AUTHCHA LEARN

Network engineer

802.1X, MAC bypass, device classes, access rules and DHCP: how laptops, phones, printers and IoT get the right network.

5 modules · about 145 minutes · Network engineers

1. 802.1X with EAP-TTLS
2. MAC bypass and device classes
3. Access rules
4. Quotas, network policies and disconnects
5. DHCP and MAC binding

Quizzes, the final assessment and live links are online at learn.authcha.cloud. Labs use the live demo at demo.authcha.cloud.

Module 1 of 5 · 30 minutes

802.1X with EAP-TTLS

By the end you can

- Explain what 802.1X, EAP-TTLS and PAP each do in one sign-in
- Configure a switch port for 802.1X and register the switch in Authcha
- Choose between trusting the server certificate or not, and distribute the CA to clients
- Test a full EAP-TTLS login from a Linux machine

What 802.1X is for

A cable or a Wi-Fi password lets anyone in. **802.1X** makes a device prove who it is *before* the switch port or the Wi-Fi network opens. The device runs a small client called a **supplicant**. The switch or access point (the **authenticator**) passes the conversation to Authcha (the **authentication server**) and only opens the port when Authcha says yes.

Authcha supports two methods for endpoints:

- **EAP-TTLS** with **PAP** inside. The device sets up an encrypted tunnel to the server, then sends the username and password through it. The users are ordinary Authcha users, so you need no separate endpoint directory, and 802.1X users need no mobile number. This module covers this method.
- **EAP-TLS**, where the device proves itself with a certificate issued by Authcha. The Identity and security track covers it.

Two words, one tunnel

EAP-TTLS is the outer method: it builds the encrypted tunnel. **PAP** is the inner method: the plain username-and-password step that happens safely inside it.

Configure a switch port

This is a Cisco IOS-XE example. Other vendors have the same pieces: a RADIUS server entry, an 802.1X authentication list, and per-port settings.

```
aaa new-model
radius server AUTHCHA
 address ipv4 192.0.2.10 auth-port 1812 acct-port 1813
 key <shared-secret>
aaa group server radius AUTHCHA-GRP
 server name AUTHCHA
 ip radius source-interface Vlan10
aaa authentication dot1x default group AUTHCHA-GRP
aaa authorization network default group AUTHCHA-GRP
aaa accounting dot1x default start-stop group AUTHCHA-GRP
dot1x system-auth-control
!
interface GigabitEthernet1/0/10
 switchport mode access
 authentication port-control auto
 authentication order dot1x mab
 authentication priority dot1x mab
 mab
 dot1x pae authenticator
 dot1x timeout tx-period 10
```

Then register the switch under **Devices** and add it as a RADIUS client, exactly as you did in the Foundations track. If you want disconnects (a later module), add the `aaa server radius dynamic-author` block too.

Notice the port allows `dot1x` first and then `mab`. A laptop with a supplicant signs in with 802.1X. A printer with none falls through to MAC bypass. One switch, both kinds.

The server certificate

On first start, each installation creates its **own private certificate authority (CA)** and uses it to issue the RADIUS server's certificate. Nothing is shared between installations and no key ships in the images. The server certificate is valid for two years and renews itself, with the same CA, when fewer than 30 days remain. Clients that trust the CA keep working across renewals.

You then choose how clients treat that certificate.

Path	What it means	Use it
A. Do not validate	The tunnel still encrypts the password, but a rogue access point with a forged certificate could collect credentials	Pilots and closed wired networks only
B. Validate	You give every client the CA certificate (`ca.pem`) and set the expected server name. Supplicants then reject any other server	**Production**

Export the CA for distribution:

```bash
cd /opt/authcha
docker compose cp freeradius:/etc/raddb/certs/authcha/ca.pem ./authcha-radius-ca.pem
openssl x509 -in authcha-radius-ca.pem -noout -subject -dates
```

Push it to clients with Group Policy, a configuration profile or your MDM. **Back up the `raddb_certs` volume with your database.** If the CA is lost, a new one is created and every client needs the new `ca.pem` .

> **Path A in production is a real risk**
>
> If clients do not check the server, anyone can stand up a look-alike Wi-Fi network and harvest passwords. Use Path B everywhere that matters.

Client settings

Every platform needs the same four choices: EAP method **TTLS**, phase 2 (inner) **PAP**, the CA certificate for Path B, and `anonymous` as the anonymous (outer) identity.

- **Windows 10 and 11:** the basic Wi-Fi dialog does not offer TTLS. Deploy a Wi-Fi profile (an XML file) with `netsh wlan add profile` , and import the CA into the machine's trusted roots. For wired, use Group Policy under Wired Network (IEEE 802.3) Policies.
- **macOS and iOS:** install a configuration profile (`.mobileconfig`) by MDM or by opening it on the device.
- **Android:** Settings, Wi-Fi, Add network, security WPA2/WPA3-Enterprise, EAP method TTLS, phase 2 PAP, the CA certificate, the username as identity, and

`anonymous` as the anonymous identity.

- **Linux:** use NetworkManager with `802-1x.eap ttls` and `802-1x.phase2-auth pap`.

Complete profile examples for each are in the Network access guide (https://demo.authcha.cloud/docs/network-access).

Test without a real client

On a Linux machine that is registered as a RADIUS client, `eapol_test` from the `wpa_supplicant` package runs a complete EAP-TTLS login.

```bash
cat > ttls.conf <<'EOF'
network={
    key_mgmt=WPA-EAP
    eap=TTLS
    identity="USERNAME"
    anonymous_identity="anonymous"
    password="PASSWORD"
    phase2="auth=PAP"
    ca_cert="/path/to/authcha-radius.pem"
}
EOF
eapol_test -c ttls.conf -a 192.0.2.10 -p 1812 -s '<shared-secret>'
```

The last line of the output should be `SUCCESS`.

Lab HANDS-ON

Steps 1 to 3 work on the demo (viewer login `demo-viewer`, password `Authcha-View-2026`). The rest need your own install and a Linux machine registered as a RADIUS client.

1	Open	Devices (https://demo.authcha.cloud/devices)	on the demo and find a switch. Note whether	**MAB** is on.

2 Open Live Auth (https://demo.authcha.cloud/activity/live) and look for 802.1X logins. Open one and read which method and which user it used.

3 Open Users (https://demo.authcha.cloud/users). Pick an 802.1X user and notice they have no mobile number requirement.

4 On your install, export the CA with the commands above and read its subject and dates.

5 Write the `ttls.conf` file for a real user and run `eapol_test` against Authcha. Confirm `SUCCESS`.

6 Change the password in the config to a wrong one and run again. Find the failure in **Live Auth** and read the reason.

7 Run it once more with `ca_cert` pointing to a different file. What happens, and why does that matter for Path B?

Cheat sheet KEEP THIS

- **EAP-TTLS** (tunnel) + **PAP** (credentials inside). Users are normal Authcha users.
- Client: method **TTLS**, phase 2 **PAP**, outer identity `anonymous`, CA certificate for Path B.
- **Path B** (validate) for production. **Path A** (do not validate) for pilots only.
- Export the CA: `docker compose cp freeradius:/etc/raddb/certs/authcha/ca.pem .`
- Back up the `raddb_certs` volume.
- Test: `eapol_test -c ttls.conf -a <authcha> -p 1812 -s '<secret>'`, expect `SUCCESS`.
- Read more: Network access (https://demo.authcha.cloud/docs/network-access).

Module 2 of 5 · 30 minutes

MAC bypass and device classes

By the end you can

- Turn on MAB for a switch and add endpoints to the allowlist, one by one and in bulk
- Explain what a device class sends to the switch or access point
- Name the seeded classes and set VLANs before use
- Use DHCP observations to suggest a class without letting Authcha change anything alone

The problem MAB solves

IP phones, printers, cameras, ONTs and badge readers have no supplicant. They cannot answer an 802.1X conversation. **MAC authentication bypass (MAB)** is the workaround: the switch takes the device's MAC address and sends it to Authcha *as the username*. Authcha looks it up in the **MAC Allowlist** and answers.

MAB is only as strong as the MAC

A MAC address can be copied. MAB tells you "a device we know about" and is far weaker than 802.1X. Put MAB devices in a restricted class, and use 802.1X wherever the device can do it.

Turn it on, per switch

MAB is **off by default on every network device**. That is deliberate: a misconfigured device cannot admit endpoints by accident. Open **Devices** and use the **MAB** switch on each access switch that should admit endpoints. **Guided Setup → Onboard IoT devices and printers** walks you through it.

The allowlist

Go to **Endpoints → MAC Allowlist → Add**.

Field	Meaning
MAC address	Any common format (`aa:bb:cc:dd:ee:ff` , `aa-bb-cc-dd-ee-ff` , `aabb.ccdd.eeff` , `aabbccddeeff`). All are stored in one normal form
Device class	The policy the endpoint receives
Description	For example "Lobby phone"
Enabled	Clear it to block the device without deleting it
Bandwidth, quota	Optional overrides of the class defaults for this device

For many devices, import a CSV. Only `mac_address` and `device_class` are required.

```
mac_address,device_class,description,enabled,bandwidth_kbps_in,bandwidth_kbps_out
aa:bb:cc:dd:ee:01,iot,Printer 2nd floor,true,,
aa:bb:cc:dd:ee:02,iot,Lobby camera,true,2048,512
```

The import is checked row by row before anything is written. If one row fails, **nothing** is imported. You can export the allowlist in the same format. Changes take effect within seconds, because Authcha clears its short MAB cache whenever you change a device or class.

When the gateway sends a password too

Some gateways send a password with the MAC (on pfSense it is the *MAC authentication secret*). Set **MAB password mode** on the device: **None** ignores it (the default), **Same as MAC** requires the password to equal the MAC in any notation, and **Fixed shared secret** requires a secret you enter, stored hashed. The **pfSense MAC auth** preset turns MAB on with *Same as MAC*.

Device classes

A **device class** is the policy an endpoint gets when it is accepted. Manage them under **Device Classes → Device Classes**.

Setting	Sent to the switch or access point as
VLAN ID (1 to 4094)	`Tunnel-Type`, `Tunnel-Medium-Type` and `Tunnel-Private-Group-Id`
QoS profile	`Filter-Id`
Max session seconds	`Session-Timeout`
Bandwidth up and down (kbps)	Vendor rate-limit attributes, where defined under Vendor Attributes
Monthly quota	A data cap (see the quotas module)

Every endpoint also receives `Class = <device class name>`. Think of a class as a labelled bundle of network rights, and the allowlist as the list of who gets which bundle.

The seeded classes

Class	Intended use
`default`	Fallback
`guest`	Voucher-admitted guests
`byod`	Employee-owned devices
`iot`	Phones, printers, cameras and other MAB devices
`kiosk`	Shared public terminals
`quarantine`	Where non-compliant devices are placed by posture. **No VLAN is set out of the box**

> **The seeded classes have no VLANs**
>
> Set the VLANs your network uses *before* you add endpoints, and set one on `quarantine` before you rely on posture.

Vendor attributes

With **Advanced** view on, **Device Classes → Vendor Attributes** lets a class send extra vendor-specific attributes, such as Aruba user roles, MikroTik rate limits or Cisco subscriber QoS policies. An attribute goes only to devices of the matching type. Values can use `${kbps_in}` and `${kbps_out}`. Only one attribute per class and vendor can be marked as the rate limit. Presets ship for MikroTik, Aruba, Cambium, Ruckus, Cisco WLC, Cisco IOS and IOS-XE, Arista, Fortinet, Huawei, HPE and F5. Review them against your equipment.

Classification suggestions

You do not have to guess which class a new device belongs in. With **Advanced** view, **Endpoints → Classification Rules** holds simple ordered rules that suggest a class from the device's DHCP fingerprint: vendor class, hostname or parameter-request list. The first matching rule wins. Suggestions appear on **DHCP → Observations** for devices not yet on the allowlist, and accepting one opens the normal **Add** dialog already filled in.

Authcha **never changes a device's class on its own**. A person always confirms. The shipped example rules are disabled until you enable them.

Lab HANDS-ON

On the demo as `demo-viewer` (password `Authcha-View-2026`), unless noted.

1	Open	Endpoints (https://demo.authcha.cloud/nac/endpoints)	and look at the allowlist. Find entries in the	`ap`	and	`ont`	classes next to the PCs and IP phones.
2	Open	Device Classes (https://demo.authcha.cloud/nac/device-classes)	and compare two classes. Which have a VLAN, a bandwidth limit or a quota?				
3	Open	Devices (https://demo.authcha.cloud/devices)	and find a switch with	**MAB**	on, and one without.		

4 Open Live Auth (https://demo.authcha.cloud/activity/live) and find a MAB decision. It shows the matched endpoint and class.

5 On your install, turn on **MAB** for one test switch only.

6 Add one test endpoint to the allowlist in a class with a VLAN you control. Plug it in and watch it land in that VLAN.

7 Import a two-row CSV where one row has a class that does not exist. Confirm nothing was imported.

Cheat sheet KEEP THIS

- **MAB**: the MAC is the username. Off by default. Turn it on per switch under **Devices**.
- Allowlist: **Endpoints** → **MAC Allowlist**. CSV import is all-or-nothing. Changes apply in seconds.
- A **device class** = VLAN, QoS, session timeout, bandwidth, quota, vendor attributes.
- Seeded: `default`, `guest`, `byod`, `iot`, `kiosk`, `quarantine`. **Set the VLANs first.**
- Suggestions come from DHCP fingerprints. They are never applied automatically.
- Read more: Network access (https://demo.authcha.cloud/docs/network-access), Device onboarding (https://demo.authcha.cloud/docs/device-onboarding).

Module 3 of 5 · 30 minutes

Access rules

By the end you can

- Explain when rules run and what they can and cannot change
- Read a rule list top to bottom and predict which rule decides
- Write rules that admit with a class, and rules that reject
- Test a rule before relying on it

Rules run after the sign-in succeeds

Authentication answers "who is this?". **Access rules** answer "given who it is, where it is and what state it is in, what should it get?". Use them to:

- give a Wi-Fi network or a branch its own VLAN
- keep contractors to working hours
- require a posture agent on corporate laptops
- reject combinations that should never be allowed

They apply to **endpoint** access: EAP-TTLS, EAP-TLS and MAB. Logins to network devices with a password and code are controlled by user types, jump servers and privilege mapping, **not** by these rules.

With no rules, or when no rule matches, a session keeps exactly what authentication decided: the MAB class or the certificate's class.

First match wins

Open **Network Access → Access Rules**. Rules are checked **from the top of the list, and the first rule whose conditions all match decides.** Put the most specific rules first and the most general last, using the arrow buttons.

A condition you leave empty matches anything. If you give several values in one condition, **any one** matches (SSID `Corp` or `Corp-5G`).

Condition	Matches
Authentication method	EAP-TTLS, EAP-TLS, MAB or Captive portal (PAP)
User type	The user's type (802.1X only)
User policy	The network policy the user resolves to (not for MAB)
Directory group	The user's group from the last directory sync (802.1X users only)
Device class	The class authentication resolved
Network device group	The group of the switch or controller the request came through
Wi-Fi network (SSID)	The SSID in Called-Station-Id
Posture	Compliant, non-compliant, awaiting first report, grace period, stale, or **no posture agent**
MDM compliance	What Intune or Jamf says: compliant, non-compliant, **not in any MDM**, or unknown
Days and time window	Weekdays and a time range. `22:00-06:00` runs across midnight

A rule does one of two things.

- **Admit with a device class.** The session gets that class's VLAN, QoS filter, timeout and vendor attributes, replacing what it had.
- **Reject.** The session is refused with your message. It is logged, and some devices show the message.

> **Posture always has the last word**
>
> If an enrolled device is not compliant, it is moved to its policy's remediation class **even when a rule admits it**. To refuse such devices instead, add a **Reject** rule on posture *Non-compliant*.

Examples

Rule	When	Then
Guest Wi-Fi	SSID `Guest`	Admit with class `guest`
Contractors after hours	User type `contractor`, outside working hours	Reject: "Outside contractor hours"
Laptops need the agent	EAP-TTLS or EAP-TLS, device group `HQ`, posture *No posture agent*	Admit with class `remediation`
Managed devices only	EAP-TLS, MDM *Non-compliant*, *Not in any MDM* or *Unknown*	Reject: "Device is not compliant"
Engineering VLAN	Directory group `Engineers`	Admit with class `engineering`
Only the network team	Directory group `NetworkAdmins`	Admit with class `mgmt`, then a rule with no conditions: Reject "Not authorised"

Admit one group, refuse everyone else

There is no "not in group" option. Put a rule that admits the group first, and below it a rule with no conditions that rejects. Directory groups also **fail safe**: a user whose groups Authcha does not know, or whose last sync is too old, never matches a rule with a group condition. A directory outage therefore stops group-based access instead of keeping it forever.

Captive-portal logins

Captive-portal PAP logins are evaluated with the method *Captive portal (PAP)*. A VLAN or class means nothing there, so **only Reject rules apply** to them. Use that to refuse, for example, one policy on one gateway group or outside opening hours.

Test before you rely on it

Below the list, **Test a session** runs the same evaluation the RADIUS server uses, at the current time. Give it the method, the identity, the NAS address and, for Wi-Fi, the Called-Station-Id. It shows which rule decides and what Authcha worked out from the

request: user type, device group, SSID, posture, MDM compliance and directory groups. The **Hits** column counts how often each rule has decided a real session.

Rules apply from the **next** authentication. To move sessions that are already connected, disconnect them from **Network Sessions** and they sign in again.

Lab HANDS-ON

1. Open Access Rules (https://demo.authcha.cloud/nac/access-rules) on the demo (viewer login `demo-viewer`, password `Authcha-View-2026`). Read the list from the top. For each rule, say in words what it does.
2. Find the rule list's order. Which rule would decide for a laptop on SSID `Guest`?
3. Use **Test a session** with method EAP-TTLS and a demo username. Which rule decides? Change only the SSID and test again.
4. Look at the **Hits** column. Which rule is used most?
5. On your install, write a **Reject** rule for a user type outside working hours. Test it with **Test a session** at an allowed and a forbidden time. (You can test a time window by changing the rule temporarily.)
6. Reorder two rules so a general rule sits above a specific one. Test again and see how the result changes. Put them back.

Cheat sheet KEEP THIS

- Rules run **after** authentication, for EAP-TTLS, EAP-TLS, MAB and captive-portal PAP (reject only).
- **First match wins.** Most specific first.
- Empty condition matches anything. Several values in one condition: any one matches.

- Actions: **Admit with class** or **Reject**.
- Posture has the last word unless you add a Reject on non-compliant.
- Group rules fail safe when groups are unknown or stale.
- **Test a session** shows the deciding rule. **Hits** shows real use.
- Read more: Access rules (https://demo.authcha.cloud/docs/access-rules).

Module 4 of 5 · 30 minutes

Quotas, network policies and disconnects

By the end you can

- Set a monthly quota on a device class and explain when it is enforced
- Build a network policy with bandwidth, daily and monthly quotas and a slow-down
- Say how a user's policy is chosen
- Set up a disconnect (Change of Authorization) for a device

Two layers of limits

Authcha limits what people and devices use in two places, and it helps to keep them apart.

Layer	Applies to	Set on
Device class	MAB endpoints (a MAC is not a user)	The class, or one allowlist entry
Network policy	802.1X and captive-portal users	A user, a directory group, a user type, or the default

Both depend on **accounting**: the switch or gateway must send usage records back to Authcha, as in the `aaa accounting` line of the switch example earlier.

Data quotas on devices

On a device class, or on one allowlist entry, set a **monthly quota** in bytes (in and out together).

- Authcha adds up accounting data every 5 minutes. Endpoints over their cap are marked **exceeded** and are refused with "Monthly quota exceeded" at their next authentication.
- If the switch has a CoA secret, Authcha also **disconnects the active session straight away**.
- Usage resets at the start of each calendar month.

- Whatever notation the switch uses for the MAC, usage and disconnects treat them as the same endpoint.
- **Device Classes** → **Data Quotas** shows current usage and has a button to run the check now.

Network policies for users

A **network policy** is what an accepted 802.1X or captive-portal user gets beyond admission. Manage them under **Network Access** → **Network Policies**. A policy can set a filter id, session and idle timeouts, an accounting interim interval, upload and download bandwidth, a **daily** and a **monthly** data quota, a warning percentage, a **slow-down** after a volume, and a short list of extra reply attributes.

Which policy a user gets

First match wins:

1. the policy assigned to the **user**
2. the policy of the user's **directory group** (the lowest priority number wins when several match)
3. the policy assigned to the user's **user type**
4. the **default** policy

With no match the user gets no extra attributes. A policy still in use cannot be deleted, so removing one never widens anyone's access.

Quotas, slow-downs and warnings

- A policy can have a daily quota, a monthly quota, or both. When the user signs in, a used-up quota means the login is refused. Otherwise the smaller remainder goes to the gateway.
- A session already open when the user reaches the quota is ended by the console within a minute, and the next login is refused. Set an interim interval so totals stay current.
- **Slow down instead of stop:** after a set number of gigabytes in the day or month, new sessions get lower throttled upload and download rates, and an open session is bounced once to pick them up. The next day or month starts at the plan's normal rates.
- **Warnings:** *Warn at (% of the quota)* records an entry when a user passes that share. Warnings, slow-downs and stops are listed on **Network Policies** → **Usage warnings and limits**. Nobody is told automatically, so feed that list into your monitoring if you want a message.

Extra attributes are limited on purpose

Extra attributes come from a fixed list. Anything that grants privilege or placement (Service-Type, Class, Tunnel attributes, names with privilege, admin, role, group or profile, and EAP or key attributes) is refused, and the database enforces the same rule.

A fixed address for one user

Under **Network Policies → Fixed address and routes for one user**, a user gets `Framed-IP-Address`, `Framed-IP-Netmask` and `Framed-Route` entries in every Access-Accept. It is meant for OpenVPN through pfSense, so a person gets the same tunnel address at each login. One address belongs to one user. Only an administrator sets it, because these attributes decide where someone lands on the network.

Disconnects (Change of Authorization)

Authcha can send RFC 5176 Disconnect-Requests. It does so:

- manually, from **Sessions → Network Sessions**, with the disconnect button on an active row
- automatically, when an endpoint goes over quota
- automatically, when a posture-managed device becomes non-compliant

To allow it for a device:

1. Open the device under **Devices**.
2. Fill in **RFC 5176 CoA-Disconnect** with a CoA secret that is **different from the RADIUS shared secret**. The port defaults to 3799.
3. Configure the device to accept dynamic authorization from Authcha's address with that secret. On Cisco: `aaa server radius dynamic-author`.
4. Allow UDP 3799 from Authcha to the device.

Every request and the device's answer are recorded in **Sessions → Disconnect Log** (Advanced view), with a result: acknowledged, refused, timed out, error, skipped because no CoA secret is set, or skipped because there is no active session.

Lab HANDS-ON

On the demo as `demo-viewer` (password `Authcha-View-2026`), then on your own install.

1. Open Network Policies (https://demo.authcha.cloud/nac/policies) and find policies that slow down after a threshold and policies with a quota.
2. Open Device Classes (https://demo.authcha.cloud/nac/device-classes) and find a class with a monthly quota. Open **Data Quotas** and see current usage.
3. Open Sessions (https://demo.authcha.cloud/activity/sessions) and find an active session. Note who or what it is, where it is and how much data it has used.
4. Open Users (https://demo.authcha.cloud/users), pick a user and find which policy they resolve to: their own, their group's, their type's, or the default.
5. On your install, give a test user a policy with a 1 GB monthly quota and a slow-down at 500 MB. Run the check and read **Usage warnings and limits**.
6. Register a CoA secret on a test device and disconnect an active session from **Network Sessions**. Find the result in **Disconnect Log**.

Cheat sheet KEEP THIS

- Devices: class or entry **monthly quota**, checked every **5 minutes**. Users: **network policy**.
- Policy choice: **user, directory group, user type, default**.
- Quota stop = login refused + live session ended within a minute. Slow-down = lower rates instead.
- Warnings are listed but **not sent**: wire them into monitoring.
- CoA: secret **different** from the RADIUS secret, port **3799**, allow UDP 3799 from Authcha.
- Results: **Sessions → Disconnect Log**.

- Read more: Network access (https://demo.authcha.cloud/docs/network-access).

Module 5 of 5 · 25 minutes

DHCP and MAC binding

By the end you can

- Explain what a MAC binding is and what it needs to be strict
- Choose which DHCP server serves a segment
- Turn on Kea, add a subnet and a reservation, and read leases
- Understand the warning about unexpected DHCP servers

What a MAC binding is

A **MAC binding** says: this device always gets this address, and a device not on the list gets nothing from the reserved range. Authcha stores fixed MAC-to-IP assignments, called **reservations**, and gives them to a **Kea** DHCP server.

The Kea server ships with Authcha but is **off until you turn it on**. That is on purpose: a second DHCP server on a segment is an outage. The console lists current **leases** and **warns about other DHCP servers** answering on the segment.

The pieces

Piece	Where	What it does
Subnets	Network Access → DHCP → Subnets	The segments you serve: range, pool, gateway, DNS, lease time
Reservations	DHCP → Reservations	MAC to IP. Each is written into Kea's host table in the same transaction, so Kea sees the change at its next lookup with no reload
Observations	DHCP → Observations	Devices seen by an optional passive DHCP listener: MAC, fingerprint and a suggested device class. Works where another DHCP server owns the segment
Kea	A container or host, optional	Answers DHCP

Which DHCP server to use

- **One DHCP server per segment.** Two on one broadcast domain hand out conflicting answers. Turn pfSense's DHCP server **off** on the interface where Kea runs, or run Kea on a segment pfSense does not serve.
- **Binding on the firewall's own segments:** keep pfSense DHCP and add **static mappings** (Services, DHCP Server, the interface, DHCP Static Mappings). Authcha does not manage those.
- **A dedicated segment:** put Kea on a server with a leg in that VLAN, or use a DHCP relay on the router pointing at Kea. This is what the console's reservations are for.

Run Kea

1. Make sure no other DHCP server serves the segment.
2. On the Authcha host: `sudo authcha dhcp enable --interface eth1`, where `eth1` faces the segment. It creates an API token for Kea, starts the container with host networking, and Kea creates its tables on first start.
3. In **DHCP → Subnets**, add the subnet with its pool and gateway. Then add reservations for devices that must keep an address.
4. After adding or changing a **subnet**, run `sudo authcha dhcp restart`. Reservations need no restart.
5. `authcha dhcp status` tells you if it runs. `authcha dhcp disable` stops and removes it; reservations and leases stay in the database.

> **Host networking matters**
>
> Kea must run with host networking, or it never sees broadcast DHCP. On a host with several network legs, name the one that faces the segment.

Leases and rogue-server warnings

DHCP → Leases lists what Kea has handed out: address, MAC, name, subnet, time left, and whether a reservation exists for the MAC. It refreshes every ten seconds, and leases that expired more than an hour ago are hidden.

Every five minutes the Kea container sends a DHCP discover on its interface and reports each server that answers. **DHCP → Servers** shows this server's own addresses as **ours**. Any other server seen in the last 24 hours is **unexpected**, and a red banner says so. A home router plugged into the LAN is the usual cause. If the server belongs there (a relay, a second Kea), choose **Mark expected**.

This check sees only what answers a discover at that moment, on that one interface. It does **not** replace DHCP snooping or port security on your switches.

Making the binding strict

Reservations alone give listed devices their address, but unlisted devices still get one from the pool. To serve **only** listed devices, the Kea subnet must have no pool. Kea allows that, but the console's subnet form requires a pool, so today you edit the rendered `kea-dhcp4.conf` to remove that subnet's `pools` entry, and redo it after each subnet change. Pair it with 802.1X or MAB on the switch so an unlisted device is not on the network at all.

Test it

On a client in the segment:

```
sudo dhclient -v -r eth0; sudo dhclient -v eth0      # a reserved MAC gets exactly i
sudo nmap --script broadcast-dhcp-discover -e eth0   # lists every DHCP server answe
```

Check three things: the reserved device got its reserved address, an unlisted device got nothing (no pool) or a pool address (pool), and `nmap` shows one server.

Lab HANDS-ON

Steps 1 to 3 on the demo (viewer login `demo-viewer`, password `Authcha-View-2026`). The rest need a lab network.

1	Open	DHCP (https://demo.authcha.cloud/dhcp)	on the demo. Look at the subnets and the reservations. Find one MAC with a fixed address.
2	Open the	**Observations**	tab and look at a device's fingerprint and suggested class.
3	Open the	**Servers**	tab and see how ours and unexpected servers are shown.
4	On a lab host:	`sudo authcha dhcp enable --interface <interface>`	for a segment with no other DHCP server.

5 Add a subnet and one reservation for a test machine. Renew its lease and confirm it gets exactly the reserved address.

6 Plug a home router into the segment. Within about five minutes find it under **Servers** as unexpected, then mark it expected or remove it.

Cheat sheet KEEP THIS

- **Reservation** = MAC to IP, written into Kea's host table, no reload.
- Enable: `sudo authcha dhcp enable --interface <interface>`. After a subnet change: `authcha dhcp restart`.
- Status: `authcha dhcp status`. Stop: `authcha dhcp disable`.
- **One DHCP server per segment.** Host networking for Kea.
- Strict binding = a subnet with **no pool** + 802.1X or MAB.
- Rogue check: every 5 minutes, listed in **DHCP → Servers**.
- Read more: DHCP with MAC binding (https://demo.authcha.cloud/docs/dhcp-mac-binding).