Incentivize decentralized WiFi roaming through VPN on home routers

Master thesis Security and Network Engineering, University of Amsterdam

Sander.Lentink@os3.nl Peter.Boers@surfnet.nl

2019-12-05

Abstract

It is infeasible for most consumers to safely share their internet connection through an open Wi-Fi. This research explores the possibility of dynamically whitelisting VPN endpoints on Wi-Fi networks by providing a client's VPN endpoint details in the 802.1x identity. This reduces the liability concerns of Wi-Fi Access Point (AP) providers and forces clients accessing these APs to use a VPN, which will increase their privacy. We modified a RADIUS server to implement our proposed protocol and verified it through a Proof of Concept (PoC).

1 Introduction

Mobile devices use multiple wireless solutions, with Wi-Fi being "the best technology for Mobile Data Offloading (MDO)" (Gupta and Rohil 2012). Providing internet access through free wireless has been seen as politeness towards guests, but concerns around security, violating terms or possible abuse for illegal content prevent individuals from providing open wireless networks (Schneier 2008). Larger entities who want to provide wireless services face different challenges. There are laws in place that prevent municipalities from providing "free WiFi" (Chamberlain 2019) and telecommunications companies lobby to prevent new projects (Gurley and O'Shaughnessy 2019).

When it comes to users accessing Wi-Fi networks, many are unaware of privacy risks (Consolvo et al. 2010). Those who are aware of the risks, place trust in the Wi-Fi provider or take extra security measures.

We can remove this trust issue between both parties when accessing clients tunnel traffic via their home location (Sastry, Crowcroft, and Sollins 2007). This restricts the traffic passing the Wi-Fi provider to VPN traffic and creates privacy for the client.

The goal of this research is to develop a protocol that allows secure Wi-Fi sharing.

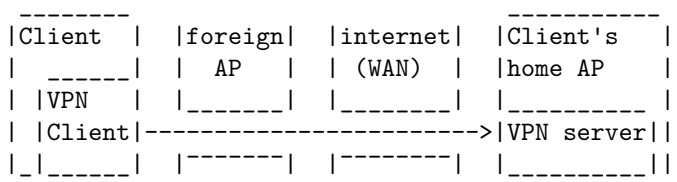

Figure 1: Client connects to VPN server at home location

When clients connect to an AP, all their traffic goes to their own VPN tunnel endpoint, as shown in fig. 1.

1.1 Research question

To understand the research question(s), an example overview is given in fig. 2, where the client connects to an AP using a VPN.

Figure 2: Client connects to VPN endpoint via foreign AP

- Can we design a protocol using existing protocols available on commercially off the shelf (COTS) clients that eliminates the need for trust between client and Wi-Fi provider by using a VPN tunnel?
  1. How can the VPN endpoint details be communicated from the supplicant to the authentication server?
  2. What modifications or configurations will enable an authentication (802.1x) server to set network policies based on a VPN endpoint?
  3. Which network policies are of interest to this protocol?
  4. Can we verify this protocol through a Proof of Concept (PoC)?

1.2 Related work

When it comes to regular APs, the AP operator is able to see the traffic of the clients it facilitates, giving the client less privacy and the operator more control on the network content. In the context of Open Wireless, the EFF states "that operators of open networks sometimes worry that they could be legally responsible if people use their networks to engage in copyright infringement."[1] The topic of copyright for Open Wireless is discussed in a white paper ("Open Wi-Fi and Copyright: A Primer for Network Operators" 2014). Clients using our

[1] https://www.eff.org/wp/open-wi-fi-and-copyright-primer-network-operators

protocol are merely allowed to use VPN traffic, making the AP a mere passive conduit, unable to read the traffic.

We need traffic prioritization to prevent "free riding" by neighbors (Eckersley 2011).

For National Research and Educational Networks it was found that authentication via 802.1x was more safe than a web portal and more scalable than using VPN-based authentication (Wierenga and Florio 2005). They also mention that standardization for wireless configuration is required to prevent reconfiguration at different locations. In our research the 802.1x authentication is used to provide the AP with the VPN endpoint location. The actual authentication is done by validating this VPN endpoint, instead of the password field.

The Open Garden protocol plans to make every hotspot a VPN exit node, creating a decentralized VPN setup (Hainsworth 2018). This matches our research where every AP SHOULD be a VPN server.

Unaware users can be made aware of their behavior in order to improve security and privacy (Consolvo et al. 2010). However, we propose a technical solution instead of increasing awareness.

In the context of SlyFi; "for such infrastructural solutions to be effective, they need both to be incorporated into wireless standards and to become widely deployed." (Klasnja et al. 2009) However, we will use the protocols available on COTS clients.

Wi-Fi Protected Access version 3 (WPA3) enables privacy between clients by providing individualized encryption. However, this does not prevent the AP provider from eavesdropping. Our research will use a VPN tunnel, creating **end** (client) **to** VPN **end**point **encryption**.

When providers enable Wi-Fi sharing (e.g. using Fon) between homes, they could enable Wi-Fi roaming, preventing the IP address of the client to change when walking through a street. This can also be done using this protocol at a larger network (e.g. campus), but not between houses (independent APs).

Modern web browsers are starting to use encrypted DNS, which will increase the client's privacy (NCSC 2019). However, we propose to use a VPN tunnel, which also hides additional data such as the server name indication (SNI).

1.2.1 Related wireless solutions

We present various Wi-Fi solutions to get an impression of existing approaches.

Open to selective users There are Wi-Fi initiatives for specific groups of people, such as Eduroam for students and Govroam for government employees. Various internet service providers (ISPs) configure routers at home locations as Wi-Fi APs for their customers, such as Comcast's xfinitywifi, KPN's Fon or British Telecom's BT Wi-fi.

Open to all users Various initiatives exist to provide free wireless for communities and residential areas. Fon provides residential WiFi in European cities and has its own router, the Fonera (Ojeda-Zapata 2014). Fon has an option for

paid users called “Bill” and the “Linus” option to provide free access (Schneier 2008). Other initiatives use mesh networking to create the wireless network, such as FunkFeuer in Austria, Freifunk in Germany and the Open Garden protocol.

Airports often provide broker based solutions, such as iPass, Tmobile/Vodafone hotspots, or Boingo. Alternative forms of payment include ad based solutions, examples are World Wi-Fi and the three-stage Stackelberg game based platform (Yu et al. 2017).

Open to all providers and users The following Wi-Fi solutions allow anyone to create an AP.

The Commotion Construction Kit provides a “guide to building community wireless networks”. The Open Wireless Movement provides software that can be installed on OpenWrt compatible routers. This movement is backed by the Electronic Frontier Foundation. Other solutions offer Wi-Fi in exchange for points/data/credits, examples are Karma and the Open Garden protocol (which uses VPN) (Hainsworth 2018).

2 Method

This section provides a *Protocol introduction* by showing an example sequence diagram. Next we start the protocol definition by looking at the requirements for a supplicant (section *Protocol: Supplicant*) — the one initiating the protocol — followed by the specification for the AP (section *Protocol: AP Specification*). We conclude by describing the *Test setup*.

2.1 Protocol introduction

Participating APs have a shared procedure of authenticating and authorizing clients. We will introduce this shared procedure (the protocol) by explaining the connection setup between a supplicant and an AP.

In the sequence diagram shown in fig. 3, `hostapd` is the Wireless Access Point (WAP), which is the Network Access Server (NAS). In this example *Test setup*, all aspects of the protocol are implemented on the same system on chip (SoC).

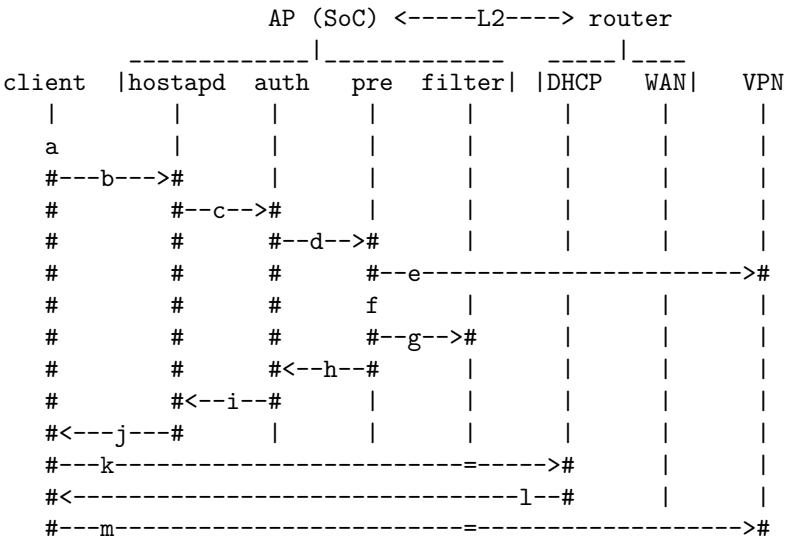

Figure 3: Sequence diagram of example implementation of the protocol

a. Supplicant (client) scans for an AP and finds a foreign AP that matches the *Wireless configuration* (SSID)
b. Supplicant connects to the authenticator (`hostapd`) using *802.1x credentials*, which communicate the VPN endpoint's *Protocols and ports* in the *802.1x identity*
c. Authenticator connects to the *802.1x authentication server*
d. This authentication server uses a custom *Pre-authorize* script
e. Script validates that provided information points to a VPN endpoint.
f. `if VPN: continue else return` 802.1x rejected
g. Script creates *Network policies*, these *Implemented policies* are set to whitelist the VPN endpoint
h. OK
i. OK
j. 802.1x client accepted (`wlan` bridged with `eth0`; L2 access to router)
k. client requests DHCP lease (IP address)
l. router provides IP to client (thus NAT in router)
m. client connects to *VPN server*

The custom code for this protocol is implemented in the *Pre-authorize* script.

In larger networks (e.g. campus) we expect the WAP (`hostapd`) to be on different hardware than the 802.1x authentication server.

2.2 Protocol: Supplicant

We have seen the *Protocol introduction*, which showed that the supplicant initiates the protocol. We will now look at the configuration required on the supplicant (client).

The client needs to configure two aspects, the 802.1x *Wireless configuration* and

a VPN client. The *Wireless configuration* is part of this protocol and is created with the following information:

- domain or IP of *VPN server*
- IP *Protocols and ports* used to connect to *VPN server*
- will the supplicant validate the 802.1x certificate?

This section details the configuration of the client, these settings are motivated in section *Protocol: AP Specification*.

2.2.1 802.1x credentials

We now look at the 802.1x credentials, which will encode the VPN endpoint in the anonymous identity, as described in section *802.1x identity*. This will allow foreign APs to whitelist traffic to it.

Figure 4: Supplicant sets network configuration (snippet from fig. 3)

The clients needs to add a new Wi-Fi network (fig. 4), using the configuration shown in table 1.

Table 1: Network configuration on supplicant

field	value	default
SSID	“tunroam.org 19”	
Security	802.1x EAP	
EAP type	Protected EAP (PEAP)	
phase 2	MSCHAPv2	
CA certificate	*variable*	no validation
anonymous identity	*variable*	
identity	*variable*	
password	*variable*	“password”

The *variable* fields in table 1 are dependent on the *validate_certificate* flag being set. The static values are motivated in section *Wireless configuration*.

When the client connects to an AP (fig. 5), the anonymous identity is used to communicate the VPN endpoint to the AP.

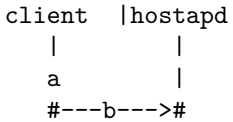

Figure 5: Supplicant connects to AP (snippet from fig. 3)

Clients have the option to indicate that their 802.1x request MUST be proxied to their own server (to get a known certificate).

validate_certificate disabled Without certificate validation, the AP does not proxy the 802.1x request, giving the supplicant a different certificate at every location.

Example: When we describe the identities for an OpenVPN/WireGuard VPN server running on port 443 UDP (IP protocol 0x11), it is encoded as `11443` and has the flag character appended (as described in section *802.1x identity*):

`11443a@<ADDRESS-VPN-SERVER>`

where `<ADDRESS-VPN-SERVER>` is an IP address or fully qualified domain name (FQDN).

The password field needs to have the literal value "password", to allow the authentication server to complete the Challenge-Handshake Authentication Protocol (*CHAP*) procedure.

validate_certificate enabled If the supplicant desires to validate the certificate, or has no option/rights to disable validation; it needs different values for the identities and password fields.

In the following example of an anonymous identity, the flag character (`b`) indicates the request needs to be proxied:

`11443b@<FQDN-EAP-SERVER>`

The regular identity is not exposed to the AP but only seen by the server providing the certificate. The realm part (`<FQDN-EAP-SERVER>`) MUST be the same for both identities from table 1 (as described in section *Identities*). The user part of the identity and the password MAY be specified for the specific needs of the external authentication server proxied to.

Multiple protocols and ports To allow clients to use various *VPN protocols*, one can define multiple protocols and ports. These are separated by an underscore, as seen in the following IPsec example:

`11500_32_33_114500a@10.10.10.10`

The corresponding values are explained in table 2.

Table 2: Protocols and ports for IPsec example

value	IP proto. id	IP protocol name	port	description
`11500`	`0x11`	User Datagram Protocol (UDP)	500	Internet Key Exchange (IKE)
`32`	`0x32`	Encap Security Payload (ESP)	N/A	
`33`	`0x33`	Authentication Header (AH)	N/A	
`114500`	`0x11`	User Datagram Protocol (UDP)	4500	NAT-T (RFC 3947 sec. 4)

The AP MAY only allow traffic for the *L4 Transport layer* protocols it supports

and were found valid (i.e. listening).

2.2.2 VPN

The AP expects the client to connect to a VPN endpoint, to which it will allow (whitelist) traffic.

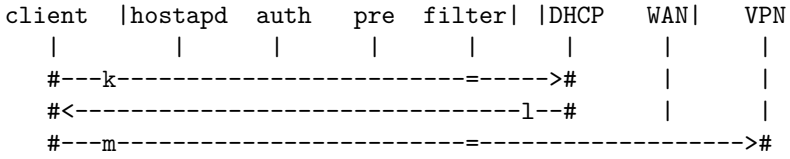

Figure 6: Client connects to VPN (snippet from fig. 3)

The client MAY use the DNS server provided by the DHCP server of an AP to resolve its VPN endpoint, as shown in step `l` of fig. 6.

The client SHOULD use a VPN client that connects to the VPN server using UDP (e.g. OpenVPN, WireGuard or IPsec using NAT traversal as defined in RFC3947 section 4). VPN clients that use other protocols MAY be supported by the AP, as described in section *L4 Transport layer*.

A rogue AP could accept a client and use the VPN endpoint information to only block the VPN, hoping the client will ignore the VPN. Therefore the client SHOULD only allow traffic via the VPN when connecting to APs.

2.3 Protocol: AP Specification

In the previous section (*Protocol: Supplicant*) we read what the supplicant communicates to the AP. We will now look at the authentication server the AP uses to process authentication requests.

We start by defining/parsing the *802.1x identity* received by the supplicant, followed by the *802.1x authentication server* that receives this data. This section concludes with *Network policies* the AP SHOULD implement.

A VPN server SHOULD be installed on an AP when configured for a home location. However, configuration and VPN identity management is not part of this protocol.

The goal of this protocol is to allow an AP to share Wi-Fi without the facilitator being liable for the traffic generated by clients. We do this by only allowing VPN traffic and *DNS* to locate the VPN server to be seen by the ISP of that AP. The client will appear on the internet with the IP address of its VPN server.

2.3.1 802.1x identity

First we clarify that the 802.1x identity is separate from the VPN credentials. The 802.1x identity only states the VPN endpoint, not the user login for the VPN.

There are two identities/usernames (as shown in table 1), 1) an anonymous (provided to the authentication proxy) and 2) the identity provided to the authentication server that provides the certificate. We call the anonymous identity the `anonid` and the regular identity the `innerid`. The `anonid` is handled at the AP and thus needs the VPN information.

A practical example of parsing the `anonid` is given in section *Pseudo code parse identity*.

Identities Identities consist of a user part and a realm/domain part, separated by the `@` delimiter (i.e. `userpart@realmpart`). The `anonid` MUST contain the same realm as the `innerid` since Windows copies the realm part of the `innerid` and uses it in the `anonid` ("PEAP Identity Privacy Support in Windows7" 2009).

2.3.1.1 Protocols and ports

The `anonid` starts with the user part, which starts with listing IP protocols and ports. The IP protocols are discussed in section *L4 Transport layer*.

The supplicant communicates the IP protocols to the authentication server by the IP protocol id in hexadecimal. Additional information (i.e. port number) is appended to the id. If multiple are given, they are separated by an underscore.

An example for port 443 TCP (0x06) and UDP (0x11):

`06443_11443f@10.10.10.10`

The AP MUST accept at least the first 3 tuples it supports. If the AP only allows UDP (0x11), it MUST be able to retrieve the UDP configuration if unsupported protocols are preceding in the identity (e.g. `32_33_2f_06443_11443a@10.10.10.10`). The AP SHOULD accept the supplicant when — of the supported protocols — a subset of the ports were found valid (e.g. when `0622` and `11443` are in the identity and no UDP 443 socket is detected, only TCP 22, it MUST whitelist TCP 22 and accept the 802.1x request).

The AP MUST accept/reject supplicants based on the information provided in the identity. It MUST NOT present a captive portal.

Motivation While WireGuard and OpenVPN are protocols that only need one UDP port, IPsec requires more ports. Therefore we require at least 3 tuples to be processed. More on VPN protocols can be found in section *VPN protocols*.

The supplicant MUST be accepted when only a subset is found valid, to allow configurations such as `32_33_11500_114500` (IPsec).

2.3.1.2 Flags

The flag character is appended to the list of tuples, making it the last character of the user part.

This flag character is a base32 character as defined by RFC4648, to represent 5 bits. We have defined only the first (least significant bit) for this flag character; the `validate_certificate` flag, as shown in table 3. The flag character is an '`a`' if the supplicant does not validate the 802.1x authentication certificate (default) or a '`b`' when the supplicant does want to validate the certificate. When

the supplicant does desire to validate the certificate, the AP MUST proxy the request.

Table 3: Bits in flag character

bit	name	description
0000?	`validate_certificate`	Validate 802.1x certificate?
000?0	RESERVED	
00?00	RESERVED	
0?000	RESERVED	
?0000	RESERVED	

Motivation We place the flag before the default delimiter (`@`), to enable it to be included in the delimiter, becoming a custom delimiter (`f@`). It is prefixed and not appended to require no parsing of the realm part when using the default delimiter. We note that the realm module of FreeRADIUS requires a single character as delimiter.

We pick base32 for the flag notation from the RFC instead of base64 to make the identity case insensitive.

2.3.1.3 Hostname After the delimiter we either have an IP address or an FQDN.

If the realm part of the identity contains an FQDN instead of an IP address, it MUST contain "`tunroam.`". This subdomain requirement MUST be enforced in the authentication server when validating the identity and MAY also be enforced with DNS. The authentication request MUST be rejected if this requirement is not met.

All IPv4 addresses MUST be in the dotted-decimal notation (RFC1123). IPv6 addresses MUST be in the format specified in RFC5952, without square brackets (used in URLs).

IPv4 MUST be supported by the AP. IPv6 SHOULD be supported by the AP.

Motivation Section *Wireless configuration* motivates the name of the protocol. We use this name (TUNroam) as the required subdomain.

This subdomain MUST be enforced using the `anonid` since APs MAY allow encrypted DNS, resulting in this requirement not being verified when just using DNS filtering. This requirement is set to avoid inconsistencies across APs, where some do want to filter *DNS* and others do not.

The VPN client MAY require a DNS query to resolve the VPN endpoint. To allow APs to filter these requests we require the realm part of the 802.1x identity to include a specific subdomain. This requirement is to protect the AP provider from unintended DNS queries being logged by its ISP.

Mobile devices can contain apps that indicate a lifestyle (e.g. Grindr; gay dating app) that is celebrated with parades in some countries and can lead to the death

penalty in others (Amnesty 2019). When a client's VPN is off, such apps could do DNS queries, which can be blocked since they do not include the specific subdomain.

The following pseudo filter rule allows egress packets containing 'tunroam' on port 53 (DNS):

```
iptables-nft -I OUTPUT -j ALLOW --algo bm \
  -p udp --dport 53 \
  --match string --hex-string "|07|tunroam|"
```

2.3.2 802.1x authentication server

The Wireless AP uses an authentication server to validate connecting supplicants. This authentication server MAY implement the proposed protocol in the *Pre-authorize* phase, as shown in fig. 7.

```
client  |hostapd  auth   pre
   |        |      |      |
   #        #--c-->#      |
   #        #      #--d-->#
```

Figure 7: Authentication server uses *Pre-authorize* script (snippet from fig. 3)

2.3.2.1 validate_certificate When the `validate_certificate` flag is set in the `anonid`, `vpn.<DOMAIN-IN-INNERID>` is resolved for the VPN address and `<DOMAIN-IN-INNERID>` for the authentication server. If a supplicant provides an IP address instead of an FQDN, the AP MUST use this IP address for both the authentication and VPN server.

The request proxied to the external server MUST use the RADIUS protocol (RFC 2058) on UDP 1812 (not to be confused with UDP 1645, as mentioned in RFC2138) using the secret "testing123" (FreeRADIUS default).

Example FreeRADIUS client.conf:

```
client acceptall {
    ipv4addr = *    # any
    proto = udp
    secret = testing123
}
```

The `innerid` MAY hold actual user credentials, not used by the AP but by the authentication server proxied to.

If the AP desires to do VPN endpoint validation, it MUST complete the validation step before proxying the authentication request. If the VPN endpoint validation fails, the authentication request is rejected without forwarding the request.

Motivation Both the secret for proxying the RADIUS request ("testing123") and the identity validation when not proxying the RADIUS request ("password") are weak. These passwords are not used to secure the protocol, but to enable two

parties who have no prior agreement, to share a secret so they can communicate with each other.

802.1x supplicants like iOS require the validation of a certificate and other systems (e.g. Windows) require admin rights to disable validation.

Some supplicants have the option to disable certificate validation, allowing every AP to use a unique self signed certificate. For clients that validate certificates (e.g. Windows), we need to present a certificate that is trusted by the client. We have two options 1) provide a certificate that is signed by a trusted certificate authority like Let's Encrypt 2) proxy the request to a authentication server the supplicant trusts. However, the first approach requires the private key to be shared to every AP, allowing anyone to revoke this certificate.

We suggest to install the specific server certificate in the supplicant, since validating the common name (CN) of a certificate works for a DNS name but is not feasible for all supplicants (e.g. Android 9 supports it, on Chrome OS it is not an option).

Two DNS records are needed to allow the authentication server and VPN server to be on a different IP address.

The realm part of the authentication server stays the same, requiring no modifications when proxying the request. Since the VPN endpoint validation requires an update to the authentication server (to support this protocol) we require this custom script to implement the DNS logic (i.e. prefix `vpn.`).

2.3.2.2 Stripping realm When the supplicant indicates (by setting the `validate_certificate` flag) it needs the authentication request to be proxied, the authentication server serves as a Network Access Server (NAS). The server MAY strip the realm part of the identity when forwarding the authentication request.

Motivation The external RADIUS server proxied to enable the supplicant to be presented with the same certificate at each AP it visits. It is in the interest of the supplicant — who is the maintainer of the authentication server proxied to — that it accepts all requests at all time. Thus the server does not need the realm information.

We allow the proxying authentication server to `strip` the realm part of the identity. This is the default behavior of FreeRADIUS. When desired, the server responding to the request MAY obtain its realm via reverse DNS.

2.3.3 Wireless configuration

Participating APs have the wireless configuration described in this section. This shared configuration allows clients to automatically connect to a foreign AP.

The service set identifier (SSID) MUST be "tunroam.org 19".

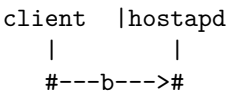

Figure 8: Client connects to AP (snippet from fig. 3)

Clients MUST be able to connect (fig. 8) using PEAP with MSCHAPv2 as discussed in section *EAP protocols*.

Motivation Versioning is possible with different SSIDs. The number in the required SSID refers to the 2019 version of the protocol. If new versions of the protocol would require the client to provide different information, the version number SHOULD be set to the year the new version is released. Clients MAY configure settings for multiple SSIDs, which then can connect to the version available.

We observed that public Wi-Fi hotspots often include the words "free" and "wifi". We did not pick the words "Open" (referring to Open Source, open to anyone to join the protocol) since it could be misunderstand as an open wireless network. We did not use "Free" since it requires the client to initially setup a VPN endpoint and users have other expectations of "free wifi", "wifi" in a Wi-Fi SSID is a pleonasm.

While the name Tunroam has resemblance to Eduroam and Govroam, this project is not associated with their brand or product. Roam stands for roaming, allowing clients to connect in multiple locations. Tun refers to VPN **tun**nel and "tun" in:

> What is the difference between TUN driver and TAP driver? TUN works with IP frames. TAP works with Ethernet frames.[2]

2.3.4 Network policies

It is in the interest of the AP provider to filter traffic, therefore we state that it MAY enforce network policies (fig. 9) on the link layer (L2) and SHOULD on L3 and L4. However, no network policies are required for a valid implementation.

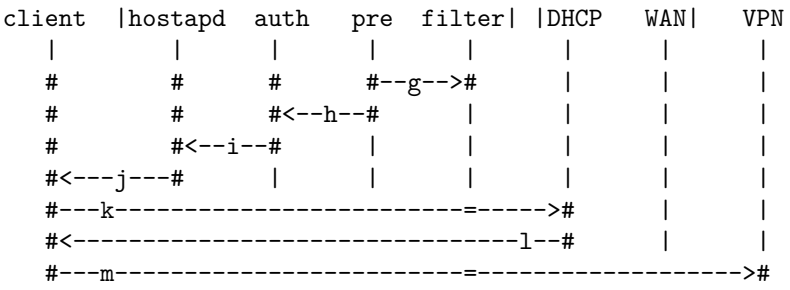

Figure 9: Setting and enforcing network policies (snippet from fig. 3)

[2] https://www.kernel.org/doc/Documentation/networking/tuntap.txt

The AP SHOULD allow all incoming traffic and only limit the outgoing, preventing the need for connection tracking.

We acknowledge that the proposed filtering/whitelisting is minimal and that one might circumvent it by using it as a covert channel. However, we expect there to be little incentive for this type of abuse, since anyone can join using a free VPN provider.

In the example implementation (fig. 3), the network policies are set in the authentication flow. When clients stay connected for an extended period, their DHCP lease MAY be renewed, but the network policies MAY not be renewed.

Network policies MUST be valid for at least 12h (or at least as long as the DHCP lease), after which the supplicant MAY need to reconnect to have access to its VPN again. A longer time span such as 24h MAY result in a laptop having to reconnect during work the next day.

2.3.4.1 L2 Link layer The protocols used on the link layer are up to the implementer of this protocol. Expected protocols are ethernet (IEEE 802.3) and Address Resolution Protocol (ARP) (RFC826) for IPv4. The AP MAY limit ARP to prevent MAC learning of other devices connected to the network and detect MAC spoofing.

We note that L3 filtering by a router is bypassed when an attacker knows the MAC address of a device connected on the link layer. Therefore one MAY keep a whitelist of MAC addresses the supplicant is allowed to connect to or provide other ways of separation (e.g. VLAN).

Suggestions on filtering frames can be found on kernel.org[3].

Motivation The implementer is responsible for any security related issues with providing a client with access to the private network. This might expose devices 'protected' by NAT such as Network Attached Storage devices.

2.3.4.2 L3 Internet layer IP packets from the client MAY be limited to network/connection management (e.g. DNS lookup for VPN endpoint) and the VPN endpoint.

The AP MAY have one segregated network (e.g. VLAN) for all clients connecting to the AP, allowing L3 filtering.

The 802.1x authentication is needed to provide the AP with the domain or IP address to which it will allow the client to connect. The AP SHOULD only route packets from the client to the VPN IP address.

2.3.4.3 L4 Transport layer The AP SHOULD filter based on the correct IP protocol and port.

To enable the client to use multiple *VPN protocols*, the requirements in table 4 are given.

[3]https://wireless.wiki.kernel.org/en/developers/openwirelessmovement

Table 4: IP protocols

IP protocol	ID	RFC	NAT	public IP
Transmission Control (TCP)	0x06	793	SHOULD	SHOULD
User Datagram (UDP)	0x11	768	MUST	MUST
Generic Routing Encapsulation (GRE)	0x2F	2784	MAY	MAY
Encap Security Payload (ESP)	0x32	4303	MAY	MUST
Authentication Header (AH)	0x33	4302	MAY	MUST

NAT refers to Network Address Port Translation (NAPT) (RFC2663) and public IP when every client is given a public IP address.

When clients are provided with a public IP address, incoming traffic should also be filtered.

The AP MAY filter well-known ports (below 1024), except for 22 (socks tunnel), 443 (HTTPS tunnel) and 500 (IKE for IPsec), but MUST allow ports (for supported IP protocols) over 1024 when given in the identity.

Motivation Only UDP is considered a MUST have, which enables *VPN protocols* such as OpenVPN, WireGuard and IPsec/IKE.

An AP operator MAY want to block ports associated with email or other well-known ports. We need UDP 500 for Internet Key Exchange (IKE), which switches to UDP 4500 for NAT Traversal as defined in RFC 3947 section 4.

If the AP provides public IP addresses to supplicants, it needs to support IP protocol 0x32 ESP and 0x33 AH. IPsec does a check for NAT, when no NAT is detected it will use ESP and AH instead of NAT Traversal.

Some ISPs block certain ports (Schellevis 2014), therefore we allow clients to pick custom ports, matching their VPN endpoint.

2.3.4.4 DNS VPN clients MAY require DNS to lookup their VPN endpoint, therefore the AP MUST support Do53 (as defined by draft-hoffman-dns-terminology-ter-01). The subdomain requirement in section *Hostname* enables the AP to filter DNS queries which do not include the specified subdomain.

We see the adoption of encrypted DNS in web browsers (NCSC 2019) and the Android operating system supports[4] RFC7858 DoT. The AP MAY allow encrypted DNS for DoT using port 853 or DoH using a whitelist of trusted providers[5]. This results in the DNS query originating from the AP instead of the VPN server, without intermediates (such as an ISP) being able to log the DNS queries.

If the AP provider does not want its ISP to see any DNS queries from connecting clients, it MAY hijack Do53 DNS traffic like some ISPs do (Farrokhi 2016), since clients MAY NOT accept the DNS server from DHCP. This allows the AP to

[4]https://developers.google.com/speed/public-dns/docs/dns-over-tls
[5]https://www.chromium.org/developers/dns-over-https

tunnel the requests using a DoH proxy, which are provided by parties such as Cloudflare[6] and Facebook[7].

Motivation The ability to filter DNS gives the AP provider more safety in countries where this might be desired[8].

The client MAY NOT accept the nameserver pushed by an AP, which is motivated by:

- When users cannot use a custom DNS, they have less privacy when VPN is off.
- Users may not know how to enable automatic nameserver configuration through DHCP.
- Windows requires admin rights to change this.

2.3.5 VPN server

The AP SHOULD have a VPN server installed, enabling the operator of this AP to connect to foreign APs using this VPN server, as shown in fig. 10. This creates decentralized peer to peer tunnels between clients and their own AP (being the VPN server).

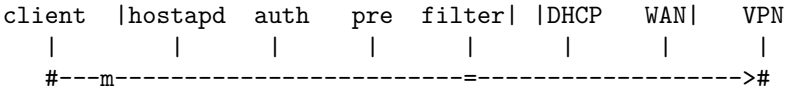

Figure 10: Client connects to VPN (snippet from fig. 3)

The VPN client will run on the mobile client, not the AP. Otherwise the AP could do malicious activities using the IP address of the VPN server.

2.4 Test setup

This section describes the Proof of Concept (PoC) created to validate the protocol. An overview of the components can be found in fig. 11.

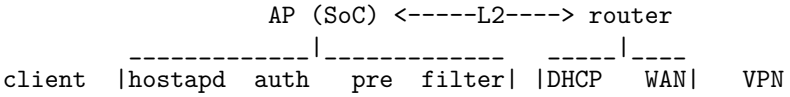

Figure 11: Components in test setup (snippet from fig. 3)

2.4.1 VPN

The VPN server used for testing was installed on a Linux server using `install-vpn-using-docker.sh`[9]. The type of VPN server for testing is motivated in section *VPN protocols*.

[6] https://developers.cloudflare.com/1.1.1.1/dns-over-https/cloudflared-proxy/
[7] https://facebookexperimental.github.io/doh-proxy/tutorials/simple-setup.html
[8] https://en.wikipedia.org/wiki/Political_repression_of_cyber-dissidents
[9] https://github.com/tunroam/scripts

2.4.2 Configuring FreeRADIUS

We use FreeRADIUS, since it "is the most popular and the most widely deployed RADIUS server"[10]. While Diameter (RFC3588) is a newer protocol, RADIUS was chosen since it is a widely adopted protocol in existing wireless infrastructures and routers.

We need three modifications on the authentication server:

- Allow our server to be reached from everywhere: for proxied requests
- Allow every user when doing the CHAP: one shared fixed password
- Install a pre-authorize script: verify identity and VPN endpoint

The first item was described in section *802.1x authentication server* and the other two can be found in section *FreeRADIUS modifications*.

Inner-tunnel The RADIUS server needs to be able to proxy the RADIUS request when the supplicant set the `validate_certificate` flag. We mimic this external authentication server by configuring a second server:

```
$ ls /etc/freeradius/*/sites-enabled
default  inner-tunnel
```

Both these RADIUS servers run on the same SoC:

```
$ ss -4lpun|grep -E "(1812|Port)"
State  Recv-Q Send-Q Local Address:Port   Peer Address:Port
UNCONN 0      0            0.0.0.0:1812        0.0.0.0:*
UNCONN 0      0          127.0.0.1:18120       0.0.0.0:*
```

The PoC is not configured to be able to proxy to external RADIUS servers, but the behavior is simulated by running two daemons.

2.4.3 Implemented policies

Section *Network policies* states that those policies are a SHOULD have. They are not implemented on the PoC for the following reasons:

- During the demo, an open WiFi (after identity validation) would be more desirable
- An actual implementation (e.g. on a campus) would not run all the components on the same device

Therefore the scripts outputs network policies instead of setting them:

```
$ validate_anonid.py 11443_06443_00testA@tunroam.lent.ink
WARNING the additional value is not a port number
INFO suggesting whitelist rules
{ 'iptables-nft -A OUTPUT -j ACCEPT -d tunroam.lent.ink \
     --protocol 17 --dport 443',
  'iptables-nft -A OUTPUT -j ACCEPT -d tunroam.lent.ink \
     --protocol 6 --dport 443' }
INFO Welcome aboard 11443_06443_00testA@localhost
```

[10] https://networkradius.com/doc/current/introduction/FreeRADIUS.html

In this output, the warning is caused by `00test`. The validation is successful since one or both of the other tuples (UDP and TCP 443) was found listening on the specified hostname.

2.4.4 Hardware setup

During development, the debugging was done on:

```
cat /proc/cpuinfo|grep Model
Model           : Raspberry Pi 3 Model B Rev 1.2
```

running the latest Raspbian OS. It was connected to the student's home router. The router was configured to do port forwarding (DMZ host option).

The configuration was captured in installation scripts[11], which were used to install a second SoC, an Orange Pi Zero Plus running the latest Armbian. This second SoC was used during the presentation demo.

These *bian distributions were chosen since OpenWrt does not support WiFi on the Orange Pi[12].

The example flow described in section *Protocol introduction* uses a bridged setup. For the presentation demo we decided to use NAT on the SoC, since this also works when the physical port — the SoC connects to in the presentation rooms — is allowed only one DHCP lease. Both the bridged and NAT setup can be created using the installation scripts[13].

3 Results

Following the infrastructure as code (IaC) paradigm, the installation scripts[14] proved that it is feasible to construct a setup that enables the proposed protocol.

When we connected a client (tested on Chrome OS and Android) to the SoC broadcasting the SSID, we were able to connect when the VPN endpoint specified in the identity pointed to a valid socket (tested with TCP and UDP). Through the logs we confirmed that the *802.1x authentication server* on the SoC does:

- *802.1x identity* parsing
- VPN endpoint validation[15]
- Is able to derive whitelist rules from the `anonid`, as seen in section *Implemented policies*

[11] https://github.com/tunroam/networking
[12] https://openwrt.org/toh/hwdata/xunlong/xunlong_orange_pi_zero_plus
[13] https://github.com/tunroam/networking
[14] https://github.com/tunroam/networking
[15] https://github.com/tunroam/auth-server/blob/57a25dc04b2b5868f7c449f7ba15de0f10fc3551/validate_anonid.py#L62

Communicating VPN endpoint

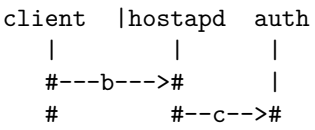

Figure 12: 802.1x identity forwarded to authentication server (snippet from fig. 3)

The protocol describes how the *802.1x identity* is used to encode the VPN endpoint's *Protocols and ports*. This identity is configured on the supplicant and communicated (via `hostapd`) to the *802.1x authentication server*, as shown in fig. 12. This answers sub-question 1 of our *Research question*.

Implementing the protocol To answer sub-question 2 of our *Research question*, we showed that the client is able to communicate the VPN endpoint using the *802.1x identity*. This requires modifications on the *802.1x authentication server*, without changes to the authentication client (`hostapd`) or supplicant (e.g. smartphone).

We described the modifications to the RADIUS server in section *FreeRADIUS modifications*.

Policies

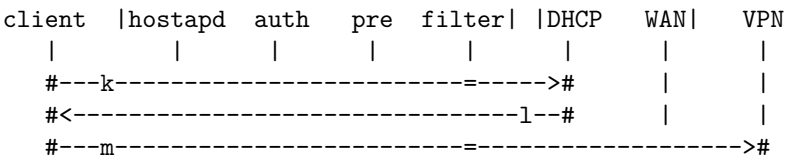

Figure 13: Network policies enforced on client (snippet from fig. 3)

To answer sub-question 3 of our *Research question*, we discussed *Network policies* that enable the AP to limit the outgoing traffic of clients to VPN traffic, as shown in fig. 13. *DNS* queries can be filtered by the AP through the required subdomain in the *Hostname*.

We showed example firewall rules that can be implemented. In our *Test setup*, these rules are set in the custom *Pre-authorize* script, used by the *802.1x authentication server*.

VPN servers might aim to appear as a different service, such as a HTTPS web server, to avoid being blocked. Therefore the current protocol only verifies if a socket is open, without any additional checks. This approach allows clients to use different *VPN protocols*.

Testing the protocol The PoC used at the presentation demo shows that the protocol is feasible. When clients attempt to connect, the authentication server validates their VPN endpoint and is able to set *Network policies*. The PoC was created using the installation scripts[16] and answers sub-question 4 of our *Research question*.

Additional The following findings do not contribute to the protocol we developed. However, they do provide insights for future research.

- The router used for our research is able to function as a RADIUS client, however, this was disabled by the ISP and updating the firmware was deemed infeasible (Lentink 2019).
- Some IoT electronics we tested only support Wi-Fi using pre-shared key (PSK).
- OpenWrt has options for remote control[17], which allows an external authentication server supporting this protocol to push network policies.

4 Discussion

Networks that only allow VPN traffic were already implemented, but we showed that it is possible to allow for dynamic VPN endpoint validation and whitelisting of specific IP protocols and ports. We use the *802.1x identity* to communicate the VPN endpoint from the supplicant to the *802.1x authentication server*.

Access Point (AP) providers can use this Open Source solution to share their internet without liability concerns. The protocol is decentralized, requiring no external dependencies for the AP provider.

Users connecting to these APs have enhanced privacy on all networks they connect to if they leave their VPN always on. Users enjoy more Wi-Fi networks they can join and are not presented a captive portal on networks implementing this protocol.

Potential AP providers include shared office space or housing, consumer routers (e.g. by using a SoC) or current open Wi-Fi providers. These providers need to configure the modified authentication server and allow it to set network policies.

Psychology of adoption Since the impact of this concept is dependent on the adoption, we need to consider that limiting some users could actually increase the adoption of APs. This could increase the network effect (as used in economics).

Clients specify the VPN they connect to, being either a VPN provider (not adding an extra AP to the network) or another AP with VPN server, as shown in fig. 14.

For larger parties providing free wifi, there is no incentive to filter clients using a VPN provider. Home providers of APs could desire to only allow clients who also provide an AP. This partisanship is also seen in MIT License versus General Public License (GPL) for software; the former allows anyone to use it, while the latter requires one to have the same shared value: open source.

[16] https://github.com/tunroam/networking
[17] https://openwrt.org/docs/guide-user/services/remote_control/ostiary.server

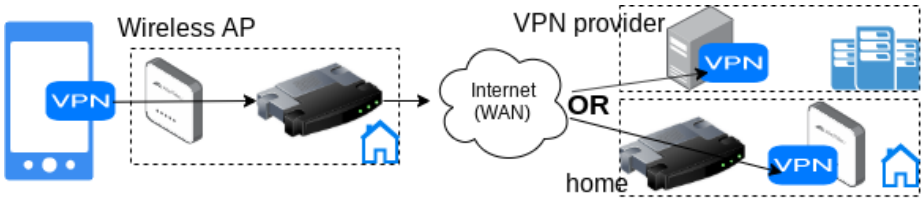

Figure 14: Client connects to VPN endpoint via foreign AP

This concept could lead to the same bigotry. However, the AP is not able to distinguish the two parties without a blacklist. And even with a blacklist of DNS records, one could duplicate the records under another apex, pointing to the public VPN provider.

The requirement of providing an external RADIUS server does not enforce this either. Offering a free RADIUS server to couple with an existing VPN provider will generate data for the RADIUS provider, creating an incentive to create such a service.

4.1 Limitations

We expect it to be unlikely that this protocol will be supported by ISP provided routers, since using a home location as a VPN endpoint will double the traffic to that router for a client when using it remotely. Using a VPN also impacts the latency and bandwidth of the client.

The specification states that the network SHOULD enforce *Network policies*, which is not implemented in our *Test setup*. While this does adhere to the specification, it is an undesirable situation. We argue that it is feasible to enforce network policies, with the tool described in section *Netfilter*, using the custom python script described in section *Implemented policies*.

Adherence specification The Proof of Concept (PoC) (created using the installation scripts) does not implement the full protocol.

The ability to set the `validate_certificate` flag is not implemented (to have the request being proxied to an external RADIUS server). We argue that this is possible (as seen in the implementation of the Eduroam network) and considered it out of scope for this research.

The PoC currently proxies the request to a second authentication server on the same device, the *Inner-tunnel*.

4.2 Future work

This section discusses some potential future work, topics that may enrich or strengthen the protocol or implementations of it.

Dynamic DNS When the VPN endpoint or the RADIUS server proxied to is located at a home location without a fixed IP address, the server should be

able to update its DNS records when the IP address changes. Besides DNS, other options exists such as *dynamic IPsec* or Tunnel Endpoint Discovery (TED). Future research could explore ways to implement this.

RADIUS as a Service If an external party offers to serve as a RADIUS server implementing this protocol; what would be the (trust/security/privacy) implications? This would allow an existing wireless infrastructure to use this external RADIUS server and would receive access control list (ACL) rules in return (if no rules, client is Rejected). The RADIUS server will receive data on connecting clients.

Malicious usage without VPN The current protocol recommends IP protocol and port number filtering. An attacker could use the identity to point to an DoH DNS resolver (e.g. `06443_11443a@[8.8.8.8|1.1.1.1]`) and use that to resolve a website. When the IP address of the website is known, the attacker disconnects and reconnects, with the IP address of the website set in the `anonid` realm. This allows the attacker to visit the website (e.g. by modifying `/etc/hosts`) without VPN, leaking the server name indication (SNI) to the ISP of the AP provider.

Future research could look into (deep) packet inspection (DPI) or other ways to avoid this potential abuse case.

DNS log poisoning When the AP provider is the target, an attacker could use `tunroam.illegalcontent.tld` in the realm part of the identity, resulting in the DNS query being observed by the AP's ISP. Future research could explore ways to mitigated this attack.

Automatic VPN client Some devices can be configured to automatically enable a VPN for specific network configurations. Future research could explore ways to dynamically enable VPN on clients for specific SSIDs.

802.11u For this research we used a shared SSID, just like Eduroam does. It might be possible to propose a new 'Access Network Type' to the 802.11u specification or use the 'Venue Name information' field.

Another option is to use the 'Network Authentication Type' field to redirect to a local server and have the required configuration stored in localStorage in the browser.

Future research could explore options to avoid the requirement of a shared SSID, allowing companies to broadcast their own name as SSID while supporting the protocol.

Additional

- Bandwidth management
- Quality of Service (QoS)
- Interoperability between IPv6 and IPv4, see section *IPv6*
- Performance measurements and optimizations

- Additional security, such as update policies for the device with exposed ports
- Passpoint (Hotspot 2.0)
- Propagating AP info using IEEE 802.11u-2011
- Create OpenWrt package of the protocol
- Can we store the VPN certificate in DNS (DANE) and retrieve it on the client (browser) using DoH?

4.3 Conclusion

We have designed a protocol using existing protocols available on commercially off the shelf (COTS) clients that eliminates the need for trust between client and Wi-Fi provider by using a VPN tunnel. Q.E.D.

We showed how the *802.1x identity* can be used to communicate the VPN endpoint of a supplicant to an AP, answering sub-question 1 of our *Research question*. Section *Configuring FreeRADIUS* details the modifications we made to the *802.1x authentication server* and section *Implemented policies* showed how the *Network policies* could be implemented, answering sub-question 2. To answer sub-question 3, we detailed *Network policies* that the AP MAY implement.

We showed through a Proof of Concept (PoC) that a FreeRADIUS authentication server is able to authorize a supplicant when they provide a valid VPN endpoint, answering sub-question 4 of our *Research question*. The authentication server is able to first validate the VPN endpoint and then whitelist the endpoint through network policies.

The TUNroam protocol enables supplicants who have a VPN client installed, to automatically connect (without captive portal) to participating Wi-Fi networks. The provider of the AP is able to safely share his internet connection through whitelisting VPN endpoints, without liability concerns.

5 Appendices

5.1 IPv6

We assume that every VPN endpoint has either one public IPv4 address (or access to it through port forwarding) or a dual stack from which the endpoint has at least one public IPv6 address and access to IPv4, either direct or through NAT.

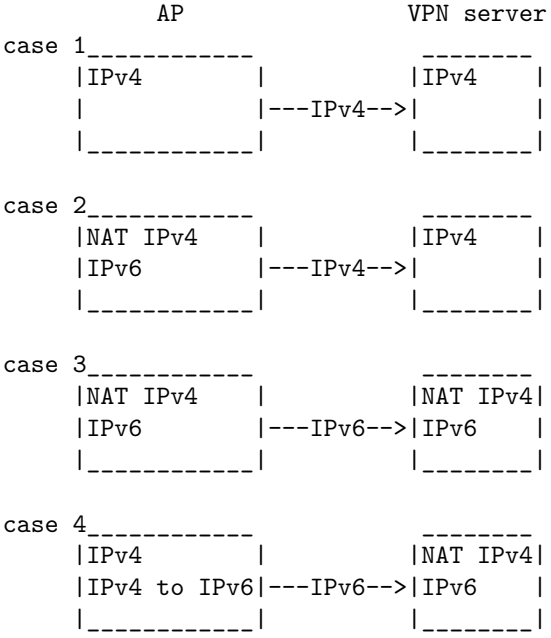

Figure 15: Possible situations of home internet connections

The first three cases shown in fig. 15 are feasible. Case 4 requires translation between v4 and v6.

We expect that with port forwarding, not having the IP on the NIC, case 4 is not possible. We consider this to be future work.

5.2 EAP protocols

This section motivates the EAP protocol we use. We looked at which EAP protocols are available by default on consumer electronics, which do not authenticate via cellular/SIM.

The mobile devices are composed of an older (5.1) and supported version (9) of the mobile Operating System (OS) with the largest market share ("Mobile Operating System Market Share Worldwide" 2019). For the desktop environments we looked at a corporate Windows laptop and Chrome OS. The data on Apple devices is retrieved from `*.apple.com`.

Android 5.1 (Moto G first gen.)

- 802.1x EAP
  - PEAP
    - None
    - MSCHAPV2
    - GTC
  - TLS
  - TTLS
    - None
    - PAP
    - MSCHAP
    - MSCHAPV2
    - GTC
  - PWD

Android 9 (Mi A1, Android ONE)

- 802.1x EAP
  - PEAP
    - None
    - MSCHAPV2
    - GTC
  - TLS
  - TTLS
    - None
    - PAP
    - MSCHAP
    - MSCHAPV2
    - GTC
  - PWD

Chrome OS (77.0.3865.35)

- EAP
  - LEAP
  - PEAP
    - EAP-MD5
    - MSCHAPV2
  - EAP-TLS
  - EAP-TTLS
    - EAP-MD5
    - MSCHAP
    - MSCHAPV2
    - PAP
    - CHAP
    - GTC

Mac

- 802.1x

  - TLS
  - EAP–FAST
  - MD5
  - LEAP
  - TTLS (“WiFi.EAPClientConfiguration,” n.d.)
    * PAP
    * EAP
    * CHAP
    * MSCHAP
    * MSCHAPv2
  - PEAP (“Mac Os X 10.5: How to Configure Network Preferences for 802.1X” 2014)
    * MSCHAPv2
    * MD5
    * GTC

iOS The following applies to “iOS 12.3 and later” (“Use Your Homepod on an 802.1X Wi-Fi Network” 2019).

- 802.1X
  - PEAP
  - EAP-TLS
  - EAP-TTLS
  - EAP-FAST

Windows 10 (1803)

- WPA2-Enterprise AES
  - Smart Card or other certificate
  - Protected EAP (PEAP)
    * Smart Card or other certificate
    * Secured password (EAP-MSCHAP v2)
  - EAP-TTLS
    * Unencrypted password (PAP)
    * Challenge Handshake Authentication Protocol (CHAP)
    * Microsoft CHAP (MS-CHAP)
    * Microsoft CHAP Version 2 (MS-CHAP v2)
    * Smart Card or other certificate
    * Secured password (EAP-MSCHAP v2)
- 802.11x WEP
  - Protected EAP (PEAP)
    * Smart Card or other certificate
    * Secured password (EAP-MSCHAP v2)
  - EAP-TTLS
    * Unencrypted password (PAP)
    * Challenge Handshake Authentication Protocol (CHAP)
    * Microsoft CHAP (MS-CHAP)
    * Microsoft CHAP Version 2 (MS-CHAP v2)
    * Smart Card or other certificate
    * Secured password (EAP-MSCHAP v2)

5.2.1 Intersection

The intersection of the observed devices gives us:

- 802.1x
  - PEAP
    - MS-CHAP v2
  - TTLS
    - PAP
    - MSCHAP
    - MSCHAPV2

However, on Windows 7 TTLS requires additional software (Fan 2016). This results in **PEAP-MSCHAPv2**, which is also suggested by FreeRADIUS[18] and is available at many Eduroam networks, such as those at the University of Amsterdam, University Utrecht, Cornell, Rijks University Groningen, Leiden University, University of Edinburgh and MIT.

5.3 VPN protocols

We reason that providers of paid VPN service want to support the VPN protocols with the largest market adoption. From the subset of these protocols we explore which one would be the best for our use case.

We took the top VPN providers from three comparisons articles. The top two ranked were NordVPN and ExpressVPN (Gewirtz 2019) (VPNdiensten 2019) (Athow 2019).

NordVPN NordVPN lists the following options[19]:

- OpenVPN
- IKEv2/IPsec

ExpressVPN ExpressVPN lists the following options[20]:

- OpenVPN
- PPTP MPPE: MSCHAP, MSCHAPv2 (PPTP is less secure)

When we look at these protocols, both IKEv2/IPsec and OpenVPN are secure solutions. Chrome OS require L2TP for using IKEv2, which requires more open ports than OpenVPN. OpenVPN allows us to filter on one port for UDP, which is what we used during our research.

5.3.1 OpenVPN

Besides requiring only one port, OpenVPN will "always first try UDP, and if that fails, then try TCP."[21] This allows our AP to whitelist UDP traffic, enabling the client to use TCP on other networks where UDP is blocked. This was the second motivation for picking OpenVPN over IPsec during testing.

[18] https://networkradius.com/doc/current/raddb/mods-available/eap/peap.html
[19] https://nordvpn.com/tutorials/linux/
[20] https://www.expressvpn.com/support/vpn-setup/#linux-setup
[21] https://openvpn.net/faq/why-does-openvpn-use-udp-and-tcp/

5.3.2 Preinstalled

We observed that OpenVPN is usually not supported by default on OSes (except Chrome OS) but IPsec based tunnels are. This made us decide that the protocol MUST support *Multiple protocols and ports*.

5.3.3 Suggestion

The VPN provides a client with access to APs implementing this protocol. The VPN will also provide additional security when using other Wi-Fi APs. We therefore suggest to use a VPN on all foreign Wi-Fi networks.

One MAY use OpenVPN with port 443 TCP and UDP. This will work on regular public Wi-Fi that have TCP 443 (HTTPS) whitelisted.

5.4 Pseudo code parse identity

We prove some example code in ECMAScript 6, which can be tested in the terminal of a web browser.

```
// https://tools.ietf.org/html/rfc4648 section 6
const BASE32_STR = 'abcdefghijklmnopqrstuvwxyz234567'
// https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
const MY_SUPPORTED_IP_PROTOCOLS = {
  '0x06' : (dest,port) => {return whitelist(dest,'tcp',port)},
  '0x11' : (dest,port) => {return whitelist(dest,'udp',port)},
  '0x32' : (dest)      => {return whitelist(dest,'esp')},
  '0x33' : (dest)      => {return whitelist(dest,'ah')}
}
function whitelist(dest, protocol, port){
  var result = "iptables-nft -A OUTPUT -j ACCEPT -d " +
               dest + " --protocol " + protocol
  if (port) return result + " --dport " + port
  else return result
}
function char_to_flags(char){
  const i = BASE32_STR.indexOf(char.toLowerCase())
  if (i === -1) return false
  // We prefix zeros for when the value is less than 16
  const binary_str = '0000' + (i).toString(2)
  return {
    validate_cert: Number(binary_str.substr(-1,1)),
    RESERVED0    : Number(binary_str.substr(-2,1)),
    RESERVED1    : Number(binary_str.substr(-3,1)),
    RESERVED2    : Number(binary_str.substr(-4,1)),
    RESERVED3    : Number(binary_str.substr(-5,1))
  }
}
function isIPaddr(str){ // up to implementer to insert regex for ipv4/ipv6
  return true
}
```

```
// WARNING this code does no input validation
function parseIdentity(id){
  const userpart = id.split('@')[0].toLowerCase()
  const realm    = id.split('@')[1].toLowerCase()
  const flags_char = userpart[userpart.length-1]
  const server_addr = realm
  if (! isIPaddr(realm) &&
        (realm.indexOf('tunroam.') === -1))
    return "ERROR required subdomain missing in hostname"
  var requested_list = userpart.substr(0,userpart.length-1).split('_')
  var rulelist = []
  for (const i of requested_list){
    const prid = '0x' + i.substr(0,2) // protocolid
    const additional_info = i.substr(2)
    if (additional_info == '53') return 'Rejected, no DNS port allowed'
    if (prid in MY_SUPPORTED_IP_PROTOCOLS){
      const rule = MY_SUPPORTED_IP_PROTOCOLS[prid](server_addr,additional_info)
      rulelist.push(rule)
    }
  }
  return {
    rules : rulelist,
    server_addr: server_addr,
    flags: char_to_flags(flags_char)
  }
}

parseIdentity('11500_114500_32_33a@10.10.10.10')
```

5.5 FreeRADIUS modifications

This section shows some of the modifications done on the FreeRADIUS server. For the Python code and the installation script, see the source code[22].

Overview FreeRADIUS configuration

- `radiusd.conf` configuration loaded by daemon
  - some general configuration
  - INCLUDE `proxy.conf`
  - INCLUDE `client.conf`
  - modules section
    - INCLUDE `mods-enabled/`
  - policy section
    - INCLUDE `policy.d/`
  - INCLUDE `sites-enabled/`

We observe that the main configuration includes 3 files in its main context.

[22] https://github.com/tunroam/auth-server

5.5.1 Pre-authorize

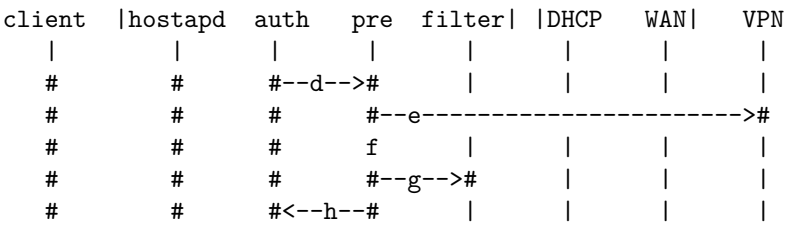

Figure 16: Custom pre-authorize script flow (snippet from fig. 3)

The `rlm_python` module provides a template and an example script. How to call the Python functions, is shown in `experimental.conf`. The mapping to the function names is done in `mods-available/python`. The functions are able to being called through `rlm_attr_filter`.

```
# $ apropos rlm_ # use this to find other modules
$ man rlm_attr_filter|grep -A 18 SECTIONS|grep -v ^$
SECTIONS
       preacct
              Filters Accounting-Request packets.
       accounting
              Filters Accounting-Response packets.
       pre-proxy
              Filters Accounting-Request or Access-Request packets prior to proxying them.
       post-proxy
              Filters Accounting-Response, Access-Accept, Access-Reject, or Access-Challen
       authorize
              Filters Access-Request packets.
       post-auth
              Filters Access-Accept or Access-Reject packets.
```

We implement the protocol in the authentication server by setting it as an `authorize` function. We inserted it before (P)EAP, enabling the script to judge if to Accept or Reject (step `f` in fig. 16).

When running the authentication server in debug mode, we observed that it first goes through the `authorize` section:

```
freeradius -X &
echo "User-Name=bob,Chap-Password=hello" | radclient -x localhost auth testing123

Sent Access-Request Id 77 from 0.0.0.0:32774 to 127.0.0.1:1812 length 44
        User-Name = "bob"
        CHAP-Password = 0xc01e12815a584bfb57210f5435ff843b8a
        Cleartext-Password = "hello"
(0) Received Access-Request Id 77 from 127.0.0.1:32774 to 127.0.0.1:1812 length 44
(0)   User-Name = "bob"
(0)   CHAP-Password = 0xc01e12815a584bfb57210f5435ff843b8a
(0) # Executing section authorize from file /etc/freeradius/3.0/sites-enabled/default
```

```
(0)   authorize {
(0)     policy filter_username {
```

Python3 We observe that the package manager on Raspbian 10 provides FreeRADIUS `v3.0.17`, while Github provides `v3.0.20`, which includes Python3. The discussion on Github suggests it will use a different format and — at the moment of writing — is under active development[23]. We will create our script using Python3, since Python2 is end of life (EOL) in less than a month.

The Python3 script is called from Python2, see the source code[24] for details.

5.5.2 CHAP

For the *Challenge-Handshake Authentication Protocol* to be performed, the server needs to know the plaintext password.

For the authentication we used `rlm_eap_peap` which is a sub module of `rlm_eap`, which will use `rlm_eap_mschapv2`, which in turn requires `rlm_mschap`.

The server validates the password using the `innerid` and the password field configured in the supplicant. FreeRADIUS allows to have a second server (locally or proxy to external for the `validate_certificate` flag) to do the password validation inside the TLS tunnel (used in PEAP and TTLS).

We used FreeRADIUS by running two servers as described in section *Inner-tunnel*.

The inner tunnel needs the plaintext password, which we set in the `authorize` module using a custom Python script:

```
  return RLM_MODULE_OK, (), \
    ( ('Cleartext-Password', 'password'), )
```

5.6 Netfilter

For the network policies we used `nftables` since it is the successor to `firewalld` (Garver 2018) and `iptables` (Debian 2019).

```
apt show nftables 2> /dev/null|grep replaces
 nftables replaces the old popular iptables, ip6tables, arptables and ebtables.
```

We used the old `iptables` syntax for familiarity, using:

```
$ apropos iptables-nft
iptables-nft (8)      - iptables using nftables kernel api
iptables-nft-restore (8) - iptables using nftables kernel api
iptables-nft-save (8) - iptables using nftables kernel api
```

[23] https://github.com/FreeRADIUS/freeradius-server/issues/2351
[24] https://github.com/tunroam/auth-server

References

Amnesty. 2019. "Amnesty International Global Report: Death Sentences and Executions 2018." https://www.amnesty.org/download/Documents/ACT5098702019ENGLISH.PDF.

Athow, Desire. 2019. "The Best Vpn Service 2019." https://www.itproportal.com/guides/best-vpn-service/.

Chamberlain, Kendra. 2019. "Municipal Broadband Is Roadblocked or Outlawed in 26 States." https://broadbandnow.com/report/municipal-broadband-roadblocks/.

Consolvo, Sunny, Jaeyeon Jung, Ben Greenstein, Pauline Powledge, Gabriel Maganis, and Daniel Avrahami. 2010. "The Wi-Fi Privacy Ticker: Improving Awareness & Control of Personal Information Exposure on Wi-Fi." In *Proceedings of the 12th Acm International Conference on Ubiquitous Computing*, 321–30. UbiComp '10. New York, NY, USA: ACM. https://doi.org/10.1145/1864349.1864398.

Debian. 2019. "Nftables." https://wiki.debian.org/nftables.

Eckersley, Peter. 2011. "Why We Need an Open Wireless Movement." https://www.eff.org/deeplinks/2011/04/open-wireless-movement.

Fan, Carl. 2016. "Windows 7 Wired 802.1x Eap-Ttls Authentication Method." *MicrosoftTechnet*. https://docs.microsoft.com/en-us/windows/client-management/advanced-troubleshooting-802-authentication.

Farrokhi, Babak. 2016. "Is Your Isp Hijacking Your Dns Traffic?" https://labs.ripe.net/Members/babak_farrokhi/is-your-isp-hijacking-your-dns-traffic.

Garver, Eric. 2018. "Firewalld: The Future Is Nftables." https://developers.redhat.com/blog/2018/08/10/firewalld-the-future-is-nftables/.

Gewirtz, David. 2019. "The Best Vpn Services for 2019." *Cnet*. https://www.cnet.com/best-vpn-services-directory/.

Gupta, Vishal, and Mukesh Kumar Rohil. 2012. "Enhancing Wi-Fi with Ieee 802.11 U for Mobile Data Offloading." *International Journal of Mobile Network Communications & Telematics (IJMNCT)* 2 (4): 19–29. https://www.researchgate.net/profile/Mukesh_Rohil/publication/268258653_Enhancing_Wi-Fi_with_IEEE_80211u_for_Mobile_Data_Offloading/links/54d1b4a80cf28370d0e0ff56/Enhancing-Wi-Fi-with-IEEE-80211u-for-Mobile-Data-Offloading.pdf.

Gurley, Bill, and Patrick O'Shaughnessy. 2019. "All Things Business and Investing." http://investorfieldguide.com/gurley/#51m12s.

Hainsworth, Paul. 2018. "Open Garden: Building the Internet of Us. Hello World, Part 2." https://medium.com/open-garden-official/open-garden-building-the-internet-of-us-hello-world-part-2-4caf5520f2bc.

Klasnja, Predrag, Sunny Consolvo, Jaeyeon Jung, Benjamin M. Greenstein, Louis LeGrand, Pauline Powledge, and David Wetherall. 2009. "'"When I Am on Wi-Fi, I Am Fearless": Privacy Concerns & Practices in Eeryday

Wi-Fi Use." In *Proceedings of the Sigchi Conference on Human Factors in Computing Systems*, 1993–2002. CHI '09. New York, NY, USA: ACM. https://doi.org/10.1145/1518701.1519004.

Lentink, Sander. 2019. "TMobile Thuis Router in Bridge Mode Krijgen." https://blog.lent.ink/post/tmobile-draytek-bridge-mode/.

"Mac Os X 10.5: How to Configure Network Preferences for 802.1X." 2014. *AppleSupport*. https://support.apple.com/en-us/HT3326.

"Mobile Operating System Market Share Worldwide." 2019. https://gs.statcounter.com/os-market-share/mobile/worldwide.

NCSC. 2019. "Factsheet Dns Monitoring Will Get Harder." https://english.ncsc.nl/publications/factsheets/2019/oktober/2/factsheet-dns-monitoring-will-get-harder.

Ojeda-Zapata, Julio. 2014. "Don't Panic, but That Public Wi-Fi Is Coming from ... Inside Your House." https://www.twincities.com/2014/02/01/dont-panic-but-that-public-wi-fi-is-coming-from-inside-your-house/.

"Open Wi-Fi and Copyright: A Primer for Network Operators." 2014. *Electronic Frontier Foundation*. https://www.eff.org/files/2014/06/03/open-wifi-copyright.pdf.

"PEAP Identity Privacy Support in Windows7." 2009. https://blogs.msdn.microsoft.com/eapteam/2009/01/16/peap-identity-privacy-support-in-windows7/.

Sastry, Nishanth, Jon Crowcroft, and Karen R Sollins. 2007. "Architecting Citywide Ubiquitous Wi-Fi Access." In *HotNets*. https://svn.wirelessleiden.nl/svn/projects/802.1x/ontwikkeling_authenticatiesysteem_2009/diversen/Architecting%20City%20wide%20Ubiquitous%20W--Fi%20Access.pdf.

Schellevis, Joost. 2014. "Telenet Heft Blokkade Poorten Onder 1024 Op." https://tweakers.net/nieuws/99860/telenet-heft-blokkade-poorten-onder-1024-op.html.

Schneier, Bruce. 2008. "My Open Wireless Network." https://www.schneier.com/blog/archives/2008/01/my_open_wireles.html.

"Use Your Homepod on an 802.1X Wi-Fi Network." 2019. *AppleSupport*. https://support.apple.com/en-hk/HT209643.

VPNdiensten. 2019. "Beste Vpn Providers in 2019." https://vpndiensten.nl/informatie/providers/beste-vpn/.

Wierenga, Klaas, and Licia Florio. 2005. "Eduroam: Past, Present and Future." *Computational Methods in Science and Technology* 11 (2): 169–73. http://lib.psnc.pl/Content/593/10.12921_cmst.2005.11.02.169-173_Wierenga.pdf.

"WiFi.EAPClientConfiguration." n.d. *Developer Documentation*. https://developer.apple.com/documentation/devicemanagement/wifi/eapclientconfiguration.

Yu, Haoran, Man Hon Cheung, Lin Gao, and Jianwei Huang. 2017. "Public Wi-Fi Monetization via Advertising." *IEEE/ACM Transactions on Networking*

25 (4): 2110–21. https://arxiv.org/pdf/1609.01951.pdf.