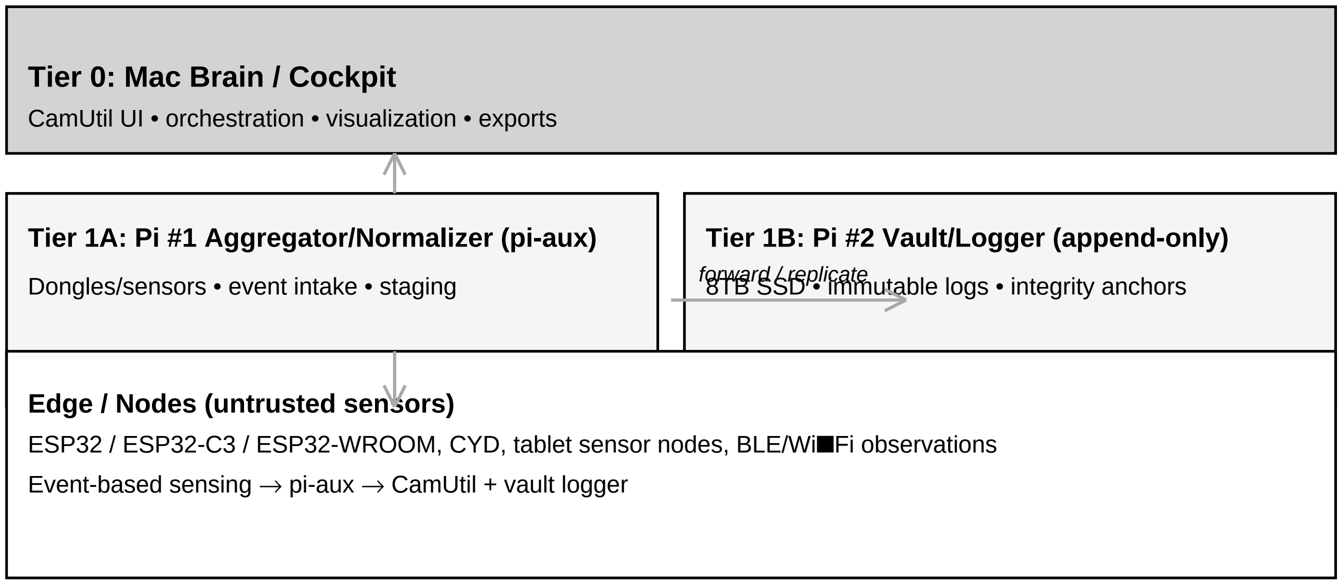
StrangeLab • CamUtil Chronicle

Scope: what we built/changed today in CamUtil, how it runs, where it writes evidence, and how it ties into the lab.



Tier 0: Mac Brain / Cockpit
CamUtil UI • orchestration • visualization • exports
Tier 1A: Pi #1 Aggregator/Normalizer (pi-aux)
Dongles/sensors • event intake • staging
Tier 1B: Pi #2 Vault/Logger (append-only)
forward / replicate
8TB SSD • immutable logs • integrity anchors
Edge / Nodes (untrusted sensors)
ESP32 / ESP32-C3 / ESP32-WROOM, CYD, tablet sensor nodes, BLE/Wi■Fi observations
Event-based sensing → pi-aux → CamUtil + vault logger

1) What CamUtil is now

- Local-first macOS app for scanning *authorized* networks and nearby BLE, producing evidence-grade exports (RAW + READABLE) under ~/StrangeLab/scans/.
- Network scan pipeline: ARP presence + open ports + HTTP fingerprinting + SSDP/ONVIF discovery + optional RTSP probing (guarded by Safe Mode).
- BLE pipeline: passive advertisement parsing, company/vendor decoding, service UUID decoding, RSSI smoothing + Find/radar UI, optional probing removed/kept minimal per your constraints.
- Confidence scoring for hosts and BLE peripherals with deterministic reasons; used for sorting and report summaries.
- Pi-Aux integration path: pi-aux can send local events into CamUtil for correlation (token-gated local HTTP).

2) Build + run (CLI)

Build command used (Debug):

```
xcodebuild -project /Users/letsdev/CamUtil/CamUtil.xcodeproj \
-scheme CamUtil -configuration Debug -destination 'platform=macOS' build
```

Run (opens the built .app from DerivedData):

```
open ~/Library/Developer/Xcode/DerivedData/CamUtil-*/Build/Products/Debug/CamUtil.app
```

Verify which binary actually launched:

```
pgrep CamUtil
ps aux | grep CamUtil.app
lsof -p $(pgrep CamUtil) | grep CamUtil.app
```

Notes from the session:

- zsh globbing: the unicode ellipsis character (…/DerivedData/...) is not '...'; it will error. Use normal path + quotes when needed.
- zsh comments: if you paste '# comment' into zsh with weird prefix characters, it can parse as a command. Keep comments outside copy blocks.
- Bluetooth usage strings: Info.plist contains NSBluetoothAlwaysUsageDescription so macOS will prompt properly.

3) Evidence outputs (where files go)

All exports are local-only under:

```
~/StrangeLab/scans/
```

Primary subfolders we wired up:

- evidence-raw/ and evidence-readable/ — per-host evidence bundles.
- ble-raw/ and ble-readable/ — per-BLE fingerprint bundles (decoded + raw).
- logs-raw/ and logs-readable/ — runtime logs (and readable summaries).
- audit-raw/ and audit-readable/ — overall scan audit exports.
- device-report-raw/ and device-report-readable/ — per-device narrative reports.
- scan-report-raw/ and scan-report-readable/ — whole-scan narrative reports.
- rtsp-hard-probe/ — per-target VLC+diagnostics runs (folder per attempt).

Operational rule we locked in:

- Buttons that produce artifacts export to file (and can auto-reveal). Clipboard is optional for convenience, not the primary evidence path.
- Readable exports include human-friendly labels + explanations, while raw exports preserve exact raw fields unchanged.

4) Safe Mode and why probes were 'blocked'

Safe Mode disables active actions (like ONVIF RTSP fetch / RTSP probe) and only allows passive discovery. When Safe Mode was ON, the app logged lines like:

```
Safe Mode on: blocked RTSP probe for 192.168.x.y
```

Fix: turn Safe Mode OFF for your own devices when you explicitly want RTSP/ONVIF fetch.

Hard Probe (VLC + Diagnostics)

- Added as a button only on the Cameras/Interesting detail pane.
- Writes a status JSON even when it cannot run (missing RTSP URI, Safe Mode, etc.).
- Drops per-attempt logs under ~/StrangeLab/scans/rtsp-hard-probe/-/.

5) BLE decoding upgrades + dataset situation

- Fingerprint fields captured: local name, connectable, TX power, service UUIDs, manufacturer company ID (little-endian), manufacturer prefix hex, beacon hints (iBeacon / Eddystone), RSSI + smoothed RSSI + trends.
- Company/vendor mapping and 'Assigned Numbers' mapping are now loaded from bundled text resources so decoding works offline.
- We agreed on a readable-layer upgrade: show 16-bit / 32-bit UUIDs alongside normalized 128-bit base form:
  • XXXX → 0000XXXX-0000-1000-8000-00805F9B34FB
  • XXXXXXXX → XXXXXXXX-0000-1000-8000-00805F9B34FB

  Important gotcha you hit:
- The repo currently contains *tiny* sample tables (11/21/13 lines). They must be replaced with the full Bluetooth SIG Assigned Numbers + Company Identifiers datasets for real decoding coverage.
- Your attempted cp failed because the big files were not in ~/Downloads. You then confirmed the repo paths exist under /Users/letsdev/CamUtil/CamUtil/Resources/.

  Quick sanity checks for 'full dataset installed':

```
wc -l CamUtil/CamUtil/Resources/BLECompanyIDs.txt \
CamUtil/CamUtil/Resources/BLEAssignedNumbers.txt \
CamUtil/CamUtil/Resources/BLEServiceUUIDs.txt
# full files should be thousands of lines, not tens
```

6) Pi■Aux event intake API (local-only)

Goal: pi-aux collects extra sensor data/dongles and ships *events* to the Mac UI so you can correlate BLE/Wi■Fi/network observations.

Local listener (as implemented in our plan):

- Bound to 127.0.0.1: on the Mac (or configurable if you later allow LAN binding).
- Token header required: X-CamUtil-Token: .
- Endpoint: POST /api/v1/events with JSON body.

Proposed minimal event schema (stable, append-friendly):

```
{
"schema": "strangelab.event.v1",
"ts": "2026-02-01T03:49:18Z",
"source": "pi-aux",
"type": "ble.advertisement",
"device_id": "ble:fingerprint:abcd1234ef",
"severity": "info",
"data": {
"rssi": -43,
"company_id": 76,
"service_uuids": [
"0000180F-0000-1000-8000-00805F9B34FB"
]
}
}
```

7) Troubleshooting snippets that actually mattered

Find the built .app location from build settings:

```
xcodebuild -project /Users/letsdev/CamUtil/CamUtil.xcodeproj \
-scheme CamUtil -configuration Debug -showBuildSettings | \
grep -E 'BUILT_PRODUCTS_DIR|TARGET_BUILD_DIR|FULL_PRODUCT_NAME'
```

Verify Bluetooth usage strings in the built app Info.plist:

```
/usr/libexec/PlistBuddy -c "Print :NSBluetoothAlwaysUsageDescription" \
~/Library/Developer/Xcode/DerivedData/CamUtil-*/Build/Products/Debug/CamUtil.app/Contents/In
fo.plist
```

Common zsh pitfall: 'no matches found' happens when a glob matches nothing. Wrap in quotes or use a variable with ls 2>/dev/null:

```
LATEST=$(ls -1t ~/StrangeLab/scans/logs-raw/CamUtil-LogsRaw-*.txt 2>/dev/null | head -n 1)
echo "$LATEST"
[ -n "$LATEST" ] && tail -n 80 "$LATEST"
```

8) Current known pain points / next fixes

- Window resizing: content disappearing when shrinking → needs minimum window size + scroll views for panes.
- Button density: cramped controls → move actions into a popover/popup action palette, keep the main pane clean.
- RTSP playback failures: can be URL path/auth/codec; Hard Probe should always generate per-attempt logs so you can debug without guessing.
- BLE dataset replacement: install full Bluetooth SIG datasets to eliminate number soup in reports.

This PDF is meant to be a 'where we are' anchor; it does not replace your source repo history, but it’s designed for human navigation.