AUTHCHA LEARN

Operations

Backups, upgrades, the encryption key, monitoring, the audit trail and how to troubleshoot a failed sign-in.

6 modules · about 155 minutes · Administrators and on-call engineers

1. Daily health, services and logs
2. Backups, restores and the restore drill
3. Upgrades, rollback and what an upgrade keeps
4. The encryption key and rotating secrets
5. Monitoring, metrics and the audit trail
6. Troubleshooting a failed sign-in

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 6 · 20 minutes

Daily health, services and logs

By the end you can

- Run the quick health checks from the console and the shell
- Restart one service, and know when a restart is not enough
- Read the right log for a RADIUS, console or code-delivery problem
- Turn debug mode on safely, and off again

A one-minute health check

Do this every morning, or wire it into monitoring. From the Authcha host:

```bash
cd /opt/authcha
docker compose ps                                  # every service running? db, redi
curl -sk https://127.0.0.1/api/health              # {"status":"ok","db_ok":true}
docker compose logs --tail 50 freeradius | grep -i "ready to process"
df -h /var/lib/docker /opt/authcha                 # disk space
```

In the console, **System** shows each container's state and health, and the backups. The **Dashboard** summarizes authentication activity and system state.

The health endpoint needs no login, which makes it good for load balancers and monitors. Alert when the request fails, when `status` is not `ok`, or when `db_ok` is `false`. Two lighter probes exist: `/api/healthz` (the process is up) for supervisors, and `/api/readyz` (the database answers within 2 seconds) for load balancers.

Managing services

```bash
docker compose ps                                  # state of every service
docker compose logs -f freeradius                  # follow one service's log
docker compose restart freeradius                  # restart one service
docker compose up -d                               # start anything stopped; apply compo
docker compose up -d --force-recreate admin        # re-read .env for one service
```

The service names are `freeradius`, `db`, `redis`, `otp_service`, `admin`, `ops` and `caddy`.

> **Restart does not re-read .env**
>
> `docker compose restart` does **not** read `.env` or `otp.env` again. After you edit either file by hand, **recreate** the affected service with `docker compose up -d --force-recreate <service>`. This is the most common reason "I changed the setting and nothing happened".

Which log, for which problem

Symptom	Look here
A login was accepted or rejected, and why	**Live Auth** → **Logs**, or `docker compose logs -f freeradius`
The console or an API call misbehaves	`docker compose logs -f admin`
One-time codes are not arriving	`docker compose logs otp_service` and **Audit Trail** → **OTP Funnel**
Everything, the last hour	`docker compose logs --since 1h`

In the console, viewers can see the RADIUS and OTP logs. The database, cache and console logs are for admins only. Passwords are never logged.

Debug mode, with a guard

For a hard problem, you can make RADIUS log every request in full. It is protected so it cannot be left on by accident. Both lines are needed in `.env`:

```
FREERADIUS_DEBUG=true
FREERADIUS_DEBUG_CONFIRM=log-secrets
```

Then `docker compose up -d freeradius`.

Debug output contains secrets

It includes request attributes, one-time codes and internal connection details. The console's log viewer masks them, but `docker compose logs` on the host does not. Turn it off as soon as you have what you need: set `FREERADIUS_DEBUG=false` and run `docker compose up -d freeradius`.

Stop logs filling the disk

Container logs use Docker's logging driver. Set rotation in `/etc/docker/daemon.json`, then restart Docker:

```JSON
{ "log-driver": "json-file", "log-opts": { "max-size": "50m", "max-file": "5" } }
```

Lab HANDS-ON

Steps 1 to 3 on the demo as `demo-viewer` (password `Authcha-View-2026`). The rest on your install.

1. Open System (https://demo.authcha.cloud/system) on the demo. Check every service shows healthy.
2. Open Live Auth (https://demo.authcha.cloud/activity/live) and filter the log by result. Find a rejection and read its reason.
3. Open the Dashboard (https://demo.authcha.cloud/) and find the authentication summary.
4. On your install, run the four quick checks above and note the results.
5. Edit a harmless value in `.env`. First run `docker compose restart admin`, then check if the console saw the change (it will not). Then run `docker compose up -d --force-recreate admin` and check again.

6 Turn debug mode on, send one login, find the full request in the log, then turn it off again.

Cheat sheet KEEP THIS

- Health: `docker compose ps`, `curl -sk https://127.0.0.1/api/health`, check disk space.
- Service names: `freeradius`, `db`, `redis`, `otp_service`, `admin`, `ops`, `caddy`.
- Edited `.env`? **Recreate**, not restart: `docker compose up -d --force-recreate <service>`.
- Debug: `FREERADIUS_DEBUG=true` **and** `FREERADIUS_DEBUG_CONFIRM=log-secrets`. Turn it off after.
- Rotate logs in `/etc/docker/daemon.json`.
- Read more: Operations (https://demo.authcha.cloud/docs/operations).

Module 2 of 6 · 30 minutes

Backups, restores and the restore drill

By the end you can

- List the scheduled jobs and what each one protects
- Name what is not in a backup and must be saved separately
- Restore a backup onto the same host
- Run and read the restore drill

Two kinds of backup

Authcha takes two independent database backups. Both contain everything in PostgreSQL: users, devices, policies, accounting, audit log and license.

Scheduled backups on the host

The installer sets up these jobs in `/etc/cron.d/authcha`.

Job	When	What it does
Backup	Every 6 hours	Writes a compressed dump, `radius-<timestamp>.dump`, to `/opt/authcha/backups/` with a `.sha256` file. Next to it: the RADIUS certificate volume (including the server key), the RADIUS clients and saved site policies, and a small text file with the version and a short fingerprint of the encryption key. Files older than 14 days are removed. `.env` **is not included.**
Restore drill	Sundays 02:00	Restores the latest dump into a temporary database, checks the key tables are filled, checks the archives, and proves that the key in `.env` can decrypt every encrypted secret in the restored copy. Then it deletes the temporary database
Audit retention	Daily 03:00	Archives security logs older than 180 days, then removes them from the database

Run them yourself whenever you like:

```bash
sudo authcha backup                    # back up now
sudo authcha backup verify             # drill the newest dump
```

Console backups

The console also writes a **daily** compressed SQL backup to its own storage and keeps it for 30 days by default (`BACKUP_RETENTION_DAYS`; `0` turns it off). List and download them under **System → Automated backups**. **Backup database** on the same page downloads a fresh one at any moment. Do this before every upgrade.

What you must back up yourself

What	Why
The `backups/` directory	The scheduled backups live on the same host. Copy them off it. Set `BACKUP_OFFSITE_PATH=backup@host:/srv/authcha-backups/` and every dump is synced there with `rsync` over SSH
`.env` and `otp.env`	They hold every bootstrap secret and are **not** in the database backup. Store them encrypted. `PKI_KEY_ENCRYPTION_KEY` **matters most**: without it, the certificate authority keys, identity-provider signing keys and MDM secrets in every backup cannot be read
Your own TLS certificate and key	In `/opt/authcha/config/tls/`, if you supplied them
Audit archives	In `backups/audit-archive/`, if regulations require logs beyond the retention window

> **A backup stored with its key defeats the encryption**
>
> Keep the encryption key **separate from the backups**, in at least two offline places. The next-but-one module covers it.

Restore

> **A restore replaces the whole database**
>
> Everything changed after the backup was taken is lost. Do this deliberately.

Stop the services that use the database and make an empty one:

```bash
cd /opt/authcha
docker compose stop admin freeradius
docker compose exec db dropdb -U radius radius
docker compose exec db createdb -U radius radius
```

Load a scheduled backup, after checking its checksum:

```bash
sha256sum -c backups/radius-20260901T000000Z.dump.sha256
docker compose exec -T db pg_restore -U radius -d radius --no-owner --no-acl --exit-
  < backups/radius-20260901T000000Z.dump
```

A console backup (`.sql.gz`) loads with `gunzip -c file | docker compose exec -T db psql -U radius -d radius -q -v ON_ERROR_STOP=1` .

Then put back the RADIUS certificates from the matching archive, put the **same** `PKI_KEY_ENCRYPTION_KEY` back in `.env` , recreate every container (do not just restart) and check the key:

```bash
docker compose up -d --force-recreate
sudo authcha kek-check
```

`kek-check` prints how many items of each kind it decrypted. If it fails, the key in `.env` is not the one the backup was made with.

The restore drill

A backup is only worth what a restore proves. Run the drill after installing, after every upgrade and at least once a quarter.

1. `sudo authcha backup` , and wait.
2. `sudo authcha backup verify` . It must end with `drill PASSED` .

3. At least once, restore for real onto a spare host with the escrowed key, run `sudo authcha kek-check`, sign in and authenticate one 802.1X client. Write down the date and the result.

The weekly job does step 2. If it warns `KEK not verified`, the admin image or `.env` was not available to it. Fix that, because a drill that skips the key check can pass on a backup you could not actually use. Check its log now and then:

```bash
tail -n 20 /var/log/authcha-restore-drill.log
```

Lab HANDS-ON

Needs your own install (or a test VM).

1. Run `sudo authcha backup`. List `/opt/authcha/backups/` and find the dump, its `.sha256` and the companion files.
2. Open **System → Automated backups** in the console and download the newest one.
3. Run `sudo authcha backup verify`. Confirm `drill PASSED`.
4. Check the weekly drill's log with `tail -n 20 /var/log/authcha-restore-drill.log`.
5. Set `BACKUP_OFFSITE_PATH` to another host you control, recreate the `admin` service, run a backup and confirm the files arrive there.
6. On a **spare** host with the same Authcha version, do a full restore with the escrowed key. Run `sudo authcha kek-check`, sign in and write down the date.

Cheat sheet KEEP THIS

- Scheduled: backup **every 6 h**, drill **Sundays 02:00**, audit retention **daily 03:00**. Kept **14 days** on the host.
- Console backup: daily, **30 days**, under **System → Automated backups**.
- Now: `sudo authcha backup`, `sudo authcha backup verify`.
- Not in the backup: `.env`, `otp.env`, your TLS key. Escrow the encryption key separately.
- Restore: stop `admin freeradius`, drop and create the database, `pg_restore`, restore certs, **same key**, recreate everything, `sudo authcha kek-check`.
- Drill after install, after each upgrade, and at least quarterly. Do one real restore.
- Read more: Operations (https://demo.authcha.cloud/docs/operations).

Module 3 of 6 · 25 minutes

Upgrades, rollback and what an upgrade keeps

By the end you can

- Check for a new version and read what changed
- Upgrade online or air-gapped and know what the upgrade does for you
- Roll back, and say what is lost when you do
- Find and fix the weak defaults an upgraded install keeps

Knowing a new version is out

Settings → **Updates** shows the installed version, whether a newer one exists, and the release notes: what is new since your version and the recent changes in yours. When a newer version exists, administrators see a banner at the top of every page, which can be dismissed per version.

The check reads a small list of versions and notes, **signed with the Authcha release key**, from the release site every six hours (or **Check now**). Only the version number is sent, in the User-Agent: no license, host or user information. Nothing is downloaded or installed by the check, and a list whose signature does not verify is ignored. Turn it off in the same tab, or with `UPDATE_CHECK=off` in `.env` for a site with no internet.

Upgrade

```bash
sudo authcha upgrade                    # the latest release
sudo authcha upgrade --version 1.11.0   # a specific release
sudo authcha --version                  # product version, bundle, image tags
sudo authcha status --json              # services, migrations, license, legacy sett
```

`authcha upgrade` does these steps for you:

1. Downloads the release with your installed license (upgrades are included while the license is valid).

2. Verifies its SHA-256 and Ed25519 signature, and checks Docker has about 3 GB free for the images. If it does not, it removes the images of older releases first, and stops with a clear message if that is still not enough.
3. **Backs up the database** to `/opt/authcha/backups/pre-upgrade-<from>-to-<to>-<time>.sql.gz`.
4. Loads the new images and replaces the product files, keeping your settings, certificates and data.
5. Restarts and waits for the console, then removes the images of releases older than the one you came from. The running release and the previous one stay, so `authcha rollback` works.

If the console is not healthy within five minutes, it rolls back on its own. Database migrations run automatically at start.

Air-gapped: copy the new bundle, its `.sig` and `install.sh` to the server and run `sudo bash install.sh --bundle authcha-X.Y.Z.tar.gz --license-file license.json`. The same backup, checks and data-keeping apply.

> **Make it a habit**
>
> Before every upgrade: take a console backup (**System** → **Backup database**), then upgrade, then run `sudo authcha backup verify` afterwards.

Roll back

```bash
sudo authcha rollback
```

It reinstalls the previous release, which is kept on the host, and restores the database from the pre-upgrade backup. **Changes made since the upgrade are lost**, so it asks first. The installer refuses to install an older release over a newer one any other way.

> **Roll back the database too**
>
> Never roll the software back without restoring the pre-upgrade database. After a key rotation, the restored database is encrypted under the **previous** key, so put that key back in `.env` and recreate the containers, or certificate issuance fails.

What an upgrade keeps, and the weak defaults

An upgrade never changes your settings. An install that began on an older release therefore keeps a few defaults a **new** install does not have. The upgrade writes each into `.env` explicitly, prints the list when it finishes, and Authcha keeps reminding you about the weak ones:

- `sudo authcha status` names each one.
- The console shows an **Action needed after the upgrade** banner and a Dashboard card until each *risk* is fixed or accepted with a reason. An accepted risk shows again after 30 days.
- `/api/health/summary` and the metric `authcha_legacy_settings{severity}` carry the same.

Examples: `RADIUS_REQUIRE_MESSAGE_AUTHENTICATOR` (a **risk**: without it a PAP request with no Message-Authenticator is accepted, the BlastRADIUS issue), the password policy for RADIUS users (a **risk**), the captive-portal rule that stops a code stage being skipped (a **risk**), and the PAP cache (a capacity recommendation).

To apply the recommended values:

```bash
sudo authcha apply-recommended
```

It shows what it will change, backs up `.env`, edits it, recreates FreeRADIUS and the console (a few seconds without RADIUS), and checks that FreeRADIUS now runs with the new value. Before you apply the Message-Authenticator setting, make sure every network device sends Message-Authenticator with PAP. A device that cannot should stay on `auto`: accept the risk in the console with a reason.

Lab HANDS-ON

1. Open Settings (https://demo.authcha.cloud/settings) on the demo as `demo-viewer` (password `Authcha-View-2026`) and find **Updates**. Read the current version and the notes.

2	On your install, run	`sudo authcha --version`	a n d	`sudo authcha status`	. Does it list any legacy settings?	
3	Open the Dashboard and look for an	**Action needed after the upgrade**			card .	
4	Take a console backup, then run	`sudo authcha upgrade`	to the latest (or	`--versi on`	for a lab version).	
5	Ru n	`sudo authcha apply-recommended`	and read what it proposes before you say yes.			
6	On a lab host, run	`sudo authcha rollback`	and read what it warns you about. Cancel if you do not want to roll back.			

Cheat sheet KEEP THIS

- **Settings** → **Updates**: version, notes, signed check every 6 h.
- `sudo authcha upgrade [--version X]` . It backs up, verifies, keeps settings, rolls itself back if unhealthy in 5 min.
- `sudo authcha rollback` : previous release + pre-upgrade database. **Loses changes since.**
- Air-gapped: `install.sh --bundle ... --license-file ...` .
- Weak defaults after an upgrade: `sudo authcha status` , then `sudo authcha apply-recommended` .
- Before every upgrade: console backup. After: `sudo authcha backup verify` .
- Read more: Operations (https://demo.authcha.cloud/docs/operations).

Module 4 of 6 · 25 minutes

The encryption key and rotating secrets

By the end you can

- Say what PKI_KEY_ENCRYPTION_KEY protects and what happens if it is lost
- Escrow the key properly and check it with kek-check
- Rotate the key with kek-rotate and clean up afterwards
- Rotate an OTP service token without downtime

One key, many secrets

`PKI_KEY_ENCRYPTION_KEY` in `.env` is 64 hexadecimal characters, made by the installer. It encrypts every long-lived secret Authcha holds in the database, with AES-256-GCM, each one tied to its own row so a copied value does not decrypt elsewhere.

What it protects
Certificate authority private keys (802.1X CA and RadSec CA)
The SCEP registration-authority key
Identity-provider token-signing keys
MDM (Intune, Jamf) client secrets
MikroTik gateway REST passwords
Enforcement connector secrets (firewall API keys, webhook signing secrets)

The good news: a database dump or backup alone cannot issue certificates or sign tokens. The bad news is the flip side: it is a **single point of loss**. If the key is lost, every one of those secrets is unrecoverable, the certificate authorities included, and every certificate you issued would have to be replaced.

Escrow it

Keep **at least two copies**, offline, in different places: a password-manager vault with restricted access, a sealed envelope in a safe. Keep them **separate from the backups**. The key is never part of a backup on purpose, because a backup and its key stored together defeat the encryption. Record which backups were made with which key: the small `radius-meta-<timestamp>.txt` next to each dump holds a short fingerprint of the key it used.

If the key is **not set**, the console still works, but certificate issuance is unavailable and identity-provider signing keys are stored **unencrypted** (it warns at every start, and the metric `authcha_oidc_plaintext_keys` shows it). Set the key and restart: existing keys are encrypted in place.

Check it

```bash
sudo authcha kek-check
```

It decrypts every sealed item with the key in `.env` and prints the counts. It changes nothing, and it is part of the restore drill.

Rotate it

Rotate on a schedule you set, when someone who knew the key leaves, and **at once** if the key, or a host backup together with `.env`, may have been exposed. Rotation re-encrypts every item under a new key. It does **not** re-issue certificates or change the certificate authority.

```bash
sudo authcha kek-rotate --dry-run      # decrypt everything with the current key, ve
sudo authcha kek-rotate                # do it
```

In order, the command: does a dry run; takes a fresh backup; writes the new key to a `.pending` file so it cannot be lost if the host fails; stops the console; re-encrypts every item in **one database transaction**, checking each against the original before it may commit (any failure rolls everything back and restarts on the old key); replaces the key in `.env`; starts the console; and runs `kek-check`. Expect about a minute without the console. **RADIUS authentication is not interrupted.**

To use a key you made yourself, export it first: `sudo NEW_PKI_KEY_ENCRYPTION_KEY=<64 hex> authcha kek-rotate`.

After a successful rotation:

1. Put the **new** key into escrow now.
2. **Backups taken before the rotation need the old key.** The command leaves the old `.env` at `backups/env-before-kek-rotation-<time>`. Move the old key into escrow, labelled with the dates it covers, and delete that file. Keep it as long as you keep those backups.
3. Run `sudo authcha backup verify`.

Rotating the other secrets

For each change to `.env` or `otp.env`, **recreate** the listed services so they read the new value.

Secret	Procedure	Recreate	Side effects
`ADMIN_SESSION_SECRET`	Replace with a long random value	`admin`	Everyone is signed out
`PROXY_TRUST_SECRET`	`openssl rand -hex 32`	`admin caddy`, together	None
`STATE_HMAC_SECRET`	`openssl rand -hex 32`	`freeradius`	Logins waiting at the code prompt must restart
`LOCAL_RADIUS_SECRET`	`openssl rand -hex 16`	`freeradius`	None
RADIUS shared secrets	**Authorization Policies → RADIUS Clients**	RADIUS restarts	Devices fail until updated
CoA secrets	Change on the device and in Authcha	None	None
API tokens	Issue a new one, update automation, revoke the old	None	None

OTP service token with no downtime

`REST_OTP_TOKEN` authenticates the RADIUS server and console to the OTP service.
The OTP service accepts a **list** of tokens, so you can rotate in three steps:

```bash
cd /opt/authcha
openssl rand -hex 24          # the new token
# 1. otp.env: VALID_API_KEYS=<old>,<new>       then recreate otp_service
# 2. .env:    REST_OTP_TOKEN=<new>             then recreate freeradius admin
# 3. otp.env: VALID_API_KEYS=<new>             then recreate otp_service
```

Lab HANDS-ON

Use a test install for steps 3 to 6.

1. Open Settings (https://demo.authcha.cloud/settings) on the demo as `demo-viewer` (password `Authcha-View-2026`). The demo is public and read-only, so you can see where settings live but not change them.
2. On your install, read the key's fingerprint from the newest `radius-meta-*.txt` in `/opt/authcha/backups/`.
3. Run `sudo authcha kek-check` and read the counts.
4. Run `sudo authcha kek-rotate --dry-run`. Nothing changes.
5. Write down where your two escrow copies would be, and who can reach them.
6. Run the real `sudo authcha kek-rotate` on the test install. Move the old key to escrow, then run `sudo authcha backup verify`.

Cheat sheet KEEP THIS

- `PKI_KEY_ENCRYPTION_KEY`: 64 hex, **AES-256-GCM**, protects CA keys, SCEP, IdP signing keys, MDM and connector secrets.

- **Lost key = lost CAs.** Two offline copies, separate from backups.
- Check: `sudo authcha kek-check`. Rotate: `sudo authcha kek-rotate --dry-run`, then `sudo authcha kek-rotate` (about 1 minute without the console, RADIUS keeps running).
- After rotating: new key to escrow, old key kept for old backups, then `sudo authcha backup verify`.
- Changed `.env`? **Recreate** the service.
- Read more: Operations (https://demo.authcha.cloud/docs/operations).

Module 5 of 6 · 25 minutes

Monitoring, metrics and the audit trail

By the end you can

- Choose the console views that answer common questions
- Turn on the metrics listener safely and write the key alerts
- Explain how stuck accounting sessions are cleaned up
- State how long audit data is kept and how to search the archive

What to watch in the console

Where	What it answers
Dashboard	Is authentication healthy, and is anything needing attention?
Live Auth → Metrics	Accepts, rejects and errors per minute, top users and devices, updating live
Live Auth → Logs	The live RADIUS log, filterable by user, device address and result
Audit Trail → Auth History	Every authentication attempt and its result
Audit Trail → OTP Funnel	Codes requested, sent and verified. Shows provider problems quickly
Sessions	Active and past network sessions
System	Container health and console backups

Latency percentiles (p50 and p99) on the metrics page appear only after you set `LATENCY_CAPTURE_ENABLED=true` in `.env` and recreate `freeradius`. They are off by default because of a small per-request cost.

Metrics for Prometheus

The console can expose Prometheus metrics on a **separate** listener. It is off until you set, in `.env`:

```
METRICS_BIND=127.0.0.1:9188
# or, for a scraper on another host:
# METRICS_BIND=0.0.0.0:9188
# METRICS_TOKEN=<at least 16 random characters>
```

Then `docker compose up -d admin`. A non-loopback address **requires** a token of at least 16 characters. Without one the listener refuses to start, and it logs why. The endpoint is never served through the console's public port, it is plain HTTP so put TLS in front of it for a remote scraper, and answers are cached for 10 seconds.

Metrics carry no secrets and no personal data: labels are fixed words (`result`, `event`, `state`), never a user, device or destination name.

Alerts worth having

Alert	Why
Console down (`up == 0` for 3 minutes)	Nothing can be administered
`authcha_ops_up == 0`	Settings and certificate changes fail
`authcha_collector_ok == 0`	A section of the metrics cannot be collected
CA or server certificate expiring (`authcha_ca_expiry_days_min < 30`)	802.1X and RadSec break when they lapse
`authcha_license_days_left < 30`	New provisioning pauses after expiry
Reject ratio above half of recent logins	A broken device, a wrong secret or an attack
SIEM behind (`authcha_siem_lag_events_max` high)	Security events are not reaching your SIEM
`authcha_enforcement_unenforced_sessions > 0`	A limited user the firewall could not be told about
`authcha_kek_configured == 0` or plaintext keys	Secrets stored unencrypted

Also monitor RADIUS itself with a synthetic login from a registered test client, disk space on the host, and the backup and restore-drill logs.

Accounting sessions that never stopped

A device that loses power never sends its Accounting-Stop. Left alone, its sessions would stay open forever in Network Sessions, in quotas and in enforcement connectors. Every minute the console closes an open session that has had no interim update for **3 times its interim interval** (600 seconds if the policy sets none), but not sooner than **5 minutes** after its last update. The session is stopped at its last update time with the terminate cause `Lost-Service`, and enforcement connectors get the same stop event as for a real one. A session that never received any interim update waits 24 hours. The metric `authcha_accounting_sweep_last_run` shows the sweep is alive.

The audit trail

- Every change made through the console or API is recorded under **Audit Trail → Admin Actions**: who, what, when and from which address. That includes license installs, setting changes and SSO sign-ins.
- The console deletes admin audit entries older than `ADMIN_AUDIT_RETENTION_DAYS` (default **180**) each day. Set it in **Settings → Admin Security**; `0` keeps entries in the database indefinitely.
- The scheduled **audit-retention** job applies a 180-day window to the console audit log, OTP events, failed sign-ins and the RADIUS authentication log. It **archives before it deletes**, to compressed JSON-lines files in `/opt/authcha/backups/audit-archive/`, which only root can read.
- 180 days matches common security-log retention rules. If your rules need a *shorter* window, lower the setting. If they need a *longer* one, keep the archives.

Search an archive with:

```bash
gunzip -c /opt/authcha/backups/audit-archive/admin_audit_log-*.jsonl.gz | grep '"ali
```

Lab HANDS-ON

1. Open Live Auth (https://demo.authcha.cloud/activity/live) and the Metrics (https://demo.authcha.cloud/metrics) page on the demo (viewer login `demo-viewer`, password `Authcha-View-2026`). Find the accept and reject rates.
2. Open Audit (https://demo.authcha.cloud/activity/audit) and filter for a type of change. See who, what and when.
3. On your install, set `METRICS_BIND=127.0.0.1:9188`, recreate `admin`, and run `curl -s 127.0.0.1:9188/metrics | head`.
4. Pick two alerts from the table and write them as Prometheus rules.

5	Look at	`/opt/authcha/backups/audit-archive/`	and search an archive for one username.
6	Set	`METRICS_BIND=0.0.0.0:9188`	**without** a token and confirm the listener refuses to start (read the log). Add a token and try again.

Cheat sheet KEEP THIS

- Views: **Dashboard**, **Live Auth** (metrics and logs), **Auth History**, **OTP Funnel**, **Sessions**, **System**.
- Prometheus: `METRICS_BIND`, and a **token (16+ chars)** for any non-loopback address. Plain HTTP, no secrets in labels.
- Alert on: console down, ops down, cert expiry under 30 days, license under 30 days, reject spikes, SIEM lag, unenforced sessions, no encryption key.
- Stuck sessions close after **3 intervals**, minimum **5 minutes**, as `Lost-Service`.
- Audit: **180 days** by default, archived to `backups/audit-archive/` before deletion.
- Read more: Operations (https://demo.authcha.cloud/docs/operations), SIEM streaming (https://demo.authcha.cloud/docs/siem).

Module 6 of 6 · 30 minutes

Troubleshooting a failed sign-in

By the end you can

- Use the console tools in the right order to localise a failure
- Read the common reject messages and their causes
- Tell "no reply at all" apart from "a reply that says no"
- Collect what support needs

Start with the console

Most questions are answered without a shell. Work in this order.

1. **Try It** → **Auth Simulator** runs the decision for any user, device and source address and shows each step.
2. **Live Auth** → **Logs** shows RADIUS decisions as they happen.
3. **Audit Trail** → **Auth History** and **OTP Funnel** show past attempts and code delivery.
4. **System** shows container states and health.

Then the quick checks from the first operations module: `docker compose ps`, the health call, "ready to process" in the RADIUS log, and free disk space.

The simulator does not check secrets

It never checks the password, the code or the shared secret. If it predicts an accept but the real login fails, the cause is almost always the credentials, the RADIUS client secret, or the source address the device reports as Calling-Station-Id. Compare the real attempt in **Live Auth** with the simulator's input.

Two very different failures

The first question is always: **did Authcha answer?**

- **A reply that says no.** Authcha received the request and refused it. The reason is in **Live Auth** and **Auth History**. Read it.
- **No reply at all.** The device times out. Authcha silently drops requests it cannot authenticate, so there is nothing to read in the log of a refusal.

When there is no reply

Check, in this order:

- **The device is not a known RADIUS client, or the secret is wrong.** Check **Authorization Policies → RADIUS Clients** and wait for **Applied**.
- **A firewall blocks UDP 1812** between the device and Authcha.
- **The device timeout is too short.** The first request waits for your messaging provider. Use at least 10 seconds.
- **The device sends from a different address** than the one you registered. Set the device's RADIUS source interface.

To see whether packets arrive at all: `docker compose logs -f freeradius | grep -i "<device-ip>"`. If that is not enough, turn on RADIUS debug mode for a few minutes, and **turn it off afterwards**.

What the reject messages mean

Message	Meaning and fix
`Authentication failed`	Deliberately generic. Causes: unknown user, disabled or locked user, disabled user type, unregistered device, no jump server rule for the source (when required), no mobile number, or a lockout. Run the same login in the **Auth Simulator**
`account is locked after repeated failures or by an administrator`	Shown in Live Auth and Auth History only. The device still gets the generic message, on purpose. **Users** → **Unlock** clears it
`Incorrect credentials`	Wrong password or code. Three failures lock the account for the code lifetime
`Service temporarily unavailable`	The code could not be sent. See codes, below
`Service busy, try again shortly`	The server refused on purpose: overloaded, or the device sends too many password logins per second. The reason is in Live Auth
`OTP rate limit exceeded, please wait`	The user asked for too many codes in the window
`OTP sent, submit it as your password`	Normal in trigger-phrase mode. Log in again with the code
`Missing User-Name or NAS-IP-Address`	The device does not send NAS-IP-Address. Configure a RADIUS source interface

> **Why the device gets a vague message**
>
> Telling a device exactly why a login failed would tell an attacker whether a username exists. The precise reason goes to the log, and only the generic one to the device.

One-time codes not arriving

1. **Try It** → **OTP Delivery**: send a test code to the same number. If that fails, the problem is the provider path, not the login.

2. **Audit Trail → OTP Funnel**: find `send_failed` events and their error text for the user.
3. `docker compose logs --tail 100 otp_service`.

Usual causes: invalid provider credentials, a WhatsApp template that is not approved or has the wrong ID, an SMS sender or DLT template that is not registered (India), no outbound internet from the host (set a proxy), a missing or badly formatted mobile number, or a mismatched token after a manual edit (`REST_OTP_TOKEN` missing from `VALID_API_KEYS`; recreate `otp_service`, `freeradius` and `admin`).

802.1X and MAB

- **TLS handshake fails:** the client does not trust the server certificate, or checks the wrong name. Export the certificate the server presents and compare it to what the client expects.
- **Handshake works but login is rejected:** the username does not exist, is locked or disabled, or the password is wrong. 802.1X users need no mobile number.
- **The client is quarantined unexpectedly:** the user is enrolled for posture and the device is non-compliant, stale, or has not reported yet.
- **MAB:** `MAB not enabled on this NAS` (turn it on under **Devices**), `MAC not authorised` (not on the allowlist, disabled, or its class does not exist), `Monthly quota exceeded`, or accepted into the wrong VLAN (check the class VLAN and vendor attributes). If an allowlist change seems ignored, use **Flush cache** under **Endpoints → MAC Allowlist**.

Wrong privilege level

Check the effective user type in the simulator (a jump server rule may have lowered it), then **Privilege Mapping** for that user type and vendor. On Juniper, the local template user named in the reply must exist on the device. On FortiGate, turn on `accprofile-override` for the remote admin.

Contacting support

Send your Authcha version (on the sign-in page and in the `VERSION` file), the output of `docker compose ps`, and the relevant logs:

```bash
docker compose logs --since 1h freeradius admin otp_service > authcha-logs.txt
```

with what you expected, what happened, and the time. **Review the logs before sending**, because they can hold usernames, addresses and numbers. **Never send `.env`, `otp.env` or backups.**

Lab HANDS-ON

1. Open Auth History (https://demo.authcha.cloud/auth-history) on the demo (viewer login `demo-viewer`, password `Authcha-View-2026`). Find a rejected attempt and read its reason.
2. Open the Auth Simulator (https://demo.authcha.cloud/activity/simulator). Enter a user that does not exist. Read which step fails.
3. Open OTP Funnel (https://demo.authcha.cloud/otp-metrics) and find the counts of sent, verified and failed codes.
4. On your install, deliberately create each failure once: wrong password, a device with the wrong secret (watch for no reply), a user with no mobile number, a MAC not on the allowlist. For each, find the evidence in **Live Auth**.
5. Write a three-line runbook for your team: first check, second check, who to call.

Cheat sheet KEEP THIS

- Order: **Simulator, Live Auth, Auth History and OTP Funnel, System**, then shell checks.
- Did Authcha answer? **No reply** = client, secret, UDP 1812, timeout (10 s+), source address. **Reply with no** = read Live Auth.
- Generic device message by design. Precise reason in the log.
- Codes: **OTP Delivery test**, OTP Funnel, `docker compose logs otp_service`.
- MAB: enable per device, allowlist, class, **Flush cache**.

- Support: version, `docker compose ps`, 1 h of logs. **Never** `.env`, `otp.env`, backups.
- Read more: Troubleshooting (https://demo.authcha.cloud/docs/troubleshooting).