CrossConnect
CONNECT. MONITOR. OPTIMIZE.
BY Cybr IQ

Experimental Features Explained

Technically accurate explanations of the five experimental capabilities CrossConnect is previewing. For each one: what it does, how it works from the signals we already collect, and where its limits are. Every one of them only reads and advises. None of them changes your network.

Audience: technical evaluators, network & security engineers, architects

Scope: the five capabilities under Admin → Experimental, grounded in the shipping code

Document: experimental-features explainer, 21 June 2026

Contact: contact_us@cybriq.io

0 How to read this document

Each section names a capability, explains the idea in everyday terms, then states the mechanism: the signals it reads, the service that computes it, and the boundary of what it can and cannot prove. Where a capability currently runs on illustrative data, or depends on a collector that is not yet connected, we say so plainly. An inference is called an inference, and a number that has a source is shown with that source.

LIVE ENGINE reads real collected signals today

HYBRID live grounding + illustrative examples in the preview

EXPERIMENTAL under Admin → Experimental; read-only, advisory, never applies a change

> **The rule they all share.** All five capabilities live in their own code packages, are reachable only from the **Admin → Experimental** menu, and are **read-only and advisory by design**. They look, explain, and recommend. Not one of them has any code path that writes configuration to a device or changes the source of truth. They follow the same advisory contract as the conversational AI: highlight the problem and explain how to fix it, but never carry out the fix.

1 What “experimental” means here

These five are previews: capabilities we are still shaping. They are kept in a clearly marked corner of the product, separate from the features you use every day. The "experimental" label is precise, not decorative. Here is what it guarantees.

Isolated by package

Each capability lives in its own Java package (`peekaboo` , `wobble` , `redtwin` , `sense` , `compliance.evidence`). Nothing the live product depends on imports them, so an experiment cannot destabilize the shipped product.

Read-only and advisory

Every service keeps no state and never writes anything. The worst thing that can go wrong is a wrong number on a screen, never a config change or an outage. Each view carries an “advisory only, no change has been applied” notice in its result.

Live-first, with honest fallback

Each one reads the real collected signals first. Where a background process or collector is not yet connected, the preview shows illustrative examples, clearly flagged with an `illustrative` marker on the report, so the capability can be shown without pretending a modelled value was actually measured.

Surfaced like any feature

Although experimental, each one is wired into every place the product requires: a reachable `@Route` view, a `FeatureCatalog` entry, a glossary term, and (where it fits) an advisory AI tool. This is checked automatically by `CapabilityCoverageTest`.

The map below shows how the five relate and what they lean on. All of them read signals CrossConnect already collects. Two of them stand on production capabilities documented elsewhere: Red Twin and (through its verdict) Gravitational Wobble use the same Batfish formal model and the production **Network Black Box** the assurance tooling uses, and Peek-a-Boo sits on top of the production AV endpoint classification. (A formal model is a precise mathematical representation of the network that questions can be tested against.)

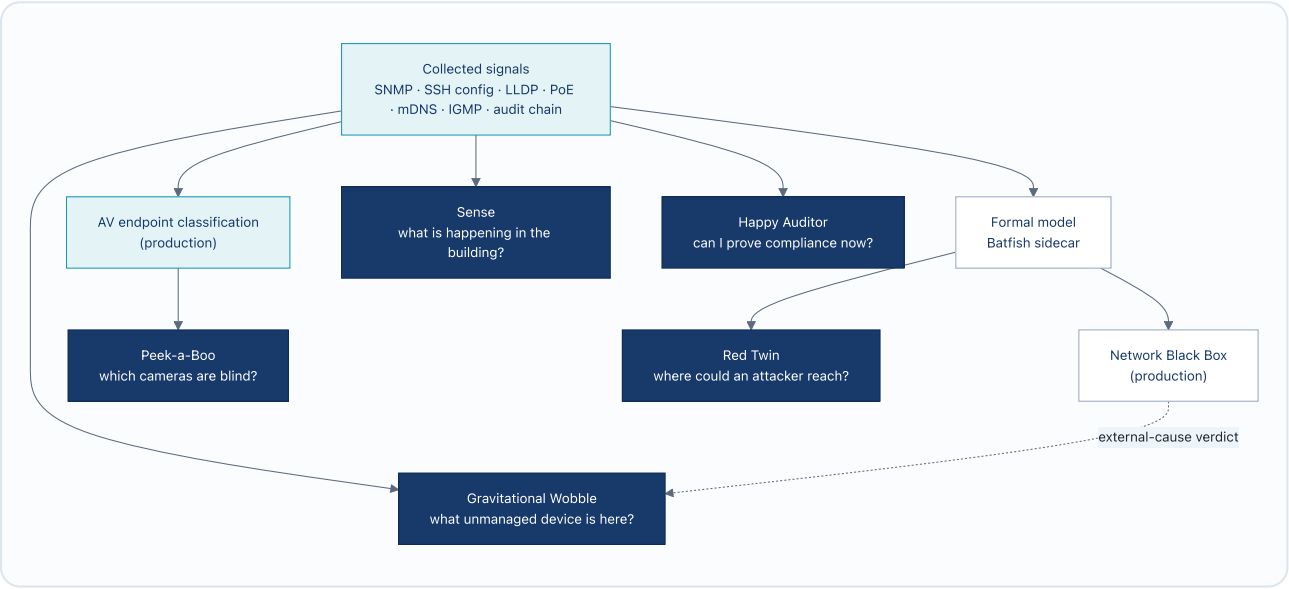

Figure 1. How the five relate. All draw on signals CrossConnect already collects. Two lean on the Batfish formal model and the production Network Black Box, Peek-a-Boo builds on the production AV endpoint classification, and all stay read-only.

2 Peek-a-Boo and the AV endpoint modules LIVE ENGINE

EXPERIMENTAL CAMERA LAYER

Know every audio-visual (AV) device on the network by type, codec, display, camera, microphone, or signal processor, from the signals the gear already gives off, then watch the ones that matter most: the cameras.

AV gear is the hardest inventory to keep honest. Integrators plug it in, it moves between rooms, and it rarely makes it onto the diagram. CrossConnect types it automatically from signals it already collects, and then, for cameras specifically, adds an experimental preview (Peek-a-Boo) that tells you not just that a camera exists but whether it is doing its job. The typing layer underneath is production; the camera preview on top is experimental.

The production layer: AV endpoint classification

The classifier (`AvEndpointClassifier`) sorts each device into an AV role from signals already in the database, then attaches a confidence label so a guess is never mistaken for a fact. It writes no new collection of its own. The roles and the views that surface them (`av-endpoints`, `av-classify-evidence`) are part of the shipped product, not the experimental corner.

Signal it reads	What it tells the classifier	Confidence
mDNS service type	Self-declaring AV announcement (`_dante`, `_qsys`, `_airplay`)	Confirmed
AvMediaFlow protocol	A live media stream in a known format (Dante, AES67, NDI, Q-SYS)	Confirmed
LLDP / SNMP model string	The device names its own make and model	Confirmed
OUI vendor	The MAC address vendor hints at a likely role	Inferred
DHCP fingerprint	The device family from its DHCP request pattern	Inferred

The roles it assigns are `av-codec`, `av-display`, `av-camera`, `av-microphone`, `av-dsp` (digital signal processor, the audio matrix in a room), `avoip-encoder` and `avoip-decoder` (AV-over-IP source and screen), `av-control` (the room control processor), `av-room` (a meeting-room PC), and `av-speaker`. Per-type modules then go a step further and ask the device itself how it is doing, live first and falling back to the classified record when a probe is not available: **Displays** (`av-displays`) over PJLink for power and input state, **signal processors** (`av-dsp`) over Q-SYS for run-state and loaded design, and **codecs** (`av-codecs`). When a typed device is not in inventory, it is surfaced as a shadow AV endpoint for an operator to confirm, never created automatically.

Peek-a-Boo: the experimental camera layer EXPERIMENTAL

Cameras are the AV devices a security team cares about most, and the ones most likely to be quietly broken. Peek-a-Boo (`PeekABooService`, route `peek-a-boo`) starts from the cameras the classifier already found (by vendor, OUI, mDNS, ONVIF announcement, and existing media flows) and adds a condition read: not just online or offline, but a **watch-state**, live, blind, looped, static, or no-stream, plus flags for cameras that are prohibited under NDAA section 889, exposed to a known CVE, gone dark, or silently swapped for different hardware. An operator can acknowledge or clear these conditions in bulk; a read-only viewer cannot.

Peek-a-Boo is genuinely experimental. The early stages are live (build the candidate set, resolve MAC to IP, and correlate each camera with its switch location, firmware age, and CVE exposure). The deeper stage, a live ONVIF probe that logs into the camera to confirm it is really streaming, is built and real-first but **dormant by default** (`crossconnect.peekaboo.probe-enabled=false`) until camera credentials are wired in. Until then a camera is reported from classification and inventory, and the report says so.

Figure 2. One classifier, many AV modules. The production classifier types every AV device from signals already collected; the per-type modules add a live read. Peek-a-Boo is the experimental camera module on top, with its deeper live-streaming probe dormant until credentials are supplied.

> **It looks, it never touches.** Peek-a-Boo reads a camera's status, firmware, and features. It does not move a pan-tilt-zoom (PTZ) camera, reboot it, or change any camera setting. Camera control is a deliberately separate, later, gated phase. Like every capability here, it is read-only and advisory: it tells you a camera is blind, it does not fix it for you.

3 Gravitational Wobble HYBRID EXPERIMENTAL

Find a device you do not manage, one you never discovered and that gives off no SNMP, no config, and no LLDP, by the mark it leaves on the equipment you already watch.

Astronomers find an unseen planet without ever photographing it: they watch the visible stars and measure the tiny wobble that the unseen planet's gravity causes. A network has the same shape. The riskiest thing on it is often the device nobody knows is there: a switch plugged in under a desk, an unauthorized wireless access point, or a hidden device spliced into a path. It never announces itself, but the moment it does anything that matters it disturbs the state of the managed devices around it, and we read that disturbance. We do not detect the device directly. We detect its wobble, and from the wobble we locate it, classify it, and tie it to whatever it broke.

How it works

`GravitationalWobbleService` runs read-only (`@Transactional(readOnly = true)`) and matches what it sees live against the managed inventory. Two wobble signals are live today:

- **Unknown LLDP neighbour.** LLDP is the protocol switches use to announce themselves to their neighbours. A managed switch reports an LLDP neighbour (`DiscoveredNeighbor.remoteSysName`) that matches no device in inventory. The matcher `unknownNeighbours(...)` ignores blank names and known devices, removes duplicates by (device, port, identity), and produces a finding pinned to the exact port.
- **Unmanaged endpoint.** `RogueDeviceService.unmanaged(tenantId)` surfaces a MAC or IP address seen in forwarding or ARP tables that belongs to no device in inventory: an unknown host on a known port.

Each `WobbleFinding` is a **ghost node**. It records the managed device and port it hangs off, an optional VLAN, the inferred identity (remote name, or MAC/IP address), a severity, a confidence label that always begins with *Inferred* (for example "Inferred (LLDP neighbour not in inventory)"),

and a plain-language recommendation, never an action. The design extends this list to many more kinds of wobble: sudden spikes in the number of MAC addresses on a port, ARP-to-MAC changes over time, MAC-flap and STP churn from an unauthorized loop, rogue DHCP servers, and the silent inline device that quietly absorbs traffic. Those depend on a syslog collector and time-based comparison snapshots that are **not yet connected**. The document is clear that today only the LLDP and endpoint signals are live.

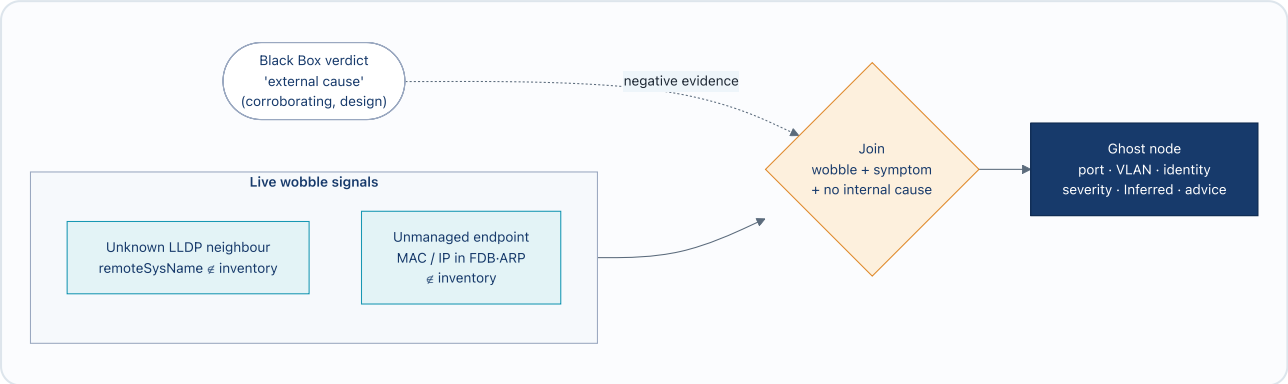

Figure 3. From wobble to a localized ghost. A wobble signal plus a symptom plus the Black Box's "external cause" verdict triangulates to one unmanaged device, placed on the exact managed port that sees it. The dashed path marks the corroboration the design adds next; the solid paths are live now.

> **Honest limits, on purpose.** No packet capture, anywhere. We see the footprint, not the device itself. We pin it to a port or segment and classify it by its signature, but we cannot read its config or prove its make and model. So a finding reads "an unmanaged device consistent with X," never "device X model Y." A truly passive device that breaks nothing leaves almost no wobble, and that is the correct result: we detect what is causing real effects, not every harmless gadget.

4 Red Twin HYBRID EXPERIMENTAL

A safe attacker, set loose on a faithful copy of your network inside the formal model, that finds the path to your most sensitive systems and writes the single line that closes it, before anything happens for real.

The only way to be sure a burglar cannot reach the safe is to try, and you would never run that test on your real network. Red Twin runs it against the **Batfish formal model**, the same mathematical model that already powers reachability and ACL analysis. It explores a copy, so the real network is never touched, and it produces two kinds of finding.

- **Attack paths.** A path an attacker could follow from a starting foothold (a guest VLAN, or a compromised AV endpoint) to a high-value target (cardholder data, an OT controller, a domain controller). Each `AttackPath` records its hops, the `weakLink`, the single `minimalCut` config line that cuts the path, a severity, and a confidence percentage. That cut is re-checked in the model to confirm it closes the path without breaking something else.
- **Failure futures.** A simulated failure (a switch reboot, a lost circuit, a failed power supply) that would cascade into an outage. Each `FailureFuture` records its trigger, how far the damage spreads, the `cheapestFix`, and a rough cost.

Red Twin stays grounded in your live network through a `ModelScope`. This records whether the formal model is online, how many real segmentation exposures exist today (management interfaces left exposed, cameras reachable from the guest network, shadow devices on sensitive segments, all totalled from the live `AvSegmentationAnalyzer`), and how many sensitive endpoints there are. The exposures it counts are the *real* footholds. The attack paths and failure futures shown in the preview are an **illustrative** set (flagged `illustrative: true`) so the capability can be shown before the background scenario runner is connected. The live grounding is real; the worked examples are for illustration.

The cost outlook

`RedTwinEconomics` adds an advisory cost layer. For each finding it matches the closest profile in a sourced reference table and shows, ahead of time, the **cost if it happens** and the **cost to fix it now**, with a return-on-investment (ROI) line. Attack paths match to breach costs by the type of target (cardholder/PCI, OT/SCADA, domain controller) against IBM 2024–2025 ranges; failure futures scale by severity against Uptime Institute 2025 and ITIC 2024 figures. Every number is a **range with a source and a caveat**. A `CostProjection` records where its figure came from and a confidence of `SPECIFIC` (a matched profile) or `BROAD` (a fallback range). When the match is weak, the interface says "no close cost match" rather than inventing a precise figure.

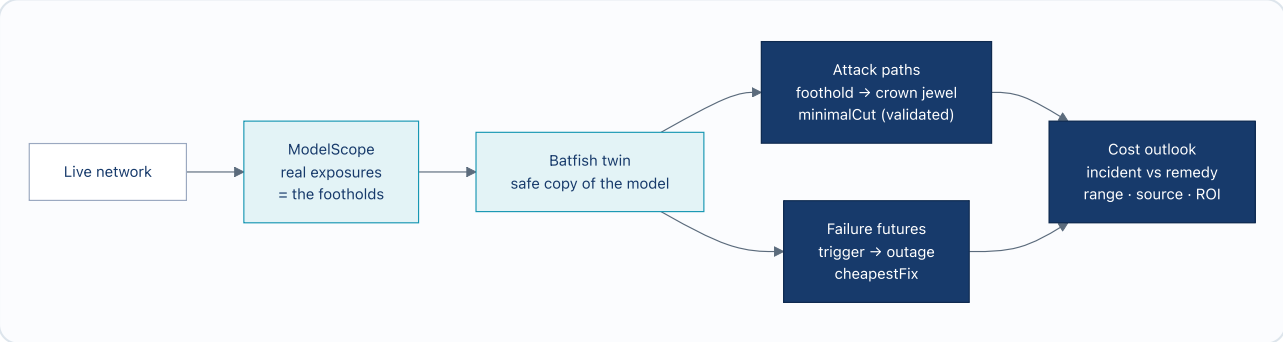

Figure 4. A safe adversary in a copy. Red Twin reads the real exposures as footholds, attacks the Batfish copy (never the live network), and for each finding writes the single severing change plus a sourced, advisory cost outlook. The fix is validated in the model and never applied.

> **Advisory by design.** Red Twin has no way to apply a change. Every cut is labelled "one line, validated in the model, advisory, never applied," and an audit-chain entry timestamps every warning, so a risk flagged today is provably on record before it causes harm. Reports are recomputed on a short cache timeout, and the service keeps no changeable state of its own.

5 Sense HYBRID EXPERIMENTAL

Read the physical and human state of a building, which rooms are in use, what is drawing power, and what just got unplugged, from the switches you already own, with no cameras, badge readers, or motion sensors.

Every meeting room runs on a network switch, and that switch already tracks more than traffic. It sees when displays and microphones wake up, how much power they pull over PoE (Power over Ethernet, which delivers electricity to devices through the network cable), the announcements that

AV gear broadcasts, and the moment a device goes dark. Sense is a thin, read-only layer that *derives* meaning from signals already in the database, turning them into a live, per-room view of who is present and what is happening physically. It inspects no packets and adds no new collection.

What it derives, and from what

Signal	Plain meaning	Derived from (staged entities)
Presence	A room is occupied or idle right now	PoE draw rising on the room's ports + recent mDNS/IGMP activity + reachability recency
AV-in-use	A live audio/video session is running	`DiscoveredIgmpMembership` on an AV group + active `DiscoveredMdnsService` (`_dante` , `_ndi` , `_airplay` , `_rtsp` , …) + healthy `DiscoveredPtpClock`
Energy	Watts drawn per room or floor	Sum of `DiscoveredPoe.consumptionPowerW` over the room's ports
Physical-tamper	A device was physically unplugged with no work order to explain it	PoE draw → 0 **and** interface oper-status down **and** LLDP neighbour vanished **and** no matching change in the audit chain

`SenseService` caches results per tenant on a 15-second timer (`TtlMemo`), since physical state changes at human speed. It reads live counts today, distinct AV mDNS services, PoE switches and total watts, and multicast groups, through `gatherReal(...)` , which is wrapped so that a failed query degrades gracefully instead of crashing the view. Presence and AV-in-use are **inferences**: each room carries a confidence percentage built from weighted `ConfidenceFactor` s, and is clearly labelled as inferred, never presented as a direct sensor reading.

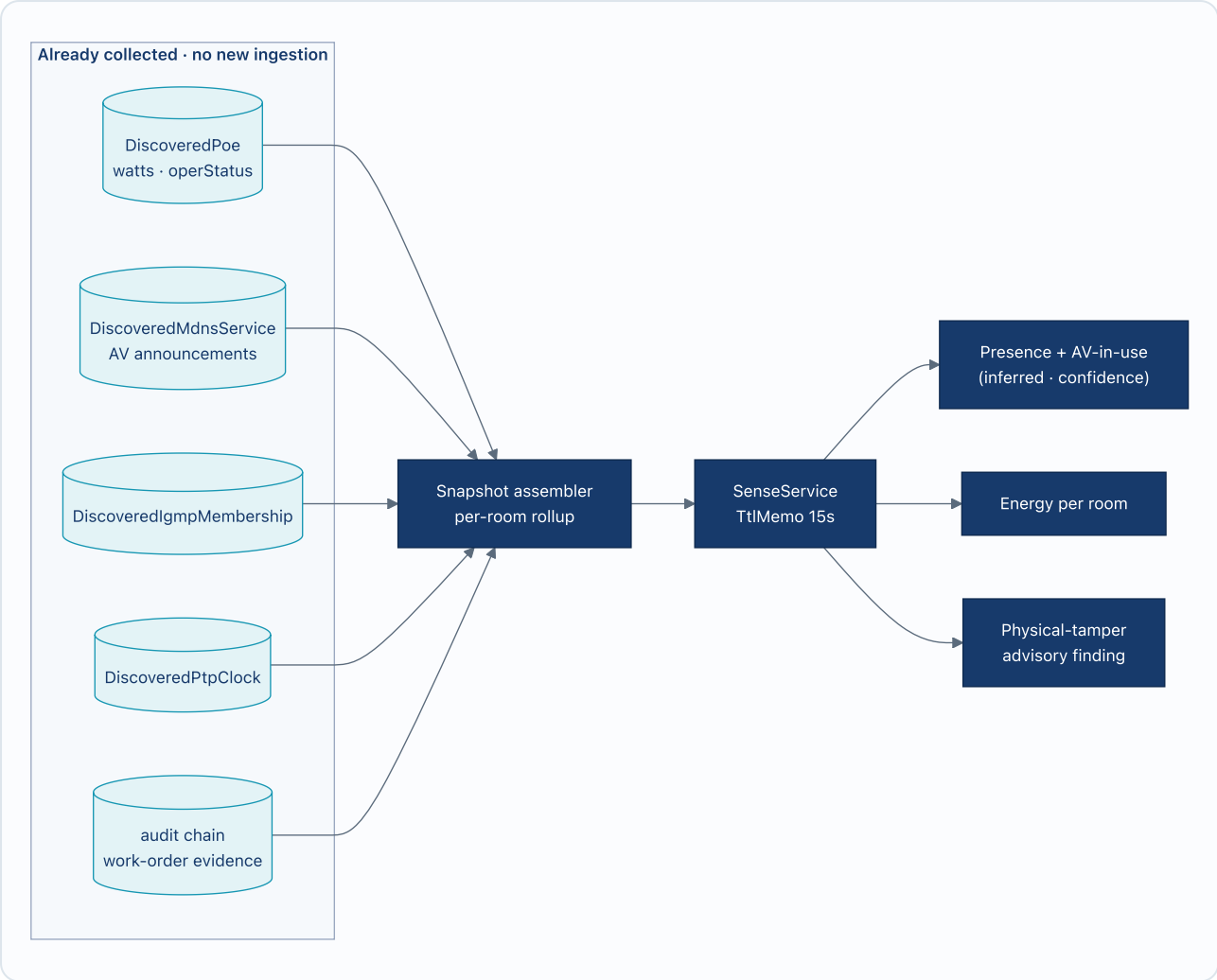

Figure 5. Presence intelligence from the switch. Sense is a pure function over signals the discovery sweep already gathers. The tamper alert fires only when power dropping to zero, the link going down, and a vanished LLDP neighbour all happen together with no matching operator change in the audit chain. That combination is the signature of a physical unplug, not a planned administrative shutdown.

Take it for what it is. Occupancy here is *inferred, not sensed*: it is presence intelligence, never life-safety or access control. How fine-grained it gets follows the wiring. Device-level location is automatic, but a shared wiring-closet switch needs an optional per-port-to-room mapping to tell rooms apart. PoE coverage varies by vendor, and gear running on wall power needs an optional PDU (power distribution unit) input to report its energy use. In the preview, the room tiles, usage patterns, and the tamper example are **illustrative** (marked with the report's `illustrative` flag), layered over the real live counts.

6 Happy Auditor LIVE ENGINE EXPERIMENTAL

Audit-ready in one action: a dated, point-by-point evidence pack with a single score and a tamper-evident proof that nothing was edited after the fact.

An audit should not cost three weeks of screenshots and spreadsheets. Happy Auditor builds the proof while you watch. One action produces a dated evidence pack for a chosen compliance framework: what is satisfied, where the gaps are, a single score, and a verdict on whether the audit trail behind it is intact.

What the pack contains, and how it is built

`EvidencePackService.build(tenantId, frameworkKey)` takes a snapshot at the moment you generate it, combining several live evaluations into one `EvidencePack` record:

- **Controls:** from `ComplianceService.evaluate(...)`, each control shown as Satisfied / Gap / Not applicable (`PASS` / `FAIL` / `NOT_APPLICABLE`) with its supporting evidence text, plus counts and a score (`scorePct`).
- **Segmentation:** what policy says must stay isolated versus the traffic actually observed (`segMustIsolate`, `segVerified`, `segLeaks`).
- **Drift:** devices that have drifted away from their approved golden-config baseline (`driftCount` and the list).
- **Audit-chain integrity:** `EventAuditService.verifyChain(tenantId)` returns an `AuditIntegrityReport` (`intact`, `total`, `verified`) that becomes the pack's `auditChainIntact` verdict, plus a sample of recent audit entries.

The evaluation runs a Batfish-backed compliance check off the UI thread, so it never freezes the view. The pack renders to a **self-contained HTML document** (with its styling embedded and no external dependencies) or to a **CSV** file for auditors who work in spreadsheets. Both are produced by `EvidencePackExport`, a self-contained renderer, so the same input produces the same bytes every time. Nothing is written to a file on the server; the export is handed to the browser as a download.

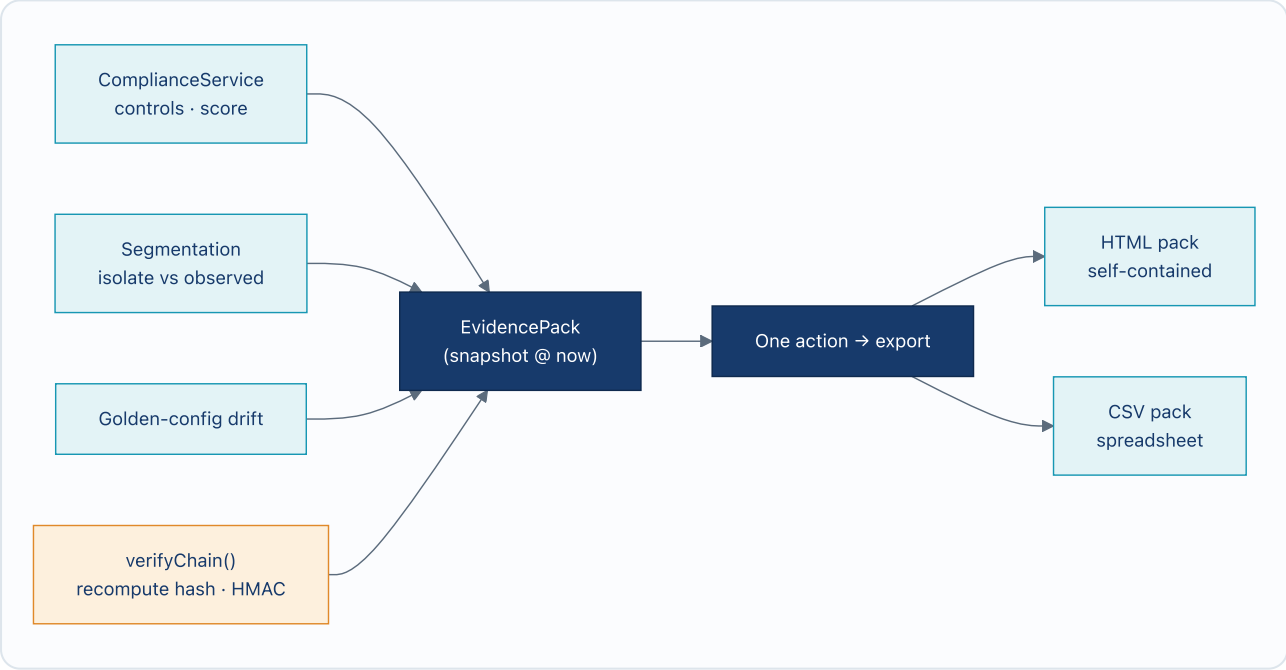

Figure 6. One action, a sealed evidence pack. Compliance, segmentation, drift, and a freshly recomputed audit-chain verdict combine into a dated pack. The chain verdict is the seal: each entry's hash incorporates the previous entry's hash and is HMAC-signed, so altering any past row breaks the chain and shows up as BROKEN.

> **Why an auditor accepts it.** The integrity verdict is not just a claim, it is a recalculation. `verifyChain` walks the entire audit trail and re-derives every entry's SHA-256 content hash (each of which folds in the prior entry's hash, starting from a fixed `GENESIS` starting point) and its HMAC signature. If a single past record was altered, every hash after it fails to match and the pack reads BROKEN rather than Intact. The whole pack is advisory and read-only: it reports your posture, it changes nothing.

7 What is real, and what is illustrative

The rule across the product is real-first: every capability tries the real source first, and clearly labels anything it cannot fully ground in live data yet. The table below is the honest summary of where each one stands today.

Capability	Route	Live engine today	Illustrative / not-yet-wired
AV endpoint modules (production)	`av-endpoints`	Live classification into AV roles with confidence; Displays (PJLink) and signal processors (Q-SYS) probed live, real-first	Codec-specific probes and some per-vendor probes are still being added
Peek-a-Boo (cameras)	`peek-a-boo`	Live: candidate set, MAC-to-IP resolve, and correlation of each camera with location, firmware age, and CVE exposure	The deeper live ONVIF streaming probe is built but dormant by default until camera credentials are supplied
Gravitational Wobble	`wobble`	Unknown-LLDP-neighbour and unmanaged-endpoint findings from live data	Faster syslog-driven wobbles (MAC-flap, STP, rogue DHCP) and time-based ARP/FDB comparisons await the syslog collector; no vendor (OUI) lookup yet
Red Twin	`red-twin`	Live `ModelScope` exposures from the segmentation analyzer; cost engine with sourced ranges	The attack paths and failure futures shown are an illustrative set; the background scenario runner is not yet connected
Sense	`sense`	Real live counts: AV mDNS services, PoE switches and watts, multicast groups	Room tiles, usage patterns, and the tamper example are illustrative; per-port room mapping is optional and planned
Happy Auditor	`happy-auditor`	Full: real compliance, segmentation, drift, and recomputed audit-chain verification; HTML/CSV export	Everything is live; nothing in this one is illustrative

> **One rule across all five.** A value read from the live source is shown as exactly that. An illustrative or inferred value is labelled, and a modelled number never poses as a measured one. None of the five can change your network: they are read-only adversaries, auditors, and observers, kept in a clearly marked experimental corner of the product.