bwCAMPUSNETZ

Zukunftsfähige Konzepte für die Campusnetze
an Universitäten und Hochschulen

Zentrales Captive Portal

Federführung bei der Erstellung dieses Dokuments: Universität Stuttgart
Kontakt: team@bwcampusnetz.de

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Anleitung beschreibt die Konfiguration eines Captive Portals für ein offenes Netzwerk an der Universität Stuttgart. Der gesamte Quellcode sowie weiterführende Informationen zu diesem Projekt sind frei verfügbar auf dem Git Repository der Universität Stuttgart.

Nutzer müssen sich nicht klassisch mit einem Account einloggen, sondern lediglich den Nutzungsbedingungen als eingeladener Gast der Universität Stuttgart zustimmen, um Zugang zum Internet zu erhalten. Das Captive Portal dient hierbei insbesondere als Zugangspunkt für eine campusweite offene ESSID (im Folgenden nur "SSID" genannt), über die Gäste der Universität ins Internet gelangen können. Technisch ist die Nutzung des Portals (siehe Kapitel 2) nicht zwingend auf WLAN beschränkt, da keine Konfiguration auf dem WLAN-Controller in den Authentifizierungs- und Freigabeprozess integriert ist. Die Startseite des Captive Portals sieht dabei folgt aus:

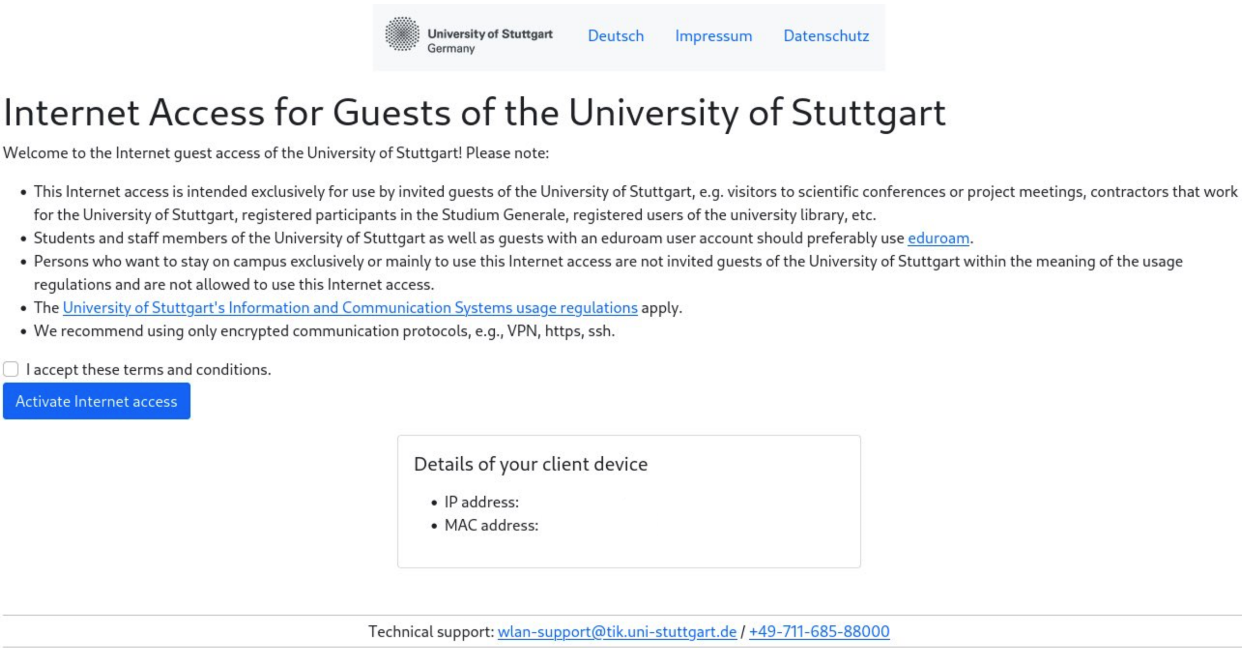

Abb. 1.1: Captive Portal Startseite

Das Captive Portal unterstützt sowohl IPv4 als auch IPv6: Clients erhalten im Produktionssetup eine private IPv4-Adresse (inside local), die per NAT44 auf eine öffentliche IPv4-Adresse (inside global) umgesetzt wird. Die zugewiesene öffentliche IPv6-Adresse (Global

Unicast) wird hingegen ohne NAT ins Internet geroutet und ermöglicht eine direkte Kommunikation mit weltweiten Zielen.

Die Authentifizierung erfolgt über die MAC-Adresse des Endgeräts. Sobald die Nutzungsbedingungen akzeptiert wurden, wird die entsprechende MAC-Adresse, dann im obigen Bild in der Details Box (inklusive inside local IPv4 Adresse) zu sehen, zeitlich befristet freigeschaltet.

Generell kann ein Captive Portal auf verschiedene Weisen implementiert werden. Im Folgenden ist sowohl eine Captive Portal API implementiert. Dabei werden Clients mittels DHCPv4/DHCPv6 über die Existenz eines Captive Portals informiert, siehe auch RFC8910.

Eine weitere gängige, parallel implementierte Methode zur Erkennung eines Captive Portals seitens der Clients, ist der HTTP Redirect. Dabei werden alle HTTP-Anfragen, die ein Client an TCP Port 80 stellt, auf den eigenen Captive-Portal-Server umgeleitet, der mit einer HTTP-Weiterleitung zur eigenen Captive-Portal-Seite antwortet. Moderne Betriebssysteme bzw. Browser testen ihre Verbindung nach dem Verbindungsaufbau, indem sie eine HTTP-Anfrage an eine spezifische URL, am Beispiel von Firefox etwa die URL http://detectportal.firefox.com/canonical.html, senden und eine HTTP 200 OK oder 204 No Content-Antwort erwarten. Falls stattdessen ein Redirect (Status Code 303) zurückgegeben wird, wird die eingeschränkte Internetverbindung erkannt und die Captive Portal Seite wird geöffnet.

Das Setup selbst besteht aus zwei hochverfügbaren (HA) Nodes, die autorisierte MAC-Adressen persistent (d.h. über Neustarts hinaus) speichern und synchronisieren.

Im nächsten Kapitel wird zunächst die Netzwerktopologie des Captive Portals erläutert, einschließlich der relevanten Netzwerkdienste und deren Konfiguration. Anschließend folgt in Kapitel 3 eine detaillierte Beschreibung der internen Abläufe.

In den letzten Jahren wurden mehrere RFCs veröffentlicht, die sich mit der Architektur von Captive Portal Lösungen beschäftigen. Verwiesen wird deshalb auf die RFCs 7710 [1], 8908 [2], 8910 [3] und 8952 [4]. Darüber hinaus fand auf dem BelWü Tech Day 2023 ein Vortrag mit dem Titel “Erfahrungen mit einem “offenen“ WLAN für Gäste“ statt.[5]

2 Vorbereitung

Die folgende Abbildung zeigt die Netzwerktopologie des Captive Portals. Der zentrale Router stellt den Uplink (also die Verbindung in Richtung Internet) bereit und ist über ein Transfernetz mit zwei Linux-Servern (`CaptivePortal1`, `CaptivePortal2`) verbunden, auf denen (Stand 12/2024), die Captive-Portal-Lösung unter Debian 12 (Bookworm) läuft. Zusätzlich ist ein Wireless LAN Controller (WLC) integriert, der zusammen mit den Access Points die offene SSID ausstrahlt. Über diese SSID können sich Clients mit dem Netzwerk verbinden und nach Akzeptieren der Nutzungsbedingungen bzw. Bestätigung der Eigenschaft als eingeladener Gast, siehe Kapitel 1, Internetzugang erhalten. Im Folgenden ist dabei die Netzwerktopologie, bestehend aus dem Uplinkrouter, den Captive Portal Servern sowie dem WLC abgebildet:

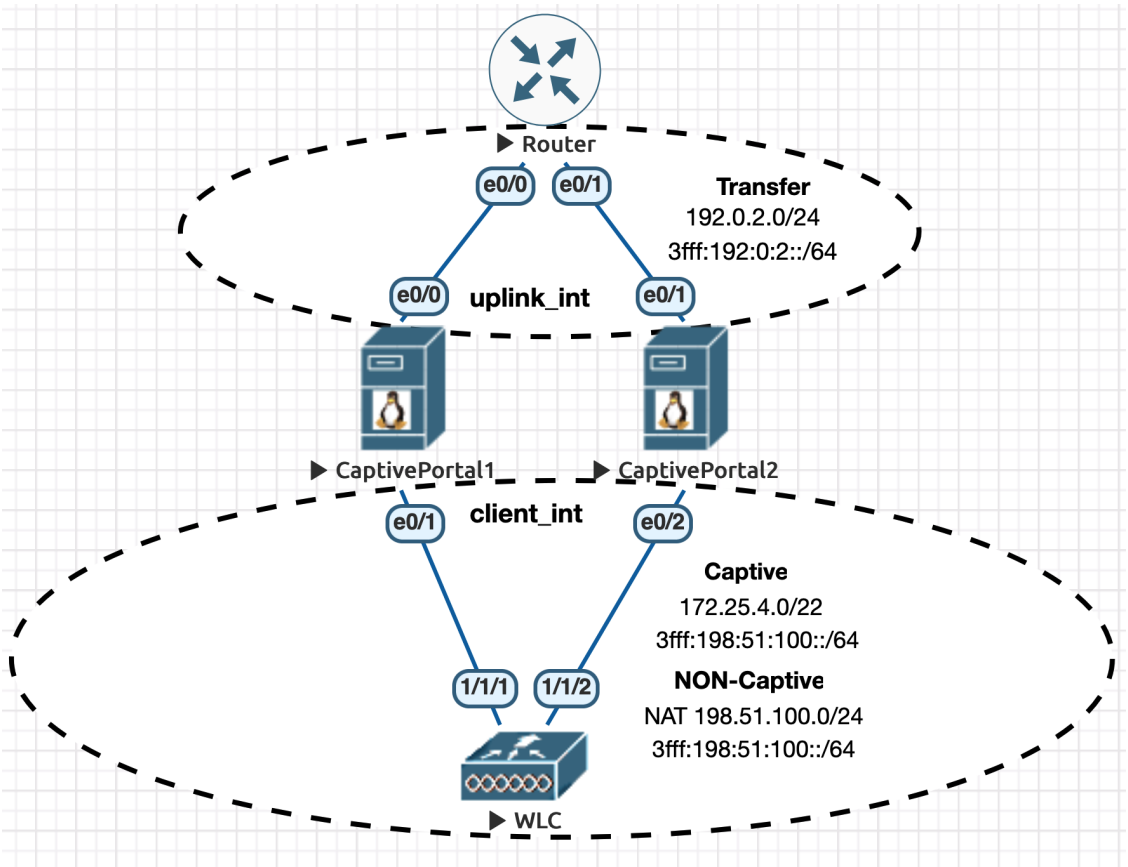

Abb. 2.1: Captive Portal Topologie

Name	v4/v6	Subnetz	Beschreibung
Captive	v4	172.25.4.0/22	Pre-SNAT IPv4, Inside-Local
NON-Captive	v4	198.51.100.0/24	Post-SNAT IPv4, Inside-Global
(NON-)Captive	v6	3fff:198:51:100::/64	IPv6 Clientnetz, kein SNAT
Transfer	v4	192.0.2.0/24	IPv4 Transfernetz
Transfer	v6	3fff:192:0:2::/64	IPv6 Transfernetz

Tab. 2.1: Captive Portal Adressbereiche

Im Weiteren wird IPv4 Source NAT (SNAT) zwischen Captive- und Non-Captive-Subnetzen angewendet. Die Captive-IP-Bereiche entsprechen den Pre-SNAT-Adressen, auch bekannt als Inside-Local-Adressen. Die Post-SNAT, auch bekannt als Inside-Global-Adressen, entsprechen folglich den NON-Captive Adressen. Für das IPv6 Clientnetz findet kein NAT statt. Die Verbindungen werden immer von den wireless Clients aus aufgebaut - das Source in SNAT bezieht sich daher immer auf die wireless Clients. Architekturell wäre hier auch ein NAT64 in Kombination mit PREF64 (das sog. "IPv6-mostly") möglich. Zwischen dem Uplink Router und den Captive Portal Nodes werden die beiden in der Tabelle gelisteten Transfernetze verwendet. Dabei kann je nach Belieben entweder statisch vom Router zu den NON-Captive Netzen geroutet oder alternativ ein Routingprotokoll, wie beispielsweise OSPF, BGP oder IS-IS, eingesetzt werden. Theoretisch ist auch eine direkte Einbindung in eine VXLAN-Fabric möglich.

Zugang zum Netz erhalten die Clients indem sie sich mit der entsprechenden SSID verbinden, wobei dies am Beispiel der Universität Stuttgart die SSID `uni-stuttgart-open` ist. Alternativ lässt sich das Captive Portal auch für den kabelgebundenen Netzzugang verwenden.

In den folgenden Unterkapiteln wird die Konfiguration der Netzwerkdienste erläutert, die auf den jeweiligen Nodes ausgeführt werden. Die Konfigurationsdateien finden sich ebenfalls unter `contrib`. Dabei handelt es sich zumeist um Embedded Ruby Dateien (ERB, `.erb`), die so z.B. in Puppet, einem Konfigurationsmanagement-Tool zur automatisierten Verwaltung von IT-Infrastrukturen, verwendet werden können. Die verwendeten Servicefiles finden sich unter `contrib/systemd`.

2.1 Keepalived

Für die Hochverfügbarkeit (High Availability, HA) der Default-Gateway IP aus Sicht der Clients wird Keepalived eingesetzt, welches eine Implementierung des Virtual Router Redundancy Protocols (VRRP) bereitstellt. Die Konfiguration des Master Nodes (hier im Beispiel das `CaptivePortal1`) sieht für die o.g. Netze entsprechend wie folgt aus:

```
global_defs {
    vrrp_no_swap
    checker_no_swap
    script_user nobody
    enable_script_security
}

vrrp_instance capport_ipv4_default {
    state Master
    interface e0/0 # Replace with actual uplink interface
    virtual_router_id 1
    priority 100

    virtual_ipaddress {
        192.0.2.3 # VRRP Address for Transfer Network
        172.25.4.3 dev e0/1 # VRRP Address for Captive Network
    }

    promote_secondaries
}

vrrp_instance capport_ipv6_default {
    state Master
    interface e0/0          # Replace with actual uplink interface
    virtual_router_id 2
    priority 100

    virtual_ipaddress {
        fe80::1:1 # VRRP Link-Local Address for Transfer Network
        3fff:192:0:2::3 # VRRP Address for Transfer Network
        fe80::1 dev e0/1 # VRRP Link-Local Address for Captive Network
        3fff:198:51:100::3 dev e0/1 # VRRP Address for Captive Network
    }
```

```
    promote_secondaries
}

vrrp_sync_group capport_default {
    group {
        capport_ipv4_default
        capport_ipv6_default
    }
}
```

List. 2.1: Keepalived Konfiguration

2.2 ISC-DHCPD

Die IPv4 Addressvergabe erfolgt ebenfalls durch das Captive Portal. Im Folgenden ist die dhcpd Konfiguration für den `primary` DHCPv4 Server, welcher zeitgleich der VRRP Master (`CaptivePortal1`) ist, abgebildet:

```
option domain-name-servers <name-server>;
option ntp-servers <ntp-server>;

# specify API server URL (RFC8910)
#
# replace "portal.example.com" with your captive portal FQDN
# matching the server certificate
#
option default-url "https://<portal.example.com>/api/captive-portal";

default-lease-time 600;
max-lease-time 3600;

authoritative;

failover peer "dhcp-peer" {
    primary;
    address 172.25.4.1;
    peer address 172.25.4.2;
    max-response-delay 60;
    max-unacked-updates 10;
    load balance max seconds 3;
    split 128;
    mclt 180;
}

subnet 172.25.4.3 netmask 255.255.252.0 {
    option routers 172.25.4.3;
    pool {
        range 172.25.4.100 172.25.7.200;
        failover peer "dhcp-peer";
    }
}
```

List. 2.2: ISC-DHCPD dhcp4 Konfiguration

Der VRRP Backup Node wird entsprechend als `secondary` konfiguriert, wobei dort folglich die `address` und `peer-address` angepasst werden muss. Darüber hinaus wird die in (https://www.rfc-editor.org/rfc/rfc8910.html) definierte Option `default-url` verwendet, um den DHCP-Clients eine URL bereitzustellen, die sie bei der ersten Verbindung mit dem Netzwerk aufrufen sollen. Darüber hinaus wird ebenfalls ein Stateless DHCPv6 Server konfiguriert. Wobei dieser, insofern ein Client diese anfragt, die Captive Portal API per `dhcp6.v6-captive-portal` Option mitliefert:

```
option dhcp6.name-servers <name-server>;
option dhcp6.sntp-servers <sntp-server>;

# specify API server URL (RFC8910)
#
# replace "portal.example.com" with your captive portal FQDN
# matching the server certificate
#
option dhcp6.v6-captive-portal
   ↪ "https://<portal.example.com>/api/captive-portal";

# The delay before information-request refresh
#  (minimum is 10 minutes, maximum one day, default is to not refresh)
#  (set to 6 hours)
option dhcp6.info-refresh-time 3600;

subnet6 3fff:192:0:2::/64{
}

subnet6 3fff:198:51:100::/64{
}
```

List. 2.3: ISC-DHCPD dhcp6 Konfiguration

2.3 radvd

Als Router Advertisement Daemon und damit zur IPv6 Präfix-Vergabe wird `radvd` verwendet. Dieser bietet ebenfalls die Möglichkeit die Captive Portal API URL im Router Advertisement mitzuschicken. Beachtet werden sollte, dass dies erst ab `radvd` Version 2.20 unterstützt wird.

```
interface e0/1
{
    AdvSendAdvert on;
    AdvDefaultPreference high;
    AdvSourceLLAddress off;
    AdvRASrcAddress {
        fe80::1;
    };

    RDNSS 3fff:172:25:80::1 {
    FlushRDNSS off;
    };

    AdvOtherConfigFlag on;

    # requires radvd > 2.19
    #
    # replace "portal.example.com" with your captive portal FQDN
    # matching the server certificate

    AdvCaptivePortalAPI "https://<portal.example.com>/api/captive-portal";

    prefix 3fff:198:51:100::/64
    {
    };
};
```

2.4 nftables

Zentraler Bestandteil des Captive Portals ist die Verwaltung der Firewallregeln und damit der Freigabe des Internetzugangs mittels `nftables`. Ähnlich zu `iptables` basiert `nftables` auf Tables, die verschiedene Chains enthalten, in denen Regeln definiert sind. Hooks bestimmen, an welcher Stelle im Netzwerkstack eine Chain ausgeführt wird, sei es beim Eingang `(prerouting)`, der Verarbeitung `(input, forward)` oder beim Ausgang `(output,postrouting)` des Pakets. Zusätzlich bestimmt die Priorität, in welcher Reihenfolge Chains innerhalb eines Hooks verarbeitet werden - niedrigere Werte werden zuerst ausgeführt, höhere später. Hauptbestandteil sind die im Folgenden erläuterten Hooks `Prerouting`, `Forward` und `Postrouting`.

In den Regelsätzen wird `uplink_int` für Interfaces verwendet, die die Verbindung zum Uplink-Router herstellen. Auf der anderen Seite bezeichnet der Begriff `client_int` die Interfaces, die mit dem WLC (logisch oder physikalisch) verbunden sind.

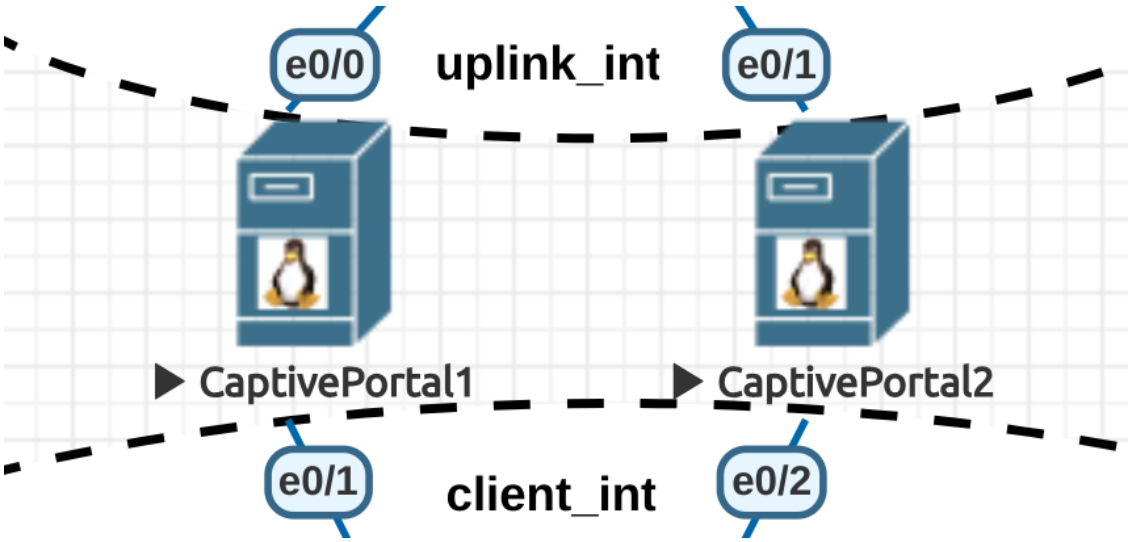

Abb. 2.2: Captive Portal Interface Benamung

Im Folgenden werden alle Tables einschließlich ihrer Chains in der Reihenfolge beschrieben, in der Pakete im Netzwerkstack verarbeitet werden. Die komplette nftables Konfiguration findet sich dabei unter `contrib/nftables.conf.erb`.

2.4.1 Prerouting

Table captive_mark - Priority -150

Die folgende Table `captive_mark` mit der Chain prerouting dient einerseits der initialen Markierung von Paketen, die innerhalb des Captive Portal Setups verarbeitet werden. Ausgangspunkt ist dabei das `allowed` Set, in welches, nach Akzeptieren der Nutzungsbedingungen, die MAC-Adresse des jeweiligen Endgeräts eingetragen wird. Abhängig davon, ob das eingehende Datenpaket aus dem Captive-Netz kommt und/oder der Client das entsprechende Formular akzeptiert hat, wird das Paket entsprechend markiert.

Wichtig für den Neustart des Regelsatzes ist, dass **kein** `flush ruleset` zum Einsatz kommt und daher das `set allowed` (also die Menge der aktuell freigeschalteten MACs/Clients) auch beim wiederholten Laden der nft Regeln erhalten bleibt.

```
table inet captive_mark {
    chain prerouting { }
}
delete chain inet captive_mark prerouting;

table inet captive_mark {
    set allowed {
        type ether_addr
        flags timeout
    }

    chain prerouting {
        type filter hook prerouting priority -150;
        policy accept;
        iifname != <client_int> accept
        # mark packets from client interface with bit 0
        meta mark set meta mark | 0x00000001
        ether saddr @allowed accept
        # mark "captive" clients with bit 1
        meta mark set meta mark | 0x00000002
        accept
    }
}
```

List. 2.4: Table captive_mark - Priority -150

Table captive_mark - Priority -110

Die folgende Table inklusive ihrer Chains legt einerseits fest, welche Adressen sowie Ports aus dem Captive State (d.h. bevor ein Client die Nutzungsbedingungen akzeptiert hat), erlaubt sind. Darüber hinaus wird festgelegt, welche Pakete in den folgenden Hooks abgelehnt oder per Destination NAT (DNAT) auf den lokalen Captive Portal Service umgeleitet werden.

```
table inet captive_filter {}
flush table inet captive_filter
table inet captive_filter {
    # we need the "redirect" decision before DNAT in prerouting:-100
    chain mark_clients {
        type filter hook prerouting priority -110;
        policy accept;
        meta mark & 0x00000001 == 0 accept comment "accept from
            ↪ non-client interface"
        meta mark & 0x00000002 == 0 accept comment "accept packets from
            ↪ non-captive clients"
        # now accept all traffic to destinations allowed in captive
            ↪ state, and mark "redirect" packets:
        jump captive_allowed
    }
    ...
    chain captive_allowed_icmp {
        # allow all pings to servers we allow other kind of traffic
        icmp type echo-request accept
        icmpv6 type echo-request accept
    }
    chain captive_allowed_dns {
        # DNS, DoT, DoH
        udp dport { 53, 853 } accept
        tcp dport { 53, 443, 853 } accept
        goto captive_allowed_icmp
    }
    chain captive_allowed_ntp {
        udp dport 123 accept
        goto captive_allowed_icmp
    }
    chain captive_allowed {
        ip daddr $dns_server_ipv4 jump captive_allowed_dns
        ip6 daddr $dns_server_ipv6 jump captive_allowed_dns
```

```
        ip daddr $ntp_server_ipv4 jump captive_allowed_ntp
        ip6 daddr $ntp_server_ipv6 jump captive_allowed_ntp

        # mark (new) http clients to redirect to local http server with
            ↪ bit 2
        tcp dport 80 ct state new meta mark set meta mark | 0x00000004
            ↪ accept comment "DNAT HTTP"
        # mark packets to reject in forward with bit 3
        meta mark set meta mark | 0x00000008 comment "reject in
            ↪ forwarding"
    }
    ...
}
```

List. 2.5: Table captive_filter - Priority -110

Table nat4/nat6 - Priority -100

In den Tabellen nat4/nat6 werden Pakete mit eingehendem Zielport 80 per Destination NAT (DNAT) umgeschrieben auf Zielport 8080 des Captive Portals. Auf dem Webserver eingehende Pakete auf Port 8080 werden dann mit einer HTTP 303 Redirect Antwort verwiesen auf den regulären https Dienst auf Port 443, siehe Apache Konfiguration.

```
table ip nat4 {}
flush table ip nat4
table ip nat4 {
    chain prerouting {
        type nat hook prerouting priority -100;
        policy accept;
        # needs to be marked from client interface (bit 0), captive (bit
            ↪ 1), and http dnat (bit 2) - otherwise accept
        meta mark & 0x00000007 != 0x00000007 accept
        tcp dport 80 redirect to 8080
    }
    ...
}
...
table ip6 nat6 {}
flush table ip6 nat6
table ip6 nat6 {
    chain prerouting {
        type nat hook prerouting priority -100;
        policy accept;
        # needs to be marked from client interface (bit 0), captive (bit
            ↪ 1), and http dnat (bit 2) - otherwise accept
        meta mark & 0x00000007 != 0x00000007 accept
        tcp dport 80 redirect to 8080
    }
}
```

List. 2.6: Table captive_filter - Priority -110

2.4.2 Forward

Table captive_filter - Priority 0

Die folgende Table, welche mehrere Chains enthält, filtert den Datenverkehr, wobei die `forward` Chain der Ausgangspunkt ist und von dort zu weiteren Chains gesprungen wird. So wird über eine `Whitelist` sowie eine `Blacklist` Chain definiert, dass Pakete zu bestimmten Zieladressen oder Quell-/Zielports aus dem NON-Captive State weitergeleitet oder verworfen werden. Die `antispoof_forward` Chain stellt sicher, dass nur Pakete von autorisierten Quelladressen weitergeleitet werden. Die `forward_down` Chain filtert Datenverkehr vom Uplink-Interface zum Client-Interface und akzeptiert beispielsweise bestehende Verbindungen.

```
table inet captive_filter {}
flush table inet captive_filter
table inet captive_filter {
    ...
    chain forward_reject {
        # close existing tcp sessions when client moves to captive state)
        ct state != new ip protocol tcp reject with tcp reset
        ct state != new ip6 nexthdr tcp reject with tcp reset

        # default icmp reject
        reject with icmpx type admin-prohibited
    }
    chain whitelist {
        ip daddr $whitelist_dest_ipv4 accept
        ip6 daddr $whitelist_dest_ipv6 accept
        tcp dport $whitelist_tcp accept
        udp dport $whitelist_udp accept
        ip protocol esp accept
        ip6 nexthdr esp accept
        icmp type echo-request accept
        icmpv6 type echo-request accept
        # icmp related to existing connections, ...
        ct state established,related accept
    }
    chain blacklist {
        tcp dport $backlist_tcp goto forward_reject
        udp dport $backlist_udp goto forward_reject
        udp sport $backlist_udp_source goto forward_reject
    }
```

```
    chain forward_down {
        # only filter uplink -> client here:
        iifname != <uplink_int> accept
        oifname != <client_int> accept
        # established connections
        ct state established,related accept
        # allow incoming ipv6 ping (ipv4 ping can't work due to NAT)
        icmpv6 type echo-request accept
        drop
    }
    chain antispoof_forward {
        meta mark & 0x00000001 == 0 goto forward_down comment "handle
            ↪ forwardings not from client interface"
        ip saddr { 172.25.4.0/22 } return
        ip6 saddr { 3fff:198:51:100::/64 } return
        drop
    }
    chain forward {
        type filter hook forward priority 0;
        policy drop;
        jump antispoof_forward
        # optimize conntrack timeouts for DNS and NTP
        jump blacklist
        # reject packets marked for rejection in
            ↪ mark_clients/captive_allowed (bit 3)
        meta mark & 0x00000008 != 0 goto forward_reject
        jump whitelist
    }
}
```

List. 2.7: Table captive_filter - Priority 0

2.4.3 Postrouting

Table nat4 - Priority 100

Schließlich werden im Postrouting die am Client-Interface eingehenden IPv4-Pakete per SNAT umgeschrieben.

```
table ip nat4 {}
flush table ip nat4
table ip nat4 {
    ...
    chain postrouting {
        type nat hook postrouting priority 100;
        policy accept;
        # needs to be marked from client interface (bit 0) - otherwise no
            ↪ NAT
        meta mark & 0x00000001 != 0x00000001 accept
        oifname <uplink_int> snat to 198.51.100.0/24
    }
}
```

List. 2.8: Table nat4 - Priority 100

2.5 IPv4/IPv6 Forwarding

Da das Captive Portal als Router fungiert und Netzwerkpakete zwischen verschiedenen Interfaces weitergeleitet werden, muss IPv4- sowie IPv6 Forwarding aktiviert werden.

```
# Enable IPv4 forwarding
sysctl -w net.ipv4.ip_forward=1

# Enable IPv6 forwarding
sysctl -w net.ipv6.conf.all.forwarding=1
```

List. 2.9: IPv4/IPv6 Forwarding

2.6 Apache

Der verwendete Apache Server hört auf den Portnummern 80, 443 und 8080. Der virtuelle Host auf Port 8080 dient als Umleitungsinstanz, die alle Anfragen auf https://portal.example.com weiterleitet. Dabei wird mit `RedirectMatch seeother` ein 303-Redirect verwendet.

Unverschlüsselte Anfragen auf Port 80 werden per Redirect permanent direkt auf Port 443 (https://portal.example.com/) umgeleitet.

Der wichtigste virtuelle Host läuft auf Port 443 (HTTPS) und dient als eigentlicher Captive-Portal-Server. Hier werden SSL/TLS-Zertifikate eingebunden, um eine sichere Verbindung zu gewährleisten. Der Host setzt verschiedene HTTP-Sicherheitsheader wie X-Frame-Options DENY (verhindert Clickjacking), Referrer-Policy same-origin (beschränkt Referrer-Informationen), X-Content-Type-Options nosniff (verhindert MIME-Type-Sniffing) und HSTS (Strict-Transport-Security), um HTTPS dauerhaft zu erzwingen.

Des Weiteren fungiert der Apache Webserver als Reverse Proxy und leitet alle Anfragen, abgesehen von `/static`, weiter an die Backend Anwendung, welche lokal auf Port 8000 läuft.

```
Listen 80
Listen 443
Listen 8080

<VirtualHost *:8080>
    ServerName redirect

    Header always set Cache-Control "no-store"
    # trailing '?' drops request query string:
    RedirectMatch seeother  ^.*$ https://portal.example.com?
    KeepAlive off
</VirtualHost>

<VirtualHost *:80>
    ServerName portal.example.com
    ServerAlias portal-node1.example.com

    Redirect permanent / https://portal.example.com/
</VirtualHost>
```

```
<VirtualHost *:443>
    ServerName portal.example.com
    ServerAlias portal-node1.example.com

    SSLEngine on
    SSLCertificateFile
       ↪ "/etc/ssl/certs/portal.example.com-with-chain.crt"
    SSLCertificateKeyFile   "/etc/ssl/private/portal.example.com.key"

    # The static directory of your theme (or the builtin one)
    Alias /static "/var/lib/python-capport/custom/static"

    Header always set X-Frame-Options DENY
    Header always set Referrer-Policy same-origin
    Header always set X-Content-Type-Options nosniff
    Header always set Strict-Transport-Security "max-age=31556926;"

    RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}

    ProxyRequests Off
    ProxyPreserveHost On
    ProxyPass /static !
    ProxyPass / http://127.0.0.1:8000/
</VirtualHost>
```

List. 2.10: Apache Konfiguration

3 Setup

3.1 Installation

Es gibt drei Möglichkeiten zur Installation:

1. Repository klonen und Abhängigkeiten installieren, Einrichten der virtuellen Umgebung mittels `./setup-venv.sh`
2. Installation als Python Paket
3. Verwendung von `pipx` - Installation in einer separaten virtuellen Umgebung

Vorausgesetzt für das weitere Vorgehen wird die Datei `capport.yaml`, wobei sich diese exemplarisch in `python-capport/capport-example.yaml` befindet.

```
---
comm-secret: mysecret
cookie-secret: mysecret
controllers:
- capport-controller1.example.com
- capport-controller2.example.com
session-timeout: 3600  # in seconds
venue-info-url: 'https://portal.example.com/'
server-names:
- portal.example.com
- some.host.example.domain
api-port: 8000
controller-port: 5000
custom: custom  # path of custom templates and static files
database-file: capport.state
database-file: capport.state
```

List. 3.1: capport.yaml

3.2 Customization

Wenn Templates nach eigenen Wünschen angepasst werden sollen, müssen diese in `custom/templates` gespeichert werden. Selbiges gilt für Templates in weiteren Sprachen, diese werden abgelegt in `i18n/<langcode>`, wobei `<langcode>` der Sprachcode ist (z. B. de für Deutsch, en für Englisch).

Zusätzlich ist die Anwendung so konfiguriert, dass sie den dauerhaften Sprachwechsel über den Query-Parameter `setlang=<langcode>` erlaubt. Der gesetzte Parameter wird in einem Session Cookie gespeichert.

3.3 Controller

Der Controller, welcher standardmäßig auf Port 5000 hört, ist die zentrale Instanz, die für die Verwaltung der Datenbank, siehe hierfür auch `src/capport/database.py`, mit der Liste der erlaubten Clients verantwortlich ist. Er speichert die Datenbank auf der Festplatte und sorgt dafür, dass die Richtlinien zur Client-Zulassung im System durchgesetzt werden, indem er die Daten in den Kernel überträgt. Der Controller ist auch für die Synchronisation der Datenbank zwischen verschiedenen Instanzen zuständig: Wenn zwei Controller sich initial per TCP miteinander verbinden, tauschen sie ihre vollständige Datenbank aus und stellen so sicher, dass alle Instanzen im Netzwerk denselben Stand der Liste der erlaubten Clients haben. Jede Änderung der Datenbankeinträge wird von da an jeweils an den anderen Server weitergegeben und erfolgt über ein Protokoll, das auf anonymous TLS basiert. Das hierbei verwendete Shared Secret findet sich ebenfalls in der Datei `capport.yaml`. Beide Systeme laufen daher ab der initialen Synchronisation immer mit dem vollen und symmetrischen Regelsatz, so dass ein Umschalten des aktiven Knotens zu jeder Zeit ohne Verlust der Berechtigungen aus Sicht der Clients möglich ist. Im vorliegenden Setup werden die Connection-Tracking Zustände nicht synchronisert, da dies überdurchschnittlich stark die Leistungsfähigkeit des Systems negativ beeinträchtigt hatte.

Die Web UI muss die MAC-Adresse der Clients kennen, um diese der Datenbank hinzuzufügen. Das bedeutet, dass die Web UI auf einem Host laufen muss, der mit dem L2-Netzwerk der Clients verbunden ist, um diese aus der Neighbor Table, siehe pyroute2 auslesen zu können. Die Verbindung des Clients zur Web UI muss ebenfalls über diese L2-Verbindung erfolgen.

Gestartet werden die Web-UI mittels `./start-api.sh` und der Controller via `./start-control.sh`. Entsprechende Service Files finden sich ebenfalls unter `contrib/systemd`.

4 Versionsverlauf

Version	Datum	Änderungen
1.0	25.03.2025	Initiale Veröffentlichung

Literatur

[1] W. A. Kumari, Ó. Guðmundsson, P. Ebersman und S. Sheng, *Captive-Portal Identification Using DHCP or Router Advertisements (RAs)*, RFC 7710, Dez. 2015. DOI: 10.17487/RFC7710. Adresse: https://www.rfc-editor.org/info/rfc7710.

[2] T. Pauly und D. Thakore, *Captive Portal API*, RFC 8908, Sep. 2020. DOI: 10.17487/RFC8908. Adresse: https://www.rfc-editor.org/info/rfc8908.

[3] W. A. Kumari und E. Kline, *Captive-Portal Identification in DHCP and Router Advertisements (RAs)*, RFC 8910, Sep. 2020. DOI: 10.17487/RFC8910. Adresse: https://www.rfc-editor.org/info/rfc8910.

[4] K. Larose, D. Dolson und H. Liu, *Captive Portal Architecture*, RFC 8952, Nov. 2020. DOI: 10.17487/RFC8952. Adresse: https://www.rfc-editor.org/info/rfc8952.

[5] S. Kiesel, *Erfahrungen mit einem offenen WLAN für Gäste*, BelWü Tech Day, Stuttgart, Germany, Jan. 2023. Adresse: https://gitlab.bwnetops.de/belwue/tech-day/-/blob/main/2023/slides/Erfahrungen%20mit%20offenem%20WLAN/20230124-offenes-wlan.pdf.