

Meridian Aegis

Installation and operations guide

Version 2.1.0 · https://meridianrp.es/aegis/ · Server-authoritative anti-cheat for FiveM

Contents

1. 1. Requirements
2. 2. Installation
3. 3. Operating modes
4. 4. Punishment philosophy
5. 5. The /aegis panel
6. 6. EventGuard, event contracts
7. 7. EntityGuard
8. 8. EconomyGuard
9. 9. Client sensors, combat and other detectors
10. 10. Commands
11. 11. Configuration (`shared/config.lua`)
12. 12. Licensing and remote control
13. 13. Privacy
14. 14. Troubleshooting and FAQ
15. 15. Exports (server API)
16. 16. Support

FiveM Security & Anti-Cheat Platform, version 2.1.0

Aegis is a server-authoritative security platform for FiveM. The client only reports; every decision (risk, kick, ban, block) is taken on the server. This document describes what the resource actually does, how to install it and how to operate it. If something you read here does not match what you see on your server, treat the server as the source of truth and contact support.

Versión en español: `DOCUMENTATION.es.md` (also at https://meridianrp.es/aegis/docs).

---

1. Requirements

Mandatory:

- FXServer with a recent artifact (build recommended by Cfx.re).
- OneSync enabled (`set onesync on`). Aegis is server-authoritative and needs OneSync to read entities and players.
- Lua 5.4. The manifest already declares `lua54 'yes'`; no action needed.

Optional (Aegis works without them):

- `oxmysql`: database persistence of bans, signals, actions and staff audit. Without it, bans are stored in `data/bans.json` inside the resource and survive restarts anyway.
- `screenshot-basic` (official Cfx resource): evidence screenshots attached to incidents. It needs a writable `.tmp` folder inside the `screenshot-basic` resource. Aegis tries to create it at startup and prints a warning if it cannot.
- Framework: `qbx_core`, `qb-core`, `es_extended` or standalone. Detection is automatic by resource state and is re-evaluated if the framework starts after Aegis.

2. Installation

1. Unzip the package into your `resources` folder. **Do not rename the folder**: it must be `meridian_aegis`. The sidecar path `@meridian_aegis/bootstrap/resource.lua` and the license binding depend on the name.
2. Make sure the `data/` folder exists inside the resource and is writable. It ships in the zip with a `README.txt`. Aegis writes `serverid.txt`, `settings.json`, `contracts.json`, `bans.json` and `aegis-support.json` there.
3. Nothing to configure for licensing. Access and duration are handled by Tebex and Cfx.re Asset Escrow. Optionally, link the server to your buyer portal: at startup the console prints `Portal code: XXXX-XXXX`; paste it at https://meridianrp.es/aegis/ → "Link server" (see 2.1). Aegis protects exactly the same whether you link it or not.
4. Grant panel access to your admin group:

```
add_ace group.admin aegis.admin allow
```

5. Recommended start order in `server.cfg`: Aegis works in any order (it re-detects the framework and the database when they start), but this order avoids warnings at boot.

```
ensure oxmysql            # optional
ensure qbx_core           # or qb-core / es_extended, if you use one
ensure screenshot-basic   # optional
ensure meridian_aegis
```

6. Start the server. In the console you should see (in Spanish until you switch `locale` to `en`): and, a few seconds later, `licencia válida` (valid license) if the server is linked to the portal. If it is not, the console shows `Server not linked · Portal code: XXXX-XXXX` instead; that is informational, protection is already running.

```
[Aegis:main] Meridian Aegis 2.1.0 listo · modo=shadow
```

2.1 The buyer portal and the portal code

Nothing has to be typed into your server to run Aegis. The resource is delivered through Cfx.re Asset Escrow and Tebex controls how long your server can use it.

The buyer portal at https://meridianrp.es/aegis/ shows your licence (plan, days remaining, renewal), your linked servers, the published version and the downloads. You enter it with your Tebex transaction id (tab "I just bought") or with the licence code it gives you (`AEGIS-XXXX-XXXX-XXXX-XXXX`). That code is only for the portal; it is never configured in the server.

To see a server in the portal, link it: when Aegis starts it prints `Portal code: XXXX-XXXX` in the console (also with the `aegis_portal` command). Paste it in the portal → "Link server". The code is derived from a hash of a random server id that Aegis generates on first boot and persists in `data/serverid.txt` (FXServer hides `sv_licenseKey` from resources, so the licence itself is never read or sent). It is stable for that server as long as `data/` is kept and reveals nothing about it; if you delete `data/`, the id and the code change and you have to link the server again (see 14). One licence links up to **2 servers** (production and development); you can unlink one from the portal at any time. If both slots are taken, a linked server that has not been seen for **14 days** is freed automatically when you link a new one. Linked servers also receive update notices.

2.2 Permissions in detail

- `aegis.admin` (ACE): opens the `/aegis` panel and unlocks the admin commands from in-game chat. By default, whoever has it is also **immune** to the anti-cheat (see 2.3).
- Console commands are registered as restricted. From the server console (txAdmin console included) they always work. To run them from in-game chat, the player needs the `command.<name>` ACE for that command **and** panel access. The default `server.cfg` line `add_ace`

`group.admin command allow` already covers `group.admin`; for another group add, for example, `add_ace group.mods command.aegis_status allow`. In practice, use the panel: everything the commands do is available there.

- Panel access without ACE: from the panel, Settings, "Who can use /aegis", the owner can grant access to specific players by name or identifier (`license:`, `discord:`, `fivem:`, `steam:`). The list is saved immediately to `data/settings.json` and survives restarts.
- Without ACE and without being on that list, `/aegis` answers "no permission" and does nothing.

2.3 Admin immunity

Admins use noclip, godmode, teleports and spawns as part of their job. An **immune** player is ignored completely: no signals, no risk, no kicks, no bans, no spawn or explosion budgets, no chat anti-spam, no EventGuard contract checks. A player is immune if any of these is true (`config.admins`):

1. ACE `aegis.immune` (`add_ace group.admin aegis.immune allow`).
2. Can open the panel (`aegis.admin`) and `admins.includePanelAdmins` is true (default).
3. Belongs to a framework group listed in `admins.groups` (default `admin`, `superadmin`, `god`).
4. Their identifier is listed in `admins.identifiers`.
5. A server-side hook `Aegis.ImmuneHook` returns true.

The list is re-evaluated every `admins.cacheSeconds` (60 s). The panel, Settings, "Immune admins" edits items 2 and 3.

2.4 Database

Two options, both valid:

- Import `aegis.sql` (InnoDB, utf8mb4) into your database before starting.
- Do nothing: with `oxmysql` started and `audit.persist = true` (default), Aegis creates the tables `aegis_bans`, `aegis_signals`, `aegis_actions` and `aegis_staff_audit` on first boot.

Old rows are pruned automatically at boot according to `audit.retentionDays` (default 30). Only signals with severity at or above `audit.persistMinSeverity` (default 50) or with hard evidence are written to the database. If you had bans in `data/bans.json` before adding `oxmysql`, they are migrated on the next boot.

3. Operating modes

Aegis boots in **Shadow** mode, called **Test** in the panel.

Mode	Panel name	What it does
`shadow`	Test	Detects, scores risk, records incidents and evidence, **learns legitimate event traffic to propose contracts** and shows everything in the panel. Does **not** kick, ban or block anything.
`live`	Live	Real enforcement: kicks, bans, event blocks, entity budgets, chat anti-spam.

Change it from the panel (mode selector at the top, then "Save changes" so it survives a restart) or with `aegis_mode live`. While the mode is not `live`, the console prints a reminder every 5 minutes so nobody leaves the server unprotected by mistake.

Things that force Test behaviour regardless of your setting:

- A signed **kill switch** or **forced mode** from the license backend, used only if a serious bug is found in a release. You will see it in the panel ops bar and in the console.
- If the license backend is unreachable, Aegis keeps working with a signed local cache (see 12). It **never** switches itself off because the backend is down.

In Test mode players are not notified. `shadowChat = false` by default; set it to true only while you calibrate yourself, because it tells the player what Aegis would have done.

Recommended rollout: 1 to 2 days in Test, review Incidents, set noisy detectors to "Log" in Detections, then switch to Live.

4. Punishment philosophy

The client reports, the server decides. The rules that matter:

- **Accumulated risk with decay.** Each signal adds a weighted contribution that halves every `risk.halfLifeMs` (5 min). Hard evidence (replay, impossible protocol) decays much slower (1 h).
- **Persistence counts 12-second windows, not bursts.** A detector can add at most one persistence step per `risk.persistWindowMs`. Thirty props placed when entering a house or nine clicks in a shop count once. Only a cheat sustained for minutes escalates.
- **Correlation.** Signals from different categories in the same window amplify each other. A non-hard detector alone never gets a player past **quarantine**; a kick by risk needs at least `risk.kickMinCategories` (2) distinct categories, or an explicit Kick/Ban action set for that detector.
- **Risk tiers** (`tiers`): 0 observe, 20 log, 40 block, 60 quarantine, 80 kick, 100 ban candidate. A ban candidate goes to the **Review** queue, never to an automatic ban.
- **Factory auto-ban**: only `honeypot.trap` (a trap event that no legitimate resource fires). `autoBan.enabled` can be turned off in Settings; then everything goes to the review queue.
- **Factory auto-kick** (`autoKick`, needs `minCount = 2` sustained samples of the same signal): `sensor.noclip`, `sensor.invisible`, `sensor.super_jump`, `sensor.weapon_mod`, `sensor.magic_bullet`, `sensor.no_recoil`, `sensor.vehicle_fly`, `sensor.speed`. Detectors that depend on the server

environment (godmode, vehicle godmode, infinite ammo, mass spawn, explosions) only add risk by default.

- **Per-detector action** from the panel (Detections): Ignore, Log, Kick or Ban for every detector. Stored in `detectorActions` and persisted with "Save changes". Kick and Ban set here act on the first sample.
- **Review queue with evidence**: every serious incident freezes an evidence case (telemetry before and after, signals, 2D replay, screenshots if available). Staff confirms the ban with the evidence or dismisses it, optionally marking it as a false positive, which lowers risk and feeds the detector quality metrics.
- **Bans by stable identifiers**: `license`, `discord`, `fivem`, `xbl`, `live`. Never by IP and never by Steam alone.
- **Quarantine**: restricts rewards, entity creation and economy without kicking. Released from the player drawer.

4.1 Public announcement

When Aegis kicks or bans (automatically or from the panel) in Live mode, it posts to chat in the Aegis language:

```
[AEGIS AC] PlayerName has been banned for Reason
```

Configurable in Settings, "Public announcement", and in `config.announce`: enabled, audience (`all`, `staff` only, `none`), show reason or not, prefix (max 24 chars), prefix color, character name or FiveM name, quotes. Nothing is announced in Test mode.

4.2 What the player sees

- Kick or ban: a disconnect screen with a human reason, in the Aegis language (es, en, pt).
- Reconnecting while banned: a connection card with the reason and an **Appeal** button that opens `enforcement.appealUrl` (Settings, "Appeal Discord"). Set it to your own Discord invite; the default is a placeholder.

5. The /aegis panel

Type `/aegis` in chat. Access requires `aegis.admin` or the staff list (2.2). Emergency close if the panel ever keeps the mouse focus: `/aegis_close`.

Top bar: mode selector (Test, Live), **Save changes** button (writes `data/settings.json`), language switch (ES, EN, PT; also `aegis_lang en`). Ops bar: mode, detector count, players online, protected events, health score, and a warning when a kill switch or a forced mode overrides the local mode.

Sections:

Section	What you find
Incidents	Grouped by player with history, risk and the detector that fired. Spectate live, open the replay, jump to the player. Archived incidents can be shown.
Players	Everyone online with status (clean, watched, alert). Opens the player drawer: risk, signals, spectate, kick, ban, quarantine, release.
Evidence	Saved cases with 2D replay (trajectory, shots, markers) and screenshots.
Banned	Date, staff, reason, evidence. Unban with one click.
Review	Ban candidates waiting for confirmation, with their case. Confirm, dismiss, dismiss as false positive, resolve appeals.
Detections	Every detector with its current action: Ignore, Log, Kick, Ban. Each one explains what it detects.
Settings	Mode, language, auto-ban switch, appeal Discord, who can use /aegis, public announcement, immune admins, weapon policy (Legal, Warn, Kick, Ban per weapon; requires `GrantWeapon` in your inventory).
Entities (advanced)	Live capture of entities created by players (who, model, provenance). Delete an entity or quarantine its creator.
Resource scanner (advanced)	ResourceShield: scans every resource, lists exploitable events with reasons, Protect and Unprotect buttons, restart button, per-event contract editor.
EventGuard (advanced)	Protected events and their contract statistics.
Economy (advanced)	Circuit breakers and anomalies; re-arm a breaker.
Detection quality (advanced)	Per-detector signals, confirmed, dismissed, false positives, p95 time. Restore a detector that auto-demoted to shadow.
Capabilities (advanced)	Capability ticket catalog and weapon catalog.
Wizard (advanced)	Onboarding checklist.
Diagnostics (advanced)	Environment checks (artifact, OneSync, database, framework, inventory, license, clock, permissions, resource order, known conflicts). Export a support bundle without secrets.
Health (advanced)	Runtime health and counters.

What "Save changes" persists: mode, language, per-detector actions, disabled detectors, weapon policy, anti-tamper challenge opt-in, auto-ban switch, appeal URL, public announcement, immune admins, staff list. The staff list is saved the moment you edit it.

6. EventGuard, event contracts

Most FiveM exploits are not "cheats": they are legitimate server events triggered with forged arguments (`TriggerServerEvent('job:reward', 999999)`). EventGuard lets you declare what a valid call looks like and rejects everything else, with evidence.

A contract (policy) can include:

Field	Meaning
`rate = { max, window }`	Maximum calls per window (ms) per player.
`args = { { index, type, min, max, equals, oneOf } }`	Schema for each argument; `equals` can reference session context (`'session.deliveryId'`).
`distance = { source, max }`	Player must be within `max` metres of a point (a session field or coordinates).
`requireSession = { type, stage }`	A security session of that type (and stage) must exist for the player.
`operation`, `completesOperation`	Names an operation so it can only be completed once per session.
`antiReplay = true`	The first argument must be a nonce issued by `IssueNonce`; each nonce is single use.
`signed = true`	Signed trigger (see below): the event only passes when the resource's own client fired it through the Aegis sidecar. Manual contracts only.
`permissions = { ace, job }`	Caller must hold the ACE or job.
`direction`	Expected direction (client to server) to catch spoofed internal events.
`advanceStage`	Moves the session to a new stage when the event passes.
`action`, `onViolation`	Override the action on violation.

Violations produce `protocol.*` signals (`rate`, `schema`, `distance`, `session`, `session_owner`, `session_expired`, `state`, `replay`, `duplicate`, `permission`, `direction`, `generic`). In Test mode or under a kill switch **no event is ever blocked**: the violation is recorded and the handler runs.

6.1 Option A, protect from the panel (no code)

Resource scanner, pick a resource, **Protect**. Aegis:

1. Scans the resource (Lua, JavaScript, C#) and creates a safe automatic contract for every client-reachable event with severity MEDIUM or higher. Automatic contracts only set a reasonable **rate**; session, distance, permissions and "derive the value server-side" appear as suggestions you enable by hand in the resource sheet.

2. Adds one marked, reversible line to the resource's `fxmanifest.lua`: with backup and checksum. That **sidecar** loads before the resource's own scripts and wraps `RegisterNetEvent`, `AddEventHandler` and `lib.callback.register`. On events with a contract it asks Aegis before running the handler. Fail-open: if Aegis is stopped or errors, the handler runs.

   ```
   shared_script '@meridian_aegis/bootstrap/resource.lua'
   ```

3. Shows the Security Diff (score before and after).

Panel states: Unprotected, Pending restart (manifest written, sidecar not yet loaded: run `refresh; ensure <resource>`), Protected and active (sidecar said hello). For the panel's Restart button to work, add:

```
add_ace resource.meridian_aegis command.refresh allow
add_ace resource.meridian_aegis command.ensure allow
```

Unprotect removes the contracts and the manifest line. Contracts live in `data/contracts.json` and reload at boot; editing one in the panel hot-reloads the sidecar. Resources protected by Asset Escrow are never modified: their contracts stay in EventGuard and you integrate with `ProtectEvent` or `RegisterProtectedEvent` in code.

6.2 Option B, SDK in your own resources

```
-- Declare the contract and register the handler in one call.
exports.meridian_aegis:RegisterProtectedEvent('delivery:complete', {
    rate = { max = 1, window = 3000 },
    requireSession = { type = 'delivery', stage = 'at_destination' },
    distance = { source = 'session.destination', max = 8.0 },
    args = { { index = 1, type = 'string', equals = 'session.deliveryId' } },
    antiReplay = true,
    operation = 'delivery', completesOperation = 'delivery',
}, function(src, deliveryId)
    -- Only runs if Aegis approved. The reward comes from the SERVER, never from the client.
    local reward = ActiveDeliveries[src].reward
    exports.qbx_core:AddMoney(src, 'bank', reward)
end)

-- Somewhere earlier, when the delivery starts:
local id = exports.meridian_aegis:StartSecuritySession(src, 'delivery', {
    stage = 'in_transit', destination = coords, deliveryId = 'A32', reward = 450,
})
exports.meridian_aegis:SetSessionStage(id, 'at_destination')

-- Anti-replay: issue a nonce server-side, hand it to your client (callback),
-- the client sends it as the FIRST argument of the protected event.
local nonce = exports.meridian_aegis:IssueNonce(src, 'delivery')
```

If you only want validation without changing your handler, call `ProtectEvent(name, policy)` once and `CheckEvent(source, name, ctx)` inside your handler; `ctx.args`, `ctx.nonce` and `ctx.playerCoords` are what you pass.

7. EntityGuard

Server-authoritative budgets for entity creation per player and per server (`entityguard.limits`), burst detection (`entityguard.burst`, with a connection grace of 45 s so streaming on login never trips it), forbidden explosion types and mission-population spawn limits (`entityguard.spawnLimits`, ped 12 / vehicle 12 in 6 s, object 30 in 4 s). Legitimate game explosions (vehicles, petrol pumps, fireworks and so on) are excluded through `entityguard.explosion.ignoreTypes`.

Injected menus: `sensor.injection`. The Aegis client snapshots its own script environment at start (global functions such as `TriggerServerEvent`, `GetEntityHealth`, the list of globals, debug hooks) and compares it every `sensors.integrity.intervalMs`. It catches menus that inject into the **Lua environment** and replace those references.

Honest scope (important): environment integrity does **NOT** detect engine-level premium menus (redEngine, Eulen, Tiago and similar). Those patch the game natives **in memory (C++)** and do not change the Lua reference, so the environment snapshot stays identical to clean. Those cheats are caught by **server-side behaviour** (godmode, teleport, mass-spawn, unregistered weapon, speed, invisibility), which is Aegis's **first line**. `sensor.injection` is one more layer, not the main one, and you should not sell or configure it as a "detector of all menus". `aegis_integrity_dump <id>` prints the client's environment diff to build signatures for a specific Lua-level menu.

Coverage in every resource: a menu can hook only the environment it targets, so the same watcher also ships inside the sidecar (`bootstrap/resource.lua`). Every resource protected with `aegis_protect <res> apply` or `aegis_protect_all` runs a client-side copy that snapshots *its own* environment and reports through Aegis (`IntegrityReport`), with a fallback straight to the server if Aegis is not loaded. The evidence names the resource where the hook was seen (`GetEntityHealth@es_extended`), and the server counts the same finding once per player every 5 minutes, so 150 sidecars seeing the same hook are one signal, not 150. Everything in the sidecar is `pcall`-wrapped and fail-open: a broken Aegis never breaks a protected resource.

Signed triggers (`signed = true`): the same event, `esx_drugs:pickedUpPoppy` say, is legitimate when the drug script fires it after the animation and an exploit when an executor fires it in a loop. Aegis tells them apart by origin. On join the server hands each client a session secret; the client sidecar living in every protected resource wraps `TriggerServerEvent` in that resource and, for events whose contract has `signed = true`, appends `counter:HMAC(secret, resource|event|counter)` as a last argument. The server sidecar verifies it and strips it before the real handler runs, so the resource never sees the token. An executor fires from its own environment, without the wrapper: the event arrives unsigned (or with a reused counter) and is blocked as `protocol.unsigned`, hard proof, severity 90. If the trigger comes from inside the resource's environment but not from one of its files (an injected chunk), the client also reports `sensor.caller`, a soft signal, because legitimate resources sometimes compile code with `load()`.

Enable it per event from the Scanner tab (checkbox "Signed trigger") or with `SetContract(res, event, { rate = ..., signed = true })`. Requirements: the resource must be protected (`aegis_protect <res> apply` or `aegis_protect_all`, restart after) so the client sidecar is loaded, and

the event must be fired from the resource's Lua client. Do not enable it on events fired from JavaScript clients or from NUI callbacks that bypass Lua, and not on events that other resources fire on this resource's behalf: those would arrive unsigned. In Test mode unsigned triggers pass and are logged as "would block", so you can enable it on a busy server and watch the panel before switching to Active. Honest limit: the secret lives in the client, so a menu that extracts it can sign; that is why the integrity sensor and the server-side contracts stay in place. Signed triggers kill the basic trigger, which is what almost every executor user actually does.

Polymorphic event protocol (`cloak = true`, **EXPERIMENTAL, off by default**): an evolution of signed triggers. Instead of signing the event, it **hides its name**: the sidecar rewrites the `TriggerServerEvent` of cloaked events into an `aegis:rpc` send with a token that **rotates per player and per epoch (15 s)**. An event logger always sees `aegis:rpc` with a different opaque token, so it cannot tell which one is "harvest", and whatever it captures is dead in the next epoch/session and for any other player: "log and replay" stops working. It includes **decoy** events (a map-dumper that replays everything triggers one, hard proof) and an anti-replay counter. It is **off by default** (`cloak.enabled = false` in `shared/config.lua`); it is an advanced layer you **must validate on your own server** before enabling, with the same rules as signed triggers (never on events fired from JS/NUI or from other resources). Same honest limit: the secret lives in the client, so it is combined with the behavioural detectors.

Signals: `entity.burst`, `entity.cage` (several client-spawned objects around another player within seconds, the typical mod-menu cage; in Active mode the object completing the cage is cancelled and the previous ones deleted, `entityguard.cage`), `entity.explosion`, `entity.forbidden`, `entity.unattributed`, `sensor.mass_spawn`. By default they add risk and appear in the panel; escalate them in Detections after you declare your own spawns. `deleteOnMassSpawn` (delete the entity in Live mode) is opt-in and off by default.

A resource that legitimately creates entities (garage, heist, dealership) declares them and is exempt:

```
exports.meridian_aegis:EntityExpect(GetCurrentResourceName(), 'vehicle', 1, 5000)
exports.meridian_aegis:EntityExpect(GetCurrentResourceName(), 'ped', 8, 10000)
```

8. EconomyGuard

Hooks the framework's AddMoney and watches gameplay income per player (`economy.player`, default 500k per 5 min, 300 items per 5 min) and per resource against its own baseline (`economy.resource.anomalyFactor` 8x). Reasons matching `economy.ignoreReasons` (bank, transfer, sell, society, casino, paycheck and so on, substring match) are not counted as income. On anomaly, a granular **circuit breaker** pauses only that resource operation for `economy.breaker.ttl` (60 s); re-arm it from the Economy section or `aegis_breakers`.

```
local r = exports.meridian_aegis:GuardReward({ resource = 'delivery', operation = 'pay', player =
src, amount = reward })
if not r.allow then return end
```

9. Client sensors, combat and other detectors

Client sensors (`sensor.*`) are treated as hints with low confidence; nothing is reported until the player is fully loaded (`sensors.loadedGraceMs`, 20 s). Server-derived invariants weigh more: `invariant.teleport` (over 200 m/s) and `invariant.speed` (sustained on-foot speed, three consecutive samples). Spawn, respawn and vehicle-exit graces are built in.

Combat Intelligence runs on the server from real damage events: `sensor.godmode` (health never drops under confirmed hits), `combat.damage_mod` (weapon damage override far above the catalog), `combat.firerate` (impossible cadence), `aim.snap`, `aim.reaction`, `aim.tracking` (accumulated profile over at least 10 shots at real players, NPC targets ignored). Aim detectors never punish alone; they correlate.

Other detectors: `weapon.forbidden` and `weapon.unregistered` (weapon policy plus `GrantWeapon` provenance), `chat.spam` (blocks flood server-side), `honeypot.trap`, `integrity.no_heartbeat` (anti-tamper challenge, opt-in through `sensors.challenge.enabled`, off by default because resource restarts and long loads can delay the client's answer).

Detector quality: if a detector's false positive rate exceeds `quality.fpRateDemote` (25 %) after `quality.minSamples` signals, it auto-demotes to shadow and the panel tells you. Restore it from Detection quality or `aegis_restore`.

10. Commands

All server commands require console or `aegis.admin` (or the staff list) plus the `command.<name>` ACE when typed from chat. `aegis_aimtest` and `aegis_selftest` are **testing tools**, not for production use.

Command	What it does
`aegis_status`	Version, mode, detector count, sessions and signal/action counters.
`aegis_mode <shadow, live>`	Changes the operating mode: `shadow` = Test (detects and learns, never punishes), `live` = Active. Save from the panel to persist.
`aegis_portal`	Prints the portal code of this server (to link it at https://meridianrp.es/aegis/).
`aegis_lang <es, en, pt>`	Changes the language of console and player messages.
`aegis_risk <playerId>`	Prints the current risk of a player.

Command	What it does
`aegis_scan [resource]`	ResourceShield scan of one resource or the whole server (score, coverage, counts).
`aegis_plan <resource>`	Proposes contracts for the sensitive events of a resource.
`aegis_protect <resource> [apply]`	Without `apply`, dry-run of the manifest diff; with `apply`, contracts plus sidecar.
`aegis_protect_all [escrow]`	Installs the sidecar in every started resource (skips escrow ones unless `escrow` is given). Restart the server afterwards. Undo per resource with `aegis_unprotect`.
`aegis_forensics <id> on/off`	**Forensic mode (research).** Dumps to the console and to `data/forensics.log`, timestamped, everything that player does (hooked functions, new globals, events fired, entities, every signal). To analyse a cheat on your test server and build signatures. Staff only; captures nobody by default.
`aegis_unprotect <resource>`	Removes contracts and the sidecar line.
`aegis_learn`	Lists contracts proposed from the traffic observed while in Test mode.
`aegis_breakers`	Lists active economy circuit breakers.
`aegis_cases`	Lists the last 20 evidence cases.
`aegis_case <id>`	Prints the replay markers of a case.
`aegis_review`	Lists pending ban candidates in the review queue.
`aegis_ban <reviewId> [reason]`	Confirms a review as a ban, with its evidence.
`aegis_dismiss <reviewId> [fp]`	Dismisses a review; with `fp` marks it as a false positive.
`aegis_quality`	Per-detector quality table (signals, confirmed, dismissed, FP rate, p95).
`aegis_fp <playerId> <detector> [suppress]`	Marks a false positive manually and retracts its risk.
`aegis_restore <detector>`	Re-enables a detector that auto-demoted to shadow.
`aegis_corpus`	Runs the false-positive regression corpus.
`aegis_caps`	Prints the capability ticket catalog.
`aegis_capture <on, off>`	Toggles live entity capture (also in the Entities section).
`aegis_report`	Prints the Aegis Security Report (summary of what Aegis saw and did).
`aegis_setup`	Prints the onboarding wizard checklist.

Command	What it does
`aegis_diagnostics`	Runs the environment checks.
`aegis_support`	Writes `data/aegis-support.json` (no secrets) to send to support.
`aegis_aimtest <on, off>`	Testing tool: lets aim analysis count NPC targets. Keep it off in production.
`aegis_selftest`	Testing tool: fires three synthetic signals to see the pipeline react.
`/aegis` (client)	Opens the control panel.
`/aegis_close` (client)	Force-closes the panel if it keeps the focus.
`/aegisunspec` (client)	Stops spectating and returns the staff member to their position.

11. Configuration (`shared/config.lua`)

`shared/config.lua` is loaded as **data**, not as a script. Aegis reads the file and evaluates it in a sandbox with no natives, no threads, no HTTP and no access to the rest of the resource. It may only declare the `Aegis.Config` table. If it contains anything else (a `CreateThread`, an `exports` call, `os.`, `_G`, and so on) Aegis refuses to start and prints a clear error: `[Aegis] CONFIG RECHAZADA: shared/config.lua contiene código no permitido (...)` (config rejected: disallowed code). Fix the file and restart.

Everything below is a default; the panel overrides several values at runtime and persists them in `data/settings.json`, which is applied on top of `config.lua` at boot.

Section	Keys	What it controls
`mode`	`shadow`, `live`	Boot mode. Ships as shadow (Test).
`shadowChat`	boolean	Tell the player in chat what Test mode would have done. Off.
`locale`	`es`, `en`, `pt`	Language of console, chat and disconnect messages.
`risk`	`max`, `halfLifeMs`, `hardHalfLifeMs`, `pruneBelow`, `correlation`, `persistenceStep`, `persistenceCap`, `persistWindowMs`, `kickMinCategories`, `noPersistence`	Risk engine: decay, persistence windows, correlation requirement. `pruneBelow` (0.5) discards contributions whose decayed weight falls under it; `correlation = { windowMs, perExtraCategory, maxBonus }` amplifies total risk per extra signal category seen inside the window (2 min, +12 % each, capped at +60 %).

Section	Keys	What it controls
`tiers`, `tierAction`	thresholds	Risk to action mapping. Ban candidate always goes to staff review.
`banCandidateThreshold`	number	Risk (100) at which a player becomes a ban candidate and enters the review queue; never an automatic ban.
`admins`	`enabled`, `ace`, `includePanelAdmins`, `groups`, `identifiers`, `cacheSeconds`	Immunity rules (2.3).
`staff`	`identifiers`	Initial list of players allowed to open /aegis; the panel maintains it.
`announce`	`enabled`, `audience`, `showReason`, `prefix`, `color`, `useCharacterName`, `quotes`	Public sanction announcement (4.1).
`enforcement.autoBan`	`enabled`, `detectors`, `categories`	Instant ban list. Factory: `honeypot.trap` only.
`enforcement.autoKick`	`enabled`, `minCount`, `detectors`	Sustained-signal kick list (4).
`enforcement.appealUrl`	URL	Appeal link on the ban card. Change it.
`enforcement.defaultFailMode`	`open`, `closed`	What a non-critical check does when Aegis itself errors: `open` (default) lets the action through; `closed` denies it. Keep `open` unless you know why.
`enforcement.tempBanSeconds`, `enforcement.quarantine`	`blockRewards`, `limitEntities`, `blockEconomy`, `isolateBucket`, `alertStaff`	Temp ban length; what quarantine restricts (see 15.1). `isolateBucket` (routing bucket isolation) is opt-in and off.
`detectorActions`	`{ ['detector.id'] = 'off', 'log', 'kick', 'ban' }`	Per-detector override, written by the Detections section.
`weaponPolicy`	`{ WEAPON_X = 'legal', 'warn', 'kick', 'ban' }`	Per-weapon policy, written by Settings.
`audit`	`ringCapacity`, `keepSignalsPerSession`, `persist`, `persistMinSeverity`, `retentionDays`	In-memory buffers, database persistence and pruning (30 days).

Section	Keys	What it controls
`entityguard`	`limits`, `burst`, `severity`, `signalWindowMs`, `forbiddenExplosions`, `explosion.ignoreTypes`, `spawnLimits`, `deleteOnMassSpawn`	Entity budgets and explosion rules (7).
`economy`	`player`, `resource`, `breaker`, `severity`, `ignoreReasons`, `signalWindowMs`	Income anomalies and breakers (8).
`sensors`	`heartbeatMs`, `reportRate`, `challenge`, `movement`, `confidence`, `godmode`, `loadedGraceMs`	Client sensor intake, movement invariants, anti-tamper opt-in.
`combat`	`snapDegPerSec`, `minSnapDeg`, `snapToShotMs`, `reactionMs`, `analyzeFrames`, `minShots`, `suspiciousRatio`, `damageModFactor`, `damageModMinDamage`, `damageModAbsolute`, `firerateIgnore`, `countNpcTargets`	Combat Intelligence thresholds (9). `minSnapDeg` (35°): a turn smaller than this is never a snap, so micro-adjustments are ignored. `snapToShotMs` (200 ms): the shot must land within this margin after the snap to count. `analyzeFrames` (40): recent telemetry frames analysed per shot. `damageModMinDamage` (100): `combat.damage_mod` only fires when the overridden damage is also at least this absolute value. `damageModAbsolute` (1000): for weapons with no catalog entry, only a value at or above this flags.
`chat`	`max`, `window`, `repeatMax`, `severity`	Chat anti-spam.
`quality`	`autoShadow`, `minSamples`, `fpRateDemote`, `evalWindow`	Auto-demotion of noisy detectors.
`evidence`	`preMs`, `postMs`, `telemetryHz`, `bufferSeconds`, `maxCases`, `maxFramesPerCase`	Evidence case capture.
`review`	`maxQueue`	Review queue size.
`learning`, `honeypot`, `capabilities`		Traffic-learning window (runs in Test mode), honeypot rotation (no seeds by default), capability ticket TTL.

Section	Keys	What it controls
`screenshots`	`enabled`, `webhook`, `quality`, `burst`, `cooldownMs`, `minRisk`	Evidence screenshots via `screenshot-basic`: burst of 4, 30 s cooldown per player, only when risk is at least 80 or Aegis is enforcing. `webhook` sends the images to your Discord instead of keeping them on the server.
`license`	`endpoint`, `recheckMinutes`, `bootGraceMinutes`	Informational: the portal endpoint comes from the encrypted module. There is no key to configure; the server identifies itself with the portal code (see 2.1).
`trustedResources`	`{ 'resource_name', ... }`	Resources allowed to call the exports that remove protection or inject signals (see 15.1). Default `{}`: only Aegis itself and the sidecar bootstrap of protected resources.
`investigate`	`spectateAce`, `entityCapture`, `entityCaptureMax`	Spectate permission and live entity capture.
`debug`	boolean	Verbose logging.

12. Licensing and remote control

- There is no key to configure. The server identifies itself to the portal with a hash of a random id persisted in `data/serverid.txt` (FXServer hides `sv_licenseKey` from resources, so the licence is never read, let alone sent) and a short portal code derived from that hash. Linking is optional and done from the portal; unlinked servers protect normally.
- Aegis validates at boot and every 30 minutes against `https://meridianrp.es/aegis/api/validate`. The response is HMAC-signed; an unsigned or badly signed response is ignored.
- One licence links up to 2 servers (production and development). Linking and unlinking happen in the portal with the portal code; a third link is refused until you unlink one, unless one of the linked servers has not been seen for 14 days, in which case that slot is freed automatically.
- The backend never switches protection off. If it is unreachable, Aegis keeps the last signed status in a local cache. If the linked licence is expired, suspended or revoked, Aegis keeps protecting and prints a warning so you can check the portal. Access to the resource itself is governed by Tebex and Asset Escrow.
- The provider can, for the whole fleet or one server, force Test mode, switch enforcement off (kill switch) or push a settings revision. All of this is signed and is only used for a serious bug in a release. The panel ops bar shows it.
- Update notices are signed too: a newer version prints a console line with the download URL, nothing is downloaded or executed automatically.

13. Privacy

Plain language, because it matters:

- Aegis **does not read files** on the player's PC. It has no kernel driver, no file scanner, no process list.
- Aegis has **no global ban list**. Bans are yours, stored on your server or your database, and are never shared with other servers or with the provider.
- Aegis **does not send screenshots or player data to the provider**. Screenshots stay on your server (memory ring and evidence cases) or go to the Discord webhook **you** configure in `screenshots.webhook`.
- What the license backend receives, every 30 minutes: a SHA-256 hash of the random server id in `data/serverid.txt` (your Cfx licence is never read), the portal code, the server's public IP (as the source of the HTTPS request), `sv_hostname`, the Aegis version and **aggregated counters** (players online, signals, actions, bans, mode). No identifiers, names, positions, chat or images.
- The support bundle (`aegis_support` or Diagnostics, Export) contains configuration and environment checks without secrets; you decide whether to send it.

14. Troubleshooting and FAQ

Console shows "Server not linked to the portal (optional). Portal code: XXXX-XXXX". Informational. Paste the code at https://meridianrp.es/aegis/ → "Link server" if you want this server in your portal and update notices. Protection is unaffected. `aegis_portal` prints the code again.

The portal says "code not found" when linking. The server has not contacted the backend yet (it does so at startup and every 30 minutes). Make sure Aegis is running and the server has outbound HTTPS, wait a minute and retry.

The portal says "slots full". Two servers are already linked and both have been seen in the last 14 days. Unlink one from the portal and retry. A linked server that has not contacted the backend for 14 days is freed automatically when you link a new one, so a dead test server never blocks you for long.

I deleted `data/` and the portal code changed. Expected: the code comes from the random id in `data/serverid.txt`, so a fresh `data/` means a fresh id. Link the server again from the portal with the new code; the old entry frees itself (immediately if you unlink it, or after 14 days without contact). Protection is not affected.

The panel does not open, or says "no permission". The player needs `aegis.admin` (check `add_ace group.admin aegis.admin allow` and that the player is really in `group.admin`) or must be on the "Who can use /aegis" list. From console, `aegis_status` tells you whether the resource is running.

No screenshots in the evidence. Install `screenshot-basic` and make sure the `.tmp` folder inside it is writable. Aegis logs a warning at boot if it cannot create it. Screenshots are only requested when risk is 80 or more, or when Aegis is enforcing, with a 30 s cooldown per player.

Bans disappear after a restart. Either start `oxmysql` (tables are created automatically) or make sure `data/` inside the resource is writable so `data/bans.json` can be written. The console prints "bans in data/bans.json" or "persistence ready" at boot.

A legitimate event is being blocked. Open Resource scanner, the resource, the event, and relax or remove the contract (rate too low, distance too short, missing session). Or Unprotect the resource. Remember that in Test mode nothing is blocked, so if it is blocked you are in Live.

False positives. Leave Aegis in Test for 1 or 2 days, review Incidents, set the noisy detector to Log in Detections, declare your spawns with `EntityExpect`, register weapon grants with `GrantWeapon`, and only then go Live. Mark false positives in Review or with `aegis_fp` so risk is retracted and the detector learns.

Aegis prints "CONFIG RECHAZADA" and does not start. `shared/config.lua` contains code that is not a plain configuration table. Undo the change or restore the file from the zip.

Mode keeps going back to Test after restart. Press "Save changes" in the panel after changing the mode, or set `mode` in `config.lua`.

Messages are in the wrong language. Disconnect messages, the ban card and the public announcement follow `locale`. Change it with `aegis_lang` or in Settings.

15. Exports (server API)

All exports are server-side and are called as `exports.meridian_aegis:Name(...)`.

Core:

- `ReportSignal(source, detector, data)`: reports evidence for a registered detector; severity and confidence come from the detector, not from the caller.
- `RegisterDetector(def)`: registers your own detector (`id`, `category`, `severity`, `confidence`, `description`).
- `GetRisk(source)`: current risk of a player.
- `GetSession(source)`: snapshot of the player's security session.
- `IsHealthy()`: true if the runtime is healthy.
- `GetLocale()`: current Aegis language.
- `Quarantine(source, reason)` and `Release(source)`: put a player in or out of quarantine.

EventGuard:

- `ProtectEvent(name, policy)`: declares a contract for an event.

- `RegisterProtectedEvent(name, policy, handler)`: declares the contract and registers the handler, validated before it runs.
- `CheckEvent(source, event, ctx)`: manual validation; returns `{ allow, sanitized, ... }`.
- `IssueNonce(source, op, ttl)`: single-use nonce for anti-replay.
- `StartSecuritySession(source, kind, context, ttl)`: opens a session and returns its id.
- `SetSessionStage(id, stage)` and `EndSecuritySession(id)`: advance or close a session.

ResourceShield and installer:

- `ScanResource(name)`, `ScanAll()`, `PlanResource(name)`: read-only scan and contract proposals.
- `ProtectResource(res)` and `UnprotectResource(res)`: the full Protect/Unprotect flow used by the panel.
- `SetContract(res, event, policy)` and `RemoveContract(res, event)`: manage sidecar contracts.
- `PlanInstall(res)`, `ApplyInstall(res, opts)`, `RollbackInstall(res)`, `Unprotect(res)`: manifest integration with backup and rollback.
- `SidecarHello(res)`, `SidecarContracts(res)`, `SidecarCheck(res, source, event, packed)`: used by the sidecar itself; you do not call them.

EntityGuard, economy, weapons and movement:

- `EntityExpect(resource, kind, count, ttl)`: declares upcoming legitimate spawns.
- `GuardReward(ctx)`: checks a reward against breakers and anomalies; returns `{ allow }`.
- `OnEconomyDelta(kind, ctx)`: feeds a custom economy delta.
- `ResetBreaker(res, op)`: re-arms a circuit breaker.
- `GrantWeapon(source, weapon, ttlMs)`: records a legitimate weapon grant.
- `AllowTeleport(source, ttl)`: exempts a legitimate teleport (staff tools, scripts).

Capabilities, learning and reports:

- `GrantCapability(source, capability, scope)`, `HasCapability(source, capability)`, `ConsumeCapability(source, capability)`, `RequestCapability(source, capability, scope)`, `CapabilityCatalog()`: temporary tickets (`teleport`, `noclip`, `invisibility`, `spectate`, `spawn_vehicle` and so on) so a legitimate action is not flagged.
- `ProposeContract(event)`: contract proposal for one event, built from traffic observed in Test mode.
- `SecurityReport()`: the Aegis Security Report as a table.
- `SupportBundle()`: the support bundle as a table (no secrets).

15.1 Trusted resources and what quarantine really does

A cheater who gets a server-side foothold (a compromised resource, an executor with a server module) must not be able to talk Aegis into switching itself off. So the exports that remove protection or inject signals only work when the calling resource is listed in `Config.trustedResources` (a table of resource names, default `{}`, which means only Aegis itself and the sidecar bootstrap of the resources it protects). The guarded exports are `ReportSignal`

when the detector is a hard one, `Unprotect`, `UnprotectResource`, `Release`, `AllowTeleport`, `GrantCapability` and `SidecarHello`. A call from any other resource is refused, does nothing and is logged in the console and the staff audit. Read-only exports (`GetRisk`, `ScanResource`, `SecurityReport` and so on) and the exports that add protection (`ProtectEvent`, `EntityExpect`, `GrantWeapon`, `GuardReward`) stay open to every resource.

Quarantine is a real restriction, not just a flag. While a player is quarantined, and according to `enforcement.quarantine`: rewards checked through `GuardReward` are denied (`blockRewards`), the entity budget is reduced so mass spawns are cut short (`limitEntities`), and framework money hooks and protected events for that player are denied (`blockEconomy`). Staff is alerted (`alertStaff`) and the player is released from the drawer or with `Release`.

16. Support

- Buyer portal, downloads and license status: https://meridianrp.es/aegis/
- Documentation online: https://meridianrp.es/aegis/docs (Spanish) and https://meridianrp.es/aegis/docs?lang=en (English); PDFs at https://meridianrp.es/aegis/docs/DOCUMENTATION.pdf and https://meridianrp.es/aegis/docs/DOCUMENTATION.es.pdf
- Support through Discord tickets: https://discord.gg/YkCtxAvbPD, button "Open ticket" in #support (also updates and false-positive reports).
- Before opening a ticket, run `aegis_diagnostics` and `aegis_support`, and attach `data/aegis-support.json`. It contains no secrets.
- Include your server's console lines starting with `[Aegis` around the problem.