Captive Portal / Hotspot Design

Internal Architecture Spec: Hotspot Profile Deployment System

Objective

To provide clients with a backend tool that automates the configuration and deployment of secure, traceable captive portal environments. The system integrates with CoovaChilli, FreeRADIUS, and optionally Squid, based on client-defined Hotspot Profiles. Aruba Access Points serve as the external SSID broadcasters and are outside our management scope.

System Components

1. Aruba Access Point (External)

- Broadcasts Wi-Fi SSID (e.g., "Guest-WiFi")
- Places connected clients in a specified VLAN (e.g., 182)
- Handles DNS/DHCP unless otherwise specified

2. Hotspot Profile Manager (Core Backend)

- Allows definition of network-facing Hotspot Profiles
- Generates and applies configurations for:
  - CoovaChilli
  - FreeRADIUS
  - Squid (optional)
- Provides secure defaults and constraints to ensure valid setups

3. CoovaChilli

- Captive portal controller
- Manages client redirection, IP routing, session initiation
- Communicates with FreeRADIUS for authentication/authorization
- Optional integration with Squid for HTTP proxying

4. FreeRADIUS

- Authenticates users (voucher, username/password, MAC)
- Stores accounting logs for tracking session data and identity
- Enables full traceability of network access

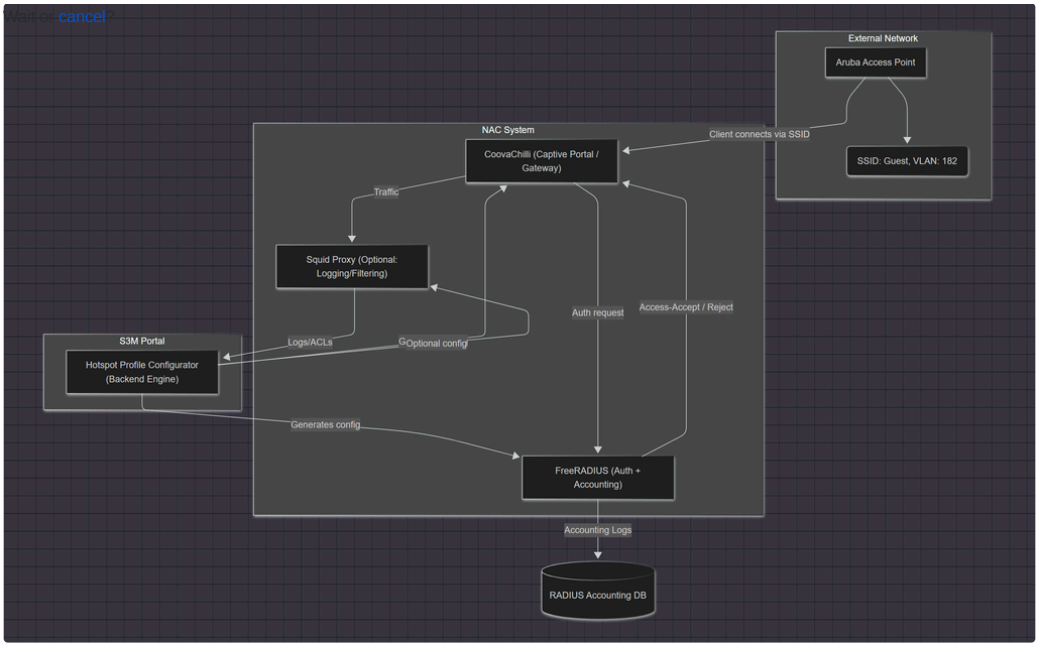
5. Squid (Optional)

- HTTP/HTTPS proxy for traffic logging and access control
- Enforced via transparent iptables NAT rules from CoovaChilli
- Adds an additional layer of monitoring or filtering if required

Architecture Diagram

App is not responding. Wait or cancel?

App is not responding. Wait or cancel?



Data and Traffic Flow

1. **Client connects to Aruba AP**
   - Receives IP via DHCP (from Aruba or firewall)
   - Placed in Guest VLAN (e.g., 182)
2. **CoovaChilli Intercepts Traffic**
   - Redirects HTTP to captive portal (hosted locally or pre-generated)
   - Handles IP/MAC mapping and tracks session
3. **User Authenticates**
   - CoovaChilli sends RADIUS Access-Request
   - FreeRADIUS replies (Accept/Reject)
   - CoovaChilli enables forwarding based on response
4. **Optional: Squid Proxy**
   - HTTP/HTTPS traffic routed to Squid via NAT
   - Squid logs domain/IP and maps to client identity
5. **RADIUS Accounting**
   - Start/Stop packets track session time, data usage
   - Stored in backend DB for traceability and reporting

System Overview

Building a **backend system** that allows clients to define **Hotspot Profiles** through a database. Each profile describes how to configure a local instance of:

- **CoovaChilli**: the captive portal gateway
- **FreeRADIUS**: the authentication/accounting backend

- **Squid** (optional): HTTP/HTTPS logging proxy

This profile-driven design dynamically generates and deploys working configurations **locally on the same server**.

Component Responsibilities

Component	Source of Config	Deployment Target	Notes
CoovaChilli	DB → `generate_coova_config(profile)`	`/etc/chilli/defaults`	Handles portal redirect and DHCP-like behavior
FreeRADIUS	DB → `generate_radius_config(profile)`	`/etc/raddb/clients.conf`, `/etc/raddb/users`, or SQL ⚠️	Handles authentication and session accounting
Squid (optional)	DB → `generate_squid_config(profile)`	`/etc/squid/squid.conf`	Logs and filters post-auth HTTP/HTTPS

Hotspot Profile Database Design (Draft Schema)

```sql
CREATE TABLE hotspot_profiles (
    id SERIAL PRIMARY KEY,
    name TEXT UNIQUE,
    vlan_id INTEGER NOT NULL,
    interface TEXT NOT NULL,
    network_cidr TEXT NOT NULL,          -- e.g., '192.168.182.0/24'
    uam_server TEXT NOT NULL,            -- e.g., 'http://192.168.182.1:3990'
    uam_secret TEXT NOT NULL,
    uam_portal_type TEXT NOT NULL CHECK (uam_portal_type IN ('radius_login', 'voucher')),
    auth_method TEXT NOT NULL CHECK (auth_method IN ('radius', 'voucher')),
    radius_ip TEXT NOT NULL,
    radius_secret TEXT NOT NULL,
    proxy_enabled BOOLEAN DEFAULT FALSE,
    proxy_port INTEGER DEFAULT 3128,
    allowed_domains TEXT[],              -- JSON or array, e.g. ['example.com']
    created_at TIMESTAMP DEFAULT now()
);
```

Note: Secure fields like `radius_secret` and `uam_secret` should be **encrypted** or at least hashed-at-rest if planned to be shown in the frontend.

Field-by-Field Explanation

Field	Why It's Important	Where It's Used	Example Data
`profile_name`	Used to identify the profile	Naming configs, NAS-ID in RADIUS	cafe_latte_guest

`vlan_id`	Matches Aruba's Guest VLAN	Used for tagging/logging only (not needed in config)	182
`interface`	Physical NIC or VLAN interface that CoovaChilli will bind to	`HS_INTERFACE` in Coova	eth0
`network_cidr`	Guest IP pool for clients	`HS_NETWORK`, `HS_NETMASK` in Coova	192.168.182.0/24
`uam_server`	IP/URL of captive portal	Used in CoovaChilli's `HS_UAMSERVER`, redirect logic	http://192.168.182.1:3990
`uam_https`	If there is a valid certificate to be used	`HS_UAMHTTPS` for the captive portal in Coova	on | off
`uam_secret`	Shared secret for UAM portal integrity	`HS_UAMSECRET` in Coova, and used in captive page submission	abcd1234
`auth_method`	Whether FreeRADIUS handles user/pass or vouchers	Chooses logic for FreeRADIUS `users` file or SQL	radius | voucher
`radius_ip`	Where to send RADIUS requests (local, but still explicitly set)	`HS_RADIUS` in CoovaChilli config	10.34.10.5
`radius_secret`	Shared secret between Coova and RADIUS	Used in both CoovaChilli and FreeRADIUS `clients.conf` (Creating Access Device)	testing123
`proxy_enabled`	Whether to activate Squid proxying	Triggers Squid setup and iptables redirect	true | false
`proxy_port`	Port for Squid to listen on	`3128` by default, injected into Squid config	3128
`allowed_domains`	Domains allowed before authentication (e.g., CDN, Google login)	`HS_UAMALLOWED` in CoovaChilli	['api.example.com', 'static.examplecdn.com']

Goal:

When a new hotspot profile is added to the database, the system should:

1. Automatically create and deploy a new **CoovaChilli instance**
2. FreeRADIUS must respond to **Access-Request** and **Accounting-Request** messages from CoovaChilli.
3. **Enhances visibility and control** over user browsing after login with Squid if requested

Workflow Overview

```
1. Admin creates hotspot profile → stored in DB
2. Backend validates input
3. Each component generator reads the DB and generates its config:
     - CoovaChilli: /etc/chilli/defaults
     - FreeRADIUS: /etc/raddb/clients.conf, /etc/raddb/users or DB
     - Squid (optional): /etc/squid/squid.conf
4. System restarts services:
     - systemctl restart chilli
     - systemctl stop freeradius
     - systemctl start freeradius
     - systemctl restart squid (if enabled)
```

Big Picture Workflow (Step-by-Step)

1. A New Hotspot Profile is Created

- A user or admin adds a new hotspot profile via CLI, API, or directly in the DB.
- Example: a profile called `cafe_latte` with interface `eth0.100` (VLAN 100).

2. `ConfigController` Detects a New Profile

- A REST API handler
- The controller receives `profile_id` (or the full profile dict)

3. `ConfigController` Starts Deployment Pipeline

- It calls:
  - `deploy_profile(profile_id)`
- Which internally triggers:
  - `- deploy_coova_config(profile)`
  - `- deploy_radius_config(profile)`
  - `- deploy_squid_config(profile)`

4. CoovaChilli Configuration Phase

`deploy_coova_config(profile)` does the following:

Sub-Step	Description
Parse the profile	Reads info like interface name, UAM server URL, IP range, RADIUS IP
Generate a config file	Uses the profile to fill out a template like `cafe_latte.conf`

Link it for systemd	Symlink this config to `/etc/chilli/cafe_latte.conf`
Register it as a systemd instance	Coova will run as `chilli@cafe_latte.service`
Start the service	`systemctl start chilli@cafe_latte` creates a running captive portal gateway

⚠️ It is crucial that **CoovaChilli *can* run multiple instances simultaneously**, as long as each instance:

- Runs as a **separate process**
- Has its **own config file**
- Binds to a **unique physical or virtual interface**
- Uses a **different UAM listen IP and tun interface**
- Avoids port conflicts

This approach is **used in production environments**, especially for multi-tenant networks, public hotspots, and ISPs.

Verified Conditions for Multiple CoovaChilli Instances

Here's what you must ensure for this to actually work:

Config Element	Must Be
`HS_INTERFACE`	Unique per instance (e.g., `eth0.100`, `eth0.200`)
`HS_NETWORK` / `HS_NETMASK`	Non-overlapping IP pools per instance
`HS_UAMLISTEN`	Unique IP per instance (e.g., `192.168.182.1`, `192.168.183.1`)
`tunX` interface	Automatically created (e.g., `tun0`, `tun1`, etc.)
`iptables rules`	Written separately per instance (ideally per `tunX`)
`systemd`	Use template units: `chilli@profile_name.service`

File-Level Isolation

Each instance should have:

- Its own config: `/etc/chilli/cafe_latte.conf`, `/etc/chilli/techfair.conf`
- Its own log files (optional, if configured)
- Its own IP range, VLAN interface, and redirect portal

Things to Watch Out For

- **UAM ports** can conflict (default `3990`) → but only if using same IP/UAM listener
- You **must manage iptables/NAT rules separately** for each instance's `tunX` device
- Each interface (e.g., `eth0.100`) must be routable to your guests
- Ensure **routing/NAT/firewall rules** are written with interface awareness

5. Recognize CoovaChilli as a valid NAS

- Coova sends requests as a RADIUS client (NAS)
- FreeRADIUS must trust it as a NAS device, such as in the NAS table in the database

6. Handle Authentication Requests

Coova sends:

```
Access-Request
  User-Name = "guest123"
  User-Password = "secret"
  Calling-Station-Id = "MAC_ADDRESS"
  NAS-Identifier = "cafe_latte"
```

FreeRADIUS must proceed to guest services:

- Look up this user
- Validate credentials (username/password or voucher)
- Return an **Access-Accept** (or **Access-Reject**) with **optional attributes**

> Depending on your profile type:

- If it's **voucher-based**: FreeRADIUS should check that the voucher exists, is unused/valid, then mark it as used
- If it's **username/password**: validate against guest user table, LDAP, or flat file

7. Return Proper Reply Attributes

You can control the session using reply attributes:

Attribute	What It Does
`Session-Timeout`	Disconnect the user after X seconds
`Idle-Timeout`	Disconnect the user after X seconds of inactivity
`ChilliSpot-Max-Total-Octets`	Data cap in bytes
`WISPr-Bandwidth-Max-Down` / `Up`	Rate limits in bps

Example Access-Accept:

```
Access-Accept
  Session-Timeout = 1800
  ChilliSpot-Max-Total-Octets = 50000000
```

8. Handle Accounting Packets

Coova will send:

```
Accounting-Start
Accounting-Interim-Update
Accounting-Stop
```

FreeRADIUS must:

- Log these (in SQL or files)
- Track session start/stop time, IP, MAC, data used
- Tie this data back to the profile (e.g., via NAS-Identifier or Calling-Station-Id)

This is **how traffic is traced back to a guest** — these logs are mandatory!

> If you use a SQL-based accounting backend (e.g., `rlm_sql`), check that your schema includes:

- username
- framed_ip
- calling_station_id (MAC)
- acct_session_id
- acct_input_octets / output_octets

9. Configure Squid

In `/etc/squid/squid.conf`:

```
# Listen on default proxy port
http_port 3128 transparent

# Define ACLs
acl allowed_localnet src 192.168.182.0/24  # Change to match Coova subnet
http_access allow allowed_localnet

# Logging format (optional but recommended)
logformat custom_with_ip %ts.%03tu %>a %un %rm %ru %Hs %<st
access_log /var/log/squid/access.log custom_with_ip
```

- `http_port 3128 transparent` enables intercept mode
- `%>a` logs client IP, which is **critical for attribution**
- You can extend logging to include usernames if available

10. Enable NAT Redirection via iptables

After a user logs in and is allowed to surf:

```
iptables -t nat -A PREROUTING -i tun0 -p tcp --dport 80 -j REDIRECT --to-port 3128
```

This command captures all HTTP traffic from clients going through `tun0` (Coova's internal interface) and sends it to Squid.

You can add similar rules for port 443 (HTTPS), but **it requires SSL bumping** — advanced and potentially privacy-sensitive.

11. Access Logs

Squid will log traffic like:

```
1681328475.123    56 192.168.182.101 TCP_MISS/200 1652 GET http://example.com/ - DIRECT/93.184.216.34 text/html
```

You can parse logs to match:

- IP → `radacct.framedipaddress`

- MAC → via CoovaChilli session table
- Username → via RADIUS session or accounting

What Is a Voucher?

A **voucher** is simply a **username** (and optionally a password) that:

- Has a **limited lifetime** (time-based or expiration date)
- Is usually **pre-generated**
- Can be **single-use or multi-use**
- Is stored in a **database or flat file**
- Can return **custom RADIUS reply attributes** (like time or data caps)

Voucher System Architecture (Simplified)

```
[CoovaChilli Portal Page]
     → User enters voucher code
         ↓
[FreeRADIUS]
     → Checks voucher code against database or file
     → If valid:
         → Returns Access-Accept + Session-Timeout + bandwidth limits
         → Marks voucher as used (if one-time)
     → If invalid/expired:
         → Returns Access-Reject
```

Voucher Lifecycle Flow

1. **Create vouchers** (CLI or portal)
   - E.g., generate 100 codes, each valid for 1 hour, expires in 30 days
2. **Store them in a database** (SQL) or file
   - Fields: code, used, created_at, expires_at, session_time, octet_limit
3. **User enters code in captive portal**
   - Form submits username (voucher code) to CoovaChilli
   - CoovaChilli passes it as `User-Name` to RADIUS
4. **FreeRADIUS authenticates**
   - Finds the voucher
   - Verifies it is not used or expired
   - Replies with Access-Accept and session control
5. **FreeRADIUS marks it as used**
   - For one-time vouchers, `used = true` or deletes it
6. **RADIUS accounting tracks usage**
   - Username, IP, MAC, duration, bandwidth

SQL-Based Voucher System

1. **Create a SQL table**

```
CREATE TABLE vouchers (
    code VARCHAR(64) PRIMARY KEY,
    created_at TIMESTAMP DEFAULT now(),
    expires_at TIMESTAMP,
    session_time INTEGER DEFAULT 1800, -- in seconds
    octet_limit BIGINT DEFAULT NULL,  -- optional
    is_used BOOLEAN DEFAULT false
);
```

2. **Enable SQL module in FreeRADIUS**

- Use `rlm_sql` and define queries in `mods-config/sql/authorize-check.sql`

3. **Authorize with SQL**

- Add a policy in `sites-enabled/default → authorize`:

```
authorize {
    ...
    sql
    if (!ok) {
        reject
    }
}
```

4. **Control usage in `authorize`**

```
if ("%{sql:SELECT is_used FROM vouchers WHERE code='%{User-Name}'}" == "t") {
    reject
}
```

5. **Add reply attributes dynamically**

```
update reply {
    Session-Timeout := "%{sql:SELECT session_time FROM vouchers WHERE code='%{User-Name}'}"
}
```

6. **Mark voucher as used (Post-Auth or Accounting-Start)**

- Hook into `post-auth` or `sql` module's `post-auth-query`:

```
UPDATE vouchers SET is_used=true WHERE code='%{User-Name}';
```

Voucher Customization Examples

Field	Purpose
`session_time`	Auto-expire after X seconds
`expires_at`	Prevent login after a certain date
`is_used`	Single-use control
`octet_limit`	Cap on total data usage
`bandwidth_up/down`	Per-user bandwidth limits
`group`	Group vouchers by location or SSID/profile

Health Check

Purpose:

To detect **if a CoovaChilli instance (or system component)** is:

- Misconfigured
- Crashed
- Hung
- Not serving clients properly

Health Check (Per Instance):

Check	How	Why
CoovaChilli process running	`systemctl is-active chilli@profile`	Detects crash or failure to start
tunX interface exists	`ip link show tun0` (or tun1...)	Verifies tunnel is up
UAM server reachable	`curl -k https://portal.example.com`	Detects portal web server failure
Session check	`chilli_query list` has output	Confirms users are able to connect

Sample Concept

```
check_chilli_health() {
  profile=$1
  systemctl is-active chilli@$profile || echo "$profile is down"
  ip link show tun0 >/dev/null || echo "Tunnel not up"
  curl -k --max-time 3 https://portal.example.com/login || echo "Portal unreachable"
}
```

Sample Code Structure

```
hotspot_manager/
│
├── main.py                        # Entrypoint (CLI or REST)
│
├── config_generators/             # Handles config generation for each tool
│   ├── __init__.py
│   ├── coova.py                   # CoovaChilli generator, deploy, delete
│   ├── radius.py                  # FreeRADIUS generator, deploy, delete
│   ├── squid.py                   # Squid generator, deploy, delete
│
├── controllers/                   # Central orchestration logic
│   ├── __init__.py
│   └── config_controller.py       # deploy_profile(), update_profile(), delete_profile()
│
├── templates/                     # Jinja2 templates for config generation
```

```
16 │   ├── coova_defaults.j2
17 │   ├── radius_clients.j2
18 │   ├── squid_conf.j2
19 │
20 ├── utils/                          # Utilities
21 │   ├── __init__.py
22 │   ├── db.py                       # DB functions: get_profile_by_id(), get_all_profiles()
23 │   ├── netutils.py                 # parse_cidr(), validate_ip(), calculate_netmask()
24 │   ├── templates.py                # Jinja2 wrapper: render_template(template_name, context)
25 │
26 ├── migrations/                     # SQL scripts for creating and upgrading the DB schema
27 │   └── create_hotspot_profiles.sql
28 │
29 └── logs/                           # Config deployment logs
30     └── deploy.log
```

Key Modules

`config_generators/coova.py`

- `generate_coova_config(profile)`
- `deploy_coova_config(profile)`
- `delete_coova_config(profile_name)`

`controllers/config_controller.py`

- `deploy_profile(profile_id)`
- `update_profile(profile_id)`
- `delete_profile(profile_id)`

`utils/db.py`

- `get_profile_by_id(profile_id)`
- `insert_profile(data)`
- `update_profile(profile_id, data)`
- `delete_profile(profile_id)`

`utils/netutils.py`

- `parse_cidr('192.168.182.0/24') → ('192.168.182.0', '255.255.255.0')`

Lifecycle Handlers

Action	Trigger	Operation
Add Profile	`POST /profiles` or `create_profile()`	Validate + generate config + deploy
Edit Profile	`PUT /profiles/:id` or `update_profile()`	Regenerate config + restart affected service(s)
Delete Profile	`DELETE /profiles/:id` or `delete_profile()`	Remove config file + restart/clean affected services