consumer router. The solution should provide a very user-friendly and easy to use interface for administrators of home networks as well as uncomplicated ways for guest users to join the implemented wireless guest network. The administration interface offers all necessary key features of a 802.1x authentication system such as user and group management as well as a network overview and some simple logging. Further an interface for a possible mobile application (for administration and distribution purposes) was proposed which could provide an even more comfortable way of interacting with the system in the future.
The higher complexity and requirements of WPA2-802.1x based authentication models compared to WPA2-PSK, which results in a tradeoff between security and usability, was minimised by a number of factors. Firstly the use of as many lightweight components as possible to ensure the support of a wide range of devices with limited (hardware) resources. A simple and user-friendly administration interface for quickly generating new temporary guest users and administrating permanent ones. To encourage user acceptance, guest user credentials were chosen to be rather short but still with the length of default passwords (ten to fifteen characters) used for WPA2-PSK secured networks. Provided that generated guest users are only valid for a short period of time this shouldn't impose a risk to security.
For administrators of wireless home networks who wish to have a more granular control over their network or simply want an increase in security, the provided solution will make it possible to do so with little effort and no additional hardware other than a consumer router.

7. Appendix

7.1. EAP-Method Types

EAP-Method	Value (Decimal)	RFC
Identify	1	RFC 3748
Notification	2	RFC 3748
NAK (Response Only)	3	RFC 3748
MD5-Challenge	4	RFC 3748
OTP, One Time Password	5	RFC 2289 RFC 3748
GTC, Generic Token Card	6	RFC 3748
Allocated	7	
Allocated	8	
RSA Public Key Authentication	9	
RSA Public Key Authentication	10	
KEA	11	
KEA-VALIDATE	12	
EAP-TLS Authentication Protocol	13	RFC 5216
Quest Defender Token	14	
RSA Security SecurID EAP	15	
Arcot System EAP	16	
Cisco-LEAP	17	
EAP-SIM, GSM Subscriber Identity Modules	18	RFC 4186
SRP-SHA-1 Part 1	19	
SRP-SHA-1 Part 2	20	
EAP-TTLS, EAP Tunneled TLS Authentication Protocol	21	RFC 5281
Remote Access Service	22	
EAP-AKA, EAP method for 3rd Generation Authentication and Key Agreement	23	RFC 4187
EAP-3Com Wireless	24	

EAP-Method	Value (Decimal)	RFC
PEAP, Protected EAP	25	
MS-EAP-Authentication (EAP/MS-CHAPv2)	26	
EAP-MAKE, Mutual Authentication w/Key Exchange	27	
CRYPTOCard	28	
PEAPv0/EAP-MSCHAPv2	29	
DynamID	30	
Rob EAP	31	
EAP-POTP, Protected One Time Password	32	RFC 4793
MS-Authentication-TLV	33	
SentriNET	34	
EAP-Actiontec Wireless	35	
Cogent Systems Biometrics Authentication EAP	36	
AirFortress EAP	37	
EAP-HTTP Digest	38	
SecureSuite EAP	39	
DeviceConnect EAP	40	
EAP-SPEKE	41	
EAP-MOBAC	42	
EAP-FAST, EAP Flexible Authentication via Secure Tunneling	43	RFC 4851 RFC 5421 RFC 5422
ZLXEAP, ZoneLabs EAP	44	
EAP-Link	45	
EAP-PAX, EAP Password Authentication eXchange	46	RFC 4746
EAP-PSK, EAP Pre-Shared Authentication and Key Establishment	47	RFC 4764
EAP-SAKE, EAP Shared-secret Authentication and Key Establishment	48	RFC 4763

EAP-Method	Value (Decimal)	RFC
EAP-IKEv2	49	RFC 5106
EAP-AKA, Improved EAP method for 3rd Generation Authentication and Key Agreement	50	
EAP-GPSK, EAP Generalized Pre-Shared Key	51	RFC 5433
Available via review by designated expert	52-191	RFC 3748
Reserved for allocation via standards action	192-253	RFC 3748
Expanded Type	254	RFC 3748
Experimental	255	RFC 3748

Table 7: EAP-Method types

7.2. RADIUS Attribute Types

RADIUS Attribute Name	Attribute Number
User-Name	1
User-Password	2
CHAP-Password	3
NAS-IP-Address	4
NAS-Port	5
Service-Type	6
Framed-Protocol	7
Framed-IP-Address	8
Framed-IP-Netmask	9
Framed-Routing	10
Filter-Id	11
Framed-MTU	12
Framed-Compression	13
Login-IP-Host	14
Login-Service	15
Login-TCP-Port	16

RADIUS Attribute Name	Attribute Number
(unassigned)	17
Reply-Message	18
Callback-Number	19
Callback-Id	20
(unassigned)	21
Framed-Route	22
Framed-IPX-Network	23
State	24
Class	25
Vendor-Specific	26
Session-Timeout	27
Idle-Timeout	28
Termination-Action	29
Called-Station-Id	30
Calling-Station-Id	31
NAS-Identifier	32
Proxy-State	33
Login-LAT-Service	34
Login-LAT-Node	35
Login-LAT-Group	36
Framed-AppleTalk-Link	37
Framed-AppleTalk-Network	38
Framed-AppleTalk-Zone	39
(reserved for accounting)	40-59
CHAP-Challenge	60
NAS-Port-Type	61
Port-Limit	62
Login-LAT-Port	63

Table 8: RADIUS Attributes defined by RFC 2865[19]

List of Figures

1 ZyXEL SSID based guest networks [3] 3
2 ZyXEL guest network options [3] 4
3 ZyXEL WPS configuration [3] 4
4 OpenWrt's generic flash layout [13] 11
5 OverlayFS construct [16] 12
6 Port-based Authentication via physical link [20] 15
7 Wireless LAN virtual link [20] 16
8 802.11 association process 16
9 802.1x primary components 17
10 802.1x port control [22] 18
11 802.1x authentication process 20
12 Communication between supplicant and authenticator[20] 21
13 Communication between supplicant and authentication server[20] 24
14 EAPoL encapsulation[20] 24
15 EAPoL packet structure 25
16 EAP layers[25] 27
17 EAP packet header 28
18 EAP-Request/Response packet format 29
19 EAP-Success/Failure packet format 30
20 EAP Method packet format 31
21 EAP-MD5 three-way handshake [20] 32
22 EAP-TLS authentication process 34
23 RADIUS packet structure 37
24 Authenticator field structure and content 39
25 RADIUS Attribute field structure 40
26 Router and switch structure [38] 43
27 Default and Guest VLANs 47
28 User management 60
29 Create user 63
30 User details 63
31 Group management 64
32 Create group 64

33 Network/VLAN overview . . . 65
34 Minimalistic application interface . . . 69
35 Memory intensive processes . . . 73

List of Tables

1 OpenWrt's architecture and system components . . . 9
2 EAPoL Type field values . . . 26
3 EAP Code field values . . . 28
4 Standards-based EAP types . . . 30
5 Comparison of authentication mechanisms [25] . . . 36
6 RADIUS Code field values . . . 38
7 EAP-Method types . . . 77
8 RADIUS Attributes defined by RFC 2865[19] . . . 78

Listings

1 UCI configuration file example . . . 13
2 UCI example . . . 14
3 Default DHCP configuration change . . . 46
4 Default Fireweall configuration change . . . 46
5 Guest network configuration . . . 46
6 Overridding VLAN ID . . . 47
7 DHCP configuration . . . 48
8 Firewall guest network configuration . . . 48
9 Wireless configuration . . . 49
10 `radiusd.conf` listen section . . . 51
11 `radiusd.conf` log section . . . 52
12 `radiusd.conf` modules section . . . 52
13 `eap.conf` eap section . . . 53
14 `client.conf` localhost section . . . 54
15 `sites/default` authorize section . . . 55
16 `sites/default` sql configuration . . . 55
17 Group definitions . . . 57

18 User definitions 58
19 User to Group assignment 58
20 Group wide attributes 58
21 Groupless user's reply section 59
22 Guest user credential generation 61
23 Class `uci` 66
24 Final QRCode string 68
25 Client side 70
26 Server side 71

References

[1] R. Moskowitz, "The intra-psk attack," 2003, wiFi Net News [Online; Status 10/21/2015]. [Online]. Available: http://wifinetnews.com/archives/2003/11/weakness_in_passphrase_choice_in_wpa_interface.html

[2] G. Haris, *Wireshark: How to Decrypt 802.11*, 2015, [Online; Status 10/21/2015]. [Online]. Available: https://wiki.wireshark.org/HowToDecrypt802.11

[3] Zyxel knowledge base. [Online]. Available: https://kb.zyxel.com/KB/searchArticle!gwsViewDetail.action?articleOid=012595&lang=EN

[4] J.-A. Lee, J.-H. Kim, J.-H. Park, and K.-D. Moon, "A secure wireless lan access technique for home network," in *2006 IEEE 63rd Vehicular Technology Conference*, vol. 2, May 2006, pp. 818–822.

[5] T. S. Kim, Y. K. Kim, B. B. Lee, S. W. Ryu, and C. H. Cho, "Designs of a secure wireless lan access technique and an intrusion detection system for home network," in *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, vol. 1, Sept 2008, pp. 318–324.

[6] S. Onno, R. Gelloz, O. Heen, and C. Neumann, "User-based authentication for wireless home networks," in *Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on*, Sept 2012, pp. 218–220.

[7] Openwrt attitude adjustment (12.09 final) release notes. [Online]. Available: https://forum.openwrt.org/viewtopic.php?id=43764

[8] Openwrt website. [Online]. Available: https://wiki.openwrt.org

[9] The uci system. [Online]. Available: https://wiki.openwrt.org/doc/uci

[10] Luci wiki. [Online]. Available: https://github.com/openwrt/luci/wiki

[11] Openwrt build system. [Online]. Available: https://wiki.openwrt.org/about/toolchain

[12] F. Fainelli, "The openwrt embedded development framework," 2008. [Online]. Available: http://www.victek.is-a-geek.com/Repositorios/Linksys/Firmware/OpenWRT/presentation.pdf

[13] The openwrt flash layout. [Online]. Available: https://wiki.openwrt.org/doc/techref/flash.layout

[14] The uci filesystem. [Online]. Available: https://wiki.openwrt.org/doc/techref/filesystems

[15] N. Brown. Overlay filesystem. [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

[16] Overlayfs in practice. [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/

[17] "Ieee standard for local and metropolitan area networks–port-based network access control," *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pp. 1–205, Feb 2010.

[18] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible authentication protocol (eap)," Internet Requests for Comments, RFC Editor, RFC 3748, June 2004, http://www.rfc-editor.org/rfc/rfc3748.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3748.txt

[19] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (radius)," Internet Requests for Comments, RFC Editor, RFC 2865, June 2000, http://www.rfc-editor.org/rfc/rfc2865.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2865.txt

[20] J. Geier, *Implementing 802.1 X security solutions for wired and wireless networks*. John Wiley & Sons, 2008.

[21] "Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pp. 1–1076, June 2007.

[22] Aaa and 802.1x authentication. [Online]. Available: https://networklessons.com/cisco/ccie-routing-switching/aaa-802-1x-authentication/

[23] "Ieee draft standard for local and metropolitan area networks: Media access control (mac) bridges (revision of ieee std 802.1d -1998 incorporating ieee std 802.1t -2001 ieee std 802.1w -2001) (replaced by 802.1d-2004)," *IEEE Std P802.1D/D4*, 2003.

[24] W. Simpson, "The point-to-point protocol (ppp)," Internet Requests for Comments, RFC Editor, STD 51, July 1994.

[25] Y.-P. W. Jyh-Cheng, "Extensible authentication protocol (eap) and ieee 802.1x: Tutorial and empirical experience," December 2005.

[26] V. Narayanan and L. Dondeti, "Eap extensions for eap re-authentication protocol (erp)," Internet Requests for Comments, RFC Editor, RFC 5296, August 2008.

[27] S. D. D. Mary Cindy Ah Kioon, ZhaoShun Wang, "Security analysis of md5 algorithm in password storage," 2013.

[28] B. Aboba and D. Simon, "Ppp eap tls authentication protocol," Internet Requests for Comments, RFC Editor, RFC 2716, October 1999.

[29] Eap-ttls configuration on windows 7 using securew2. [Online]. Available: http://www.eduroam.ie/userdocs/win7-securew2-ttls.php

[30] S. Sotillo, "Extensible authentication protocol (eap) security issues," *Dept. of Technology System East Carolina University*, 2007.

[31] N. Asokan, V. Niemi, and K. Nyberg, "Man-in-the-middle in tunnelled authentication protocols," 2002. [Online]. Available: http://eprint.iacr.org/2002/163

[32] P. Funk and S. Blake-Wilson, "Extensible authentication protocol tunneled transport layer security authenticated protocol version 0 (eap-ttlsv0)," Internet Requests for Comments, RFC Editor, RFC 5281, August 2008, http://www.rfc-editor.org/rfc/rfc5281.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5281.txt

[33] ——, “EAP Tunneled TLS Authentication Protocol Version 1 (EAP-TTLSv1),” Internet Engineering Task Force, Internet-Draft draft-funk-eap-ttls-v1-01, Mar. 2006, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-funk-eap-ttls-v1-01

[34] V. Kamath and A. Palekar, “Microsoft EAP CHAP Extensions,” Internet Engineering Task Force, Internet-Draft draft-kamath-pppext-eap-mschapv2-02, Jun. 2007, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-kamath-pppext-eap-mschapv2-02

[35] L. J. Blunk and J. R. Vollbrecht, “Ppp extensible authentication protocol (eap),” Internet Requests for Comments, RFC Editor, RFC 2284, March 1998, http://www.rfc-editor.org/rfc/rfc2284.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2284.txt

[36] G. Zorn, D. Leifer, A. Rubens, J. Shriver, M. Holdrege, and I. Goyret, “Radius attributes for tunnel protocol support,” Internet Requests for Comments, RFC Editor, RFC 2868, June 2000.

[37] Quality of service (qos-scripts) configuration. [Online]. Available: https://wiki.openwrt.org/doc/uci/qos

[38] Openwrt switch configuration. [Online]. Available: https://wiki.openwrt.org/doc/uci/network/switch

[39] H. P. Long. Freeradius server: How to install. [Online]. Available: http://itnetwork-infrastructure.blogspot.co.at/2012/01/freeradius-server-how-to-install-part-2.html?m=1

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, ________

____

Stefan Neuhuber