Raqib – An Automated IoT Security Assessment and NCA Compliance Platform

Nasser Alsulami[1*], Mohammed Abdulmajeed Alrayiqi[1], Osama Abdulhamid Aljaghthami[1], Mohamad Hamed Talagi[1], Muhammad Shahbaz Khan[2]

[1*]Cybersecurity Specialization, KAUST Academy, Kingdom of Saudi Arabia.

[2] Aston University, Birmingham, UK.

*Corresponding author(s). E-mail(s): nasser_zead22@outlook.com;
Contributing authors: jusstm7md@gmail.com; osamhj7@gmail.com;
mohamad.talagi@outlook.com; engr.msk90@gmail.com;

Abstract

We present **Raqib**, an automated IoT security assessment and NCA compliance platform for the Saudi National Cybersecurity Authority's Cybersecurity Guidelines for Internet of Things (CGIoT-1:2024). Manual assessment is slow and difficult to scale because IoT devices are closed appliances, expose heterogeneous protocol surfaces, and are governed by requirements written at different levels of abstraction. Raqib implements nine of the ten stages of the IoTGuard reference pipeline in a simulated laboratory of eight device fixtures. It brings **76 of 81 guidelines** under a guided workflow: **17** use machine-collected evidence from **32 collectors** and **50 mapping rules**, while **59** use human-answered checklists; the remaining five are reported explicitly. The system is verified by **1,433 automated test definitions** and an adversarial source audit. Separately, we classified all 81 guidelines under a fixed network-and-firmware observability model: **23 of 81 (28.4%)** contain evidence a scanner can observe, **58 (71.6%)** do not, and **none** meets the strict criterion for full automation requiring evidence sufficient to establish both compliance and non-compliance. Of the 23 reachable guidelines, 21 lie in one domain. The key result is that workflow coverage, machine-evidence coverage, and framework reachability are different quantities; any automation percentage is meaningful only when it states which quantity it measures.

Keywords: IoT security, Saudi NCA CGIoT-1:2024, OWASP IoT Top 10, compliance auditing, evidence-based assessment, policy-as-code, security automation.

1 Introduction

1.1 Motivation

Internet-connected devices now sit inside hospitals, factories, utilities and homes across the Kingdom, and they are governed accordingly. In 2024 the National Cybersecurity Authority published its Cybersecurity Guidelines for Internet of Things, CGIoT-1:2024 [1], a public document of 81 guidelines across four domains that sets out what an organisation deploying, operating or procuring IoT devices is expected to do.

Assessing an estate against those guidelines is, today, largely a manual exercise. An auditor works through the framework one requirement at a time, per device, recording what they found and what they concluded. That is slow, it does not scale past a handful of devices, and — more importantly — it is inconsistent: two auditors reading the same guideline against the same device can reasonably reach different positions, and nothing in the process makes that visible.

Automating the work is harder than it first appears, for three reasons that are specific to IoT rather than general to compliance.

- **No agent, no host access.** A conventional compliance scanner installs software on the target. An IoT device is a closed appliance: the assessor is outside it, working from network traffic and whatever firmware image can be obtained. Anything that lives only on the device's own filesystem is out of reach.
- **Heterogeneity.** A camera, a lock, an industrial gateway and a network video recorder share almost no protocol surface. A check that identifies one says nothing about another.
- **Uneven abstraction.** Framework requirements are not written at one level. Some name a specific technical condition — do not ship default credentials. Others describe an organisational obligation — define, document and approve an IoT security policy. A tool that treats those as the same kind of thing will either fail on the second or, worse, report a pass it never checked.

1.2 Problem statement

This project addresses two questions that are usually conflated.

The first is an engineering question: *how much of CGIoT-1:2024 can be assessed automatically, and what does a platform that does so look like?* That is the system described in this report.

The second is a question about the framework itself: *how much of it contains evidence a network-and-firmware scanner could observe at all, regardless of what we happened to build?* The two are different. The first is a fact about our engineering and rises as we write more collectors. The second is a fact about the framework read through a fixed observability model, and does not move as the software changes. Reporting the first while implying the second is the failure mode this project set out to avoid.

1.2.1 Four quantities that are easy to confuse

Four numbers recur throughout this report. They measure different things, and the distinction between them is the report's central claim, so they are defined in Table 1 and used consistently thereafter.

Table 1 The four coverage quantities, and what each does *not* mean.

Value	Quantity	What it means
76 of 81	Workflow coverage	Guidelines the platform brings under a guided workflow, by either mechanism. Not a measure of automation: most are human-answered.
17 of 81	Machine evidence	Guidelines for which the platform collects evidence mechanically. The only figure that describes what the machine does.
23 of 81	Reachability	Guidelines containing evidence a scanner could observe under the model of Section 3.16. A property of the framework under that model, not of this implementation.
0 of 81	Full automation	Guidelines whose evidence settles the requirement in both directions — the strictest of the four, and a subset of the reachable 23 rather than a separate set.

Guided coverage therefore always means the first of these, *machine evidence* the second, and *reachable* the third. Where this report calls a guideline *observable*, it means a deterministic data point can be extracted from the device, its firmware image or its network traffic without human interpretation.

1.3 Scope and assumptions

Four boundaries are fixed and should be read into everything that follows.

- **The vantage point is a network and firmware scanner.** Management-plane sources — cloud consoles, fleet APIs, mobile device management platforms — are out of scope, even though a technically capable tool could query them, because an external assessor auditing a deployed estate is rarely granted that access.

- **The laboratory is simulated.** Eight containerised device fixtures stand in for physical hardware. This is a real limit on what the results can claim and is discussed in Section 6.
- **CGIoT-1:2024 is guidance, not a certifiable standard.** Any assessment against it is a statement about alignment. Nothing this platform produces is an NCA certification.
- **Nothing automated is authoritative.** Every judgment the system makes is a suggestion; every recorded status carries the name and role of the person who signed it.

1.4 The proposed system

Raqib implements nine of the ten stages of the IoTGuard reference pipeline, taking an unknown network through discovery, fingerprinting, compliance evaluation, vulnerability intelligence, risk scoring, AI-assisted remediation and an executive summary. Thirty-two automated collectors gather evidence; a policy engine turns that evidence into suggested findings; a human signs them.

The design decision that carries the most weight is the one that follows directly from the abstraction problem above. The compliance module does not attempt to automate the whole framework. It covers 17 guidelines with real evidence mappings and a further 59 with hand-authored checklist questions — 76 of 81 under a guided workflow — and names the remaining five openly rather than absorbing them into a percentage. Which guidelines fall into which category is decided by scope: the 59 are precisely those the platform's catalogue records as addressed to something other than a device, and the 17 are the device-scope guidelines a collector actually reaches. Five further device-scope guidelines are reached by neither mechanism and are named in Section 4. That is a different instrument from the classification procedure described in Section 3.16, and the two do not agree everywhere — where they diverge, Section 4 reports it rather than reconciling it silently.

1.5 Use cases

Five scenarios motivated the design and are used throughout this report to test whether a decision was the right one. All are situations a Saudi organisation deploying IoT already faces, and none requires the platform to do anything it cannot do — a constraint that ruled out several otherwise attractive scenarios, discussed at the end of this subsection. The first two are developed in full because they exercise the system most completely; the remainder are stated briefly.

1.5.1 Use case 1 — Deciding whether to accept a delivery

A hospital has ordered 200 network cameras from a vendor whose datasheet claims compliance with national IoT security guidance. The devices are in the loading bay. Nobody has verified the claim, and once they are racked and cabled the cost of discovering a problem rises sharply.

The security team registers a sample and triggers a single automated run. The platform discovers what is actually on the segment, fingerprints each device, uploads

and analyses the firmware image for embedded secrets and known vulnerable components, evaluates the device-scope guidelines against the collected evidence, and ranks the sample by risk. Where a device fails, a remediation blueprint is generated and held for human review.

What the team gets is not a certificate. It is a per-device position with the evidence attached: which guidelines this hardware violates as shipped, which findings have a recorded real-world exploit, and which of the vendor's claims the evidence does not support. That is a procurement conversation with a supplier, held before deployment rather than after an incident.

The boundary matters and is stated plainly: the platform assesses the devices, not the vendor's engineering processes. It can show that a shipped image contains a hard-coded credential. It cannot show whether the vendor has a secure development lifecycle — that is a question for Domain 4, and it is answered by a person.

1.5.2 Use case 2 — Preparing for a regulatory assessment

An organisation with an IoT estate expects to be assessed against CGIoT-1:2024. It has 81 guidelines, several hundred devices, and no current picture of where it stands. The practical question is not whether it is compliant — it is where the gaps are, and how much human effort closing them will take.

Here the split between the two coverage mechanisms does something more useful than a coverage percentage suggests. Fifty-nine of the guidelines are not addressed to a device at all — they place an obligation on the organisation, a supplier, a cloud provider or a companion mobile application: approving a policy, training staff, reviewing something on a schedule. Those are answered once, through the checklist mechanism, and the answer holds for the whole estate. The seventeen guidelines carrying automated evidence are evaluated per device, by machine.

> **Key insight**
>
> The human effort is fixed; the machine effort scales. Those 59 guidelines carry 148 checklist questions between them, and an auditor answers that set once for the whole estate. Meanwhile, for 200 devices, the platform performs in the order of 17×200 device-level guideline assessments without further input. Doubling the fleet does not add a single question.

That asymmetry is the practical payoff of having judged the ceiling explicitly rather than guessed at it. The organisation can budget human auditor time against a number that does not grow with the estate, and machine time against one that does. Five guidelines fall outside both mechanisms and are named explicitly, so they can be planned for rather than discovered during the assessment.

The boundary again: CGIoT-1:2024 is guidance, not a certifiable standard, and nothing the platform produces is an NCA certification. What it produces is a defensible internal position, with every judgment traceable to either a piece of machine evidence or a named person who signed it.

1.5.3 Use case 3 — Assessing a supplier before signing

Third-party and cloud obligations occupy a whole domain of the framework, and they are the part an organisation is least able to verify for itself. Before contracting, a procurement team can assess a candidate supplier's device with the same collectors used on its own estate, while answering the supplier-scope and cloud-scope checklists about the contract and the provider's stated practices.

The value here is the split, not the automation. The device evidence is objective and can be put in front of a vendor. The contractual questions cannot be automated at all and are recorded as answers attributed to the person who gave them. Presenting both together is a materially stronger negotiating position than a datasheet review.

1.5.4 Use case 4 — Prioritising work for a small team

Two engineers responsible for forty devices do not need a compliance report; they need to know what to fix on Monday. The platform ranks the estate by a six-factor risk score in which a device with no assessment at all scores the compliance factor at maximum risk rather than neutral, generates a remediation blueprint for each failing or partial finding, and produces a fleet-level summary that names every immediate-priority item still awaiting review.

This is the scenario the AI-assisted stage exists for, and also the one where its constraint matters most: a blueprint is a draft for a human to approve, and nothing it says changes a compliance status. A small team gets a prioritised queue, not an autopilot.

1.5.5 Use case 5 — Estimating what an assessment will cost

Before committing to a framework programme, an organisation needs to know what it is signing up for. The classification in Section 3.16 answers that directly: it names which guidelines can be reached by machine, which need a person, and — crucially — how that splits across domains, since the distribution is highly uneven and an average would mislead.

This use case consumes the research rather than the platform, and it is the one that generalises furthest: the procedure can be applied to any framework, by anyone, without access to this system.

1.5.6 What the platform cannot support

Three scenarios were considered and rejected because the system cannot honestly serve them, and naming them here is more useful than discovering them later. *Continuous compliance drift monitoring* requires Stage 10, which is not built. *Live incident-response triage* requires a real-time posture the platform does not maintain — every assessment is a snapshot. *Due diligence over an acquired estate* requires operating on a network the platform cannot reach and an identity model it does not have. Each is revisited in Section 6 with the specific gap that blocks it.

1.6 Project objectives

The project objectives are stated explicitly here so that the engineering work and the research question can be evaluated against the same set of targets. They are intentionally bounded by the assumptions in Section 1.3: the prototype is evaluated in a simulated laboratory, CGIoT-1:2024 is guidance rather than a certification standard, and automated outputs are evidence-backed suggestions rather than authoritative compliance decisions.

O1. Implement an end-to-end IoT security assessment pipeline. Build the stages needed to discover and fingerprint devices, collect security evidence, evaluate NCA guidance, enrich findings with vulnerability intelligence, prioritise risk, support remediation, and present results to an auditor, while reporting any unbuilt reference-pipeline stages explicitly.

O2. Operationalise CGIoT-1:2024 without overstating automation. Bring as much of the 81-guideline framework as possible into a guided workflow by combining machine-collected evidence for device-observable conditions with human-answered checklists for organisational, supplier, cloud and other non-observable obligations.

O3. Preserve evidence traceability and human accountability. Separate observation from compliance judgment, retain the evidence that produced every suggestion, require attestation for recorded assessments, and preserve superseded records so that the history of a decision can be audited.

O4. Measure coverage and automatability as different quantities. Quantify workflow coverage, machine-evidence coverage, framework reachability under a fixed network-and-firmware observability model, and the stricter full-automation ceiling, rather than collapsing them into one percentage.

O5. Validate the prototype reproducibly. Exercise the platform against a controlled laboratory containing multiple device types and security postures, verify the collectors and application logic with automated tests, and use adversarial review to expose gaps that ordinary feature testing may miss.

O6. Relate the system to recognised IoT security baselines. Map CGIoT-1:2024 formally to the OWASP IoT Top 10 and evaluate an informational post-quantum-readiness stage against the standardised NIST post-quantum cryptographic algorithms, without presenting either comparison as an additional compliance certification.

O7. Document limitations and concrete next steps. Identify where the current prototype, laboratory, evidence model and classification procedure cannot support a stronger claim, and pair each material limitation with a specific improvement or validation activity.

The Results section returns to these objectives in an objective-completion matrix so that completion is judged from reported evidence rather than from implementation intent alone.

Takeaway

The contributions of this project are: (i) a working platform covering nine of ten pipeline stages against a national IoT framework; (ii) a dual-mechanism compliance module combining 50 automated evidence mappings with 59 authored checklists; (iii) a written, reusable procedure for judging how much of a framework is automatable at all, and the resulting classification of CGIoT-1:2024; and (iv) an adversarial audit of our own codebase, whose findings are reported here rather than quietly fixed.

2 Related Work

The closest literature falls into four immediate bodies of work; the expanded review below adds security baselines, device identification, firmware analysis and empirical IoT studies.

2.1 Automated IoT compliance auditing

The closest work builds systems that assess IoT estates continuously. Briliyant et al. [2] systematise human-centred continuous auditing for IoT security compliance, and are explicit that a human stays in the loop — a position this project shares and implements. Oranekwu et al. [3] take the opposite emphasis, using ontology-driven reasoning to push real-time assessment as far towards automation as the representation allows.

Both report what their system covers. Neither asks what the framework itself permits. That distinction matters because a coverage figure and a ceiling figure look identical on the page and mean entirely different things: the first is a statement about an implementation, the second about a document.

2.2 Policy compliance assessment against control standards

PROPARAG [4] automates the assessment of cybersecurity policy documents against security control standards. The unit of analysis is the distinction: it evaluates *written policy* against a standard, whereas this project evaluates *running devices*. The two are complementary — a policy can be perfect while every device violates it — and a complete compliance picture arguably needs both. Notably, the requirements PROPARAG can assess are close to those this project classifies as unreachable by a scanner, which is a useful corroboration rather than a competition.

2.3 Framework-to-framework mapping

A small literature works through a framework requirement by requirement, which is methodologically the nearest thing to what we did. Cindrić et al. [5] map industrial IoT onto the IEC 62443 standards, and Alsahafi et al. [6] map compliance with the Saudi NCA Essential Cybersecurity Controls onto ISO/IEC 27001.

Both are exhaustive traversals of a framework, and both are cited here for what they demonstrate rather than what they conclude: that requirement-by-requirement

analysis is an accepted method. Neither classifies by automatability, so neither answers the question this report asks of a framework: how much of it a tool could settle without a person.

2.4 Machine-readable control catalogues

NIST's OSCAL [7] provides a standard format for expressing control catalogues, baselines, implementation statements and assessment results in machine-readable form, and is used in FedRAMP and CIS Controls work. It is reasonable to ask whether this already solves the problem.

It does not, and the distinction is worth stating precisely. OSCAL is a *representation* format: it makes a control machine-readable. It does not make it machine-*decidable*, and it takes no position on which controls could be assessed automatically. Encoding "the organisation shall approve an IoT security policy" in OSCAL produces a well-structured record of an unobservable requirement. The question asked here is orthogonal to the encoding.

2.5 Security baselines and authoritative IoT guidance

The broader IoT-security literature reinforces the separation in this report between a control catalogue, an implementation and evidence that can actually be observed. NIST's IoT device cybersecurity capability baseline identifies a small set of device capabilities—including device identification, configuration, data protection, logical access to interfaces, software update and cybersecurity-state awareness—that organisations can use when defining device requirements [11]. NIST SP 800-213 then turns those capabilities into federal acquisition and deployment guidance [12], while NIST IR 8259B and the later IR 8259 revision extend the baseline with non-technical manufacturer activities and lifecycle expectations [13,17]. SP 800-213A supplies a catalogue of IoT device cybersecurity requirements that can be selected according to risk [18]. These publications are important comparators because they describe what secure IoT devices and their supporting processes should provide; they do not claim that an external scanner can establish every requirement.

The post-quantum stage follows the same discipline. NIST standardised ML-KEM for key establishment in FIPS 203, ML-DSA in FIPS 204 and SLH-DSA in FIPS 205 [14–16]. Raqib therefore treats post-quantum readiness as an informational technical assessment grounded in named standards, not as a new compliance framework. The distinction matters: a device can reveal cryptographic negotiation and library evidence, but those observations do not by themselves demonstrate an organisation-wide migration programme.

Survey literature gives the wider threat context. Roman et al. [19] frame IoT security around distributed, resource-constrained and heterogeneous systems; Sicari et al. [20] organise the field around security, privacy and trust; Mosenia and Jha [21] catalogue threats across IoT layers and applications; and Khan and Salah [22] review security challenges and later trust architectures. Collectively, these surveys explain why a single host-style compliance agent is a poor fit for IoT, but they do not provide a method for distinguishing framework requirements that are externally observable from those that remain documentary or organisational.

2.6 Network identification and intrusion detection

A second research stream asks what can be learned from network-visible behaviour. Zarpelão et al. [23] survey IoT intrusion-detection approaches and show that protocol and traffic features are a common source of security signal. Hajiheidari et al. [24] similarly review network-based intrusion detection for IoT and emphasise the diversity of attack surfaces and datasets. This work supports Raqib's use of network evidence, while also illustrating why network evidence should not be confused with an organisation's full compliance posture.

Device-identification systems are especially close to Raqib's discovery and fingerprinting stages. IoT Sentinel identifies device types from network communication so that enforcement can be applied without installing software on the endpoint [25]. Meidan et al. demonstrate machine-learning identification of unauthorised IoT devices from network traffic [26]. N-BaIoT uses deep autoencoders to identify botnet behaviour from device traffic [27], and Doshi et al. show that lightweight traffic features can detect consumer-IoT DDoS activity [28]. IoT Inspector scales the collection of labelled

smart-home traffic by crowdsourcing observations from real deployments [29]. These systems demonstrate that useful device identity and security evidence can be obtained externally. Their output, however, is a technical classification or anomaly score; none settles requirements such as whether a policy was approved, staff were trained, a contract contains a required clause, or a periodic review occurred.

2.7 Firmware and embedded-system analysis

Firmware analysis provides the second major evidence source in Raqib's observability model. Costin et al. [30] performed a large-scale security analysis of embedded firmware and demonstrated that firmware images can expose credentials, keys, vulnerable components and other defects that are unavailable from a simple port scan. Firmadyne automates dynamic analysis of Linux-based embedded firmware through emulation [31], and FirmAE improves the scalability and success rate of firmware emulation for IoT analysis [32]. These systems motivate the use of offline firmware evidence and show that static and dynamic firmware analysis can substantially widen what a scanner can observe.

They also illustrate the boundary adopted in this report. Even perfect firmware analysis observes what is present in an image; it does not prove that an organisation has approved a policy, maintained an asset register, reviewed supplier risk or carried out a scheduled governance activity. Firmware analysis therefore increases technical reach without making a mixed technical-and-organisational framework fully machine-decidable.

2.8 Empirical smart-home and ecosystem security

Empirical studies show how the same weakness class can span device, network and ecosystem layers. Fernandes et al. [33] analyse smart-home applications and demonstrate that platform permission and application-design choices can create security risks outside the embedded device itself. Ronen et al. [34] show how vulnerabilities in connected-lighting ecosystems can propagate through ZigBee networks. Antonakakis et al. [35] document the Mirai botnet and the operational consequences of weakly protected IoT devices at Internet scale. Alrawi et al. [36] systematically evaluate home-based IoT deployments and expose security failures across devices, applications, cloud services and local networks.

These studies are directly relevant to the scope choice in Section 3.16. They show that important security properties live outside a network-and-firmware scanner's vantage point. A cloud API, mobile application, vendor process or user's deployment practice may determine whether a system is secure even when the device-facing evidence looks clean. Raqib therefore treats management-plane and organisational evidence as a different evidence class rather than silently treating its absence as a pass.

2.9 Research gap

The expanded literature leads to the same gap as the narrower comparison above, but with a stronger basis. Existing work provides IoT security baselines [11–13,17,18],

network and device classifiers [23–29], firmware-analysis techniques [30–32], empirical ecosystem studies [33–36], automated compliance systems [2,3], policy-document assessment [4], framework mappings [5,6] and machine-readable control representations [7]. These works establish that many useful observations can be automated and that controls can be represented systematically.

What they do not provide is a requirement-by-requirement estimate of how much of a named security framework an external network-and-firmware scanner could settle *in principle*, independently of the implementation coverage of one tool. This report therefore keeps two measurements separate: what Raqib currently implements, and what CGIoT-1:2024 contains that is observable under a fixed evidence model. The search reported here is a targeted literature review rather than a systematic review; the claim is accordingly limited to the works reviewed, not to an assertion that no such method exists anywhere in the literature.

3 Methodology

This section describes the system: how it is built, what each pipeline stage does, how the whole thing was verified, and—separately—the procedure used to measure how much of CGIoT-1:2024 is automatable at all. Figure 1 gives the shape of the whole system: the ten-stage pipeline above, and beneath it the evidence path that the compliance stage runs on. Before the implementation detail, the project requirements are formalised so that the Results section can evaluate the built system against explicit criteria.

3.1 Functional requirements

ID	Functional requirement	Verification evidence
FR1	Discover live hosts within an explicitly configured, validated audit scope and classify discovered hosts without assuming every live host is an IoT device.	Discovery collector; scope validation; laboratory branches for IoT, uncertain and unknown.
FR2	Fingerprint registered devices using service and protocol observations while preserving whether identity was asserted by a person or auto-detected.	Fingerprinting collectors; provenance flag; vendor/MAC consistency check.
FR3	Collect structured network, web/authentication, firmware and device-intelligence evidence without letting collectors issue compliance verdicts.	32-collector catalogue; structured observation schema.
FR4	Evaluate device-observable CGIoT-1:2024 conditions through reusable evidence-to-guideline mapping rules and a single evaluator.	50 mapping rules reaching 17 guidelines; evaluator unit tests.
FR5	Provide guided human checklists for non-device requirements that cannot be settled by scanner evidence.	59 checklist-covered guidelines; 148 authored questions.
FR6	Analyse firmware and device identity for vulnerabilities, while distinguishing unknown/unmatched from a clean result.	Grype/SBOM path, CISA KEV cross-reference, device-level vulnerability lookup.
FR7	Compute a transparent, prioritisation-oriented device risk score with a per-factor breakdown and no feedback into compliance verdicts.	Six-factor model and invariant tests.
FR8	Produce remediation blueprints for failing/partial findings but require human review before a blueprint is treated as reviewed.	Structured remediation schema and review gate.
FR9	Present device and fleet results through an executive summary and dashboard without recomputing independent versions of core metrics.	Shared evaluator/risk/report functions; dashboard routes.
FR10	Orchestrate discovery through compliance in a single automated run while retaining the distinction between system-generated and human-signed assessments.	Fully Automated Run; system-attestation flag; automatic passes withheld.
FR11	Preserve assessment history, evidence links and supersession so a later decision does not erase the earlier one.	Append-only evidence/verdict rows; supersession pointers; audit events.
FR12	Assess post-quantum readiness as an informational stage based on standardised algorithms, without changing NCA compliance or risk.	TLS and firmware PQ collectors; NIST FIPS 203–205 [14–16].

3.2 Non-functional requirements

ID	Non-functional requirement	Operational criterion / evidence
NFR1	**Assessment safety.** Scanning should minimise the risk of destabilising constrained IoT devices.	Conservative nmap timing, reduced version intensity and a 50-packet/s maximum rate.
NFR2	**Isolation.** Deliberately vulnerable fixtures must not have a direct path to persisted auditor state.	Separate audit/internal networks; only the worker is dual-homed and it holds no database credentials.
NFR3	**Input and command safety.** Target input must be constrained before any external command executes.	API and worker revalidation; argument-list execution; scope/loopback/internal-service rejection.
NFR4	**Traceability.** A reader must be able to recover what evidence supported a decision and what it superseded.	Evidence attachment, signer/role fields, append-only history and audit events.
NFR5	**Reproducibility.** The evaluated environment should be repeatable rather than dependent on undocumented workstation state.	Docker Compose laboratory and pinned tool/container dependencies.
NFR6	**Honest uncertainty.** Missing, unmatched or indeterminate evidence must not be rendered as a clean result.	Explicit unknown/unmatched states; null rather than fabricated vulnerability/PQ answers.
NFR7	**Human accountability.** An authoritative recorded compliance status should remain attributable to a named role, and an unattended run must not silently certify a pass.	Attestation fields; automatic fail/review system flag; automatic passes left for review.
NFR8	**Deterministic scan-time inputs.** External intelligence should not change unpredictably during one scan.	Locally cached NVD/KEV/vendor/Grype data; refresh occurs separately from scan execution.
NFR9	**Explainability.** Risk and compliance outputs should expose their basis rather than only a scalar score or verdict.	Per-factor risk breakdown; matched observation fields displayed for suggested passes.
NFR10	**Bounded claims.** Results must be reported as prototype/laboratory evidence, not production readiness or NCA certification.	Simulated fixtures, no production authentication, Stage 10 unbuilt; limitations reported explicitly.

Two design principles run through every stage and explain most of the choices that follow.

Key insight
Collectors observe; they never issue a compliance verdict. A scan test returns mechanical observations, and may attach an advisory note for a human reader, but the policy engine reads only the structured observation fields. Some

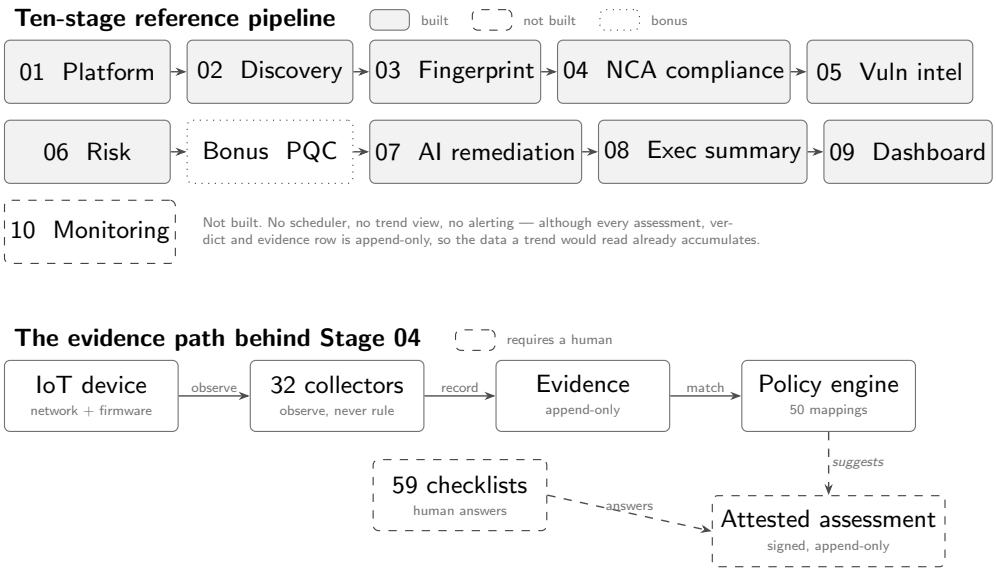


Fig. 1 System overview. **Top:** the ten-stage reference pipeline — nine stages implemented, plus a post-quantum bonus stage, with continuous monitoring unbuilt. Solid outlines mark built stages, dashed the unbuilt one, dotted the bonus stage; on the evidence path a dashed outline or arrow marks a step that requires a person. **Bottom:** the evidence path that Stage 04 runs on. Collectors may classify technically but never issue a compliance verdict; the policy engine only ever *suggests* a status — a fail, a review, or a pass, never a recorded one; and the fifty-nine guidelines no collector can reach enter through authored checklists rather than being omitted. Every recorded assessment carries the name and role of the person who signed it.

collectors do classify at the technical layer — discovery labels a host as IoT, uncertain or unknown; fingerprinting judges whether a claimed vendor matches a hardware address — and those are decisions. What no collector does is decide what an observation *means for compliance*: that meaning is supplied by a policy file or by a human, never by the collector.
Nothing inferred is ever recorded as a decision. Every automated judgment is a suggestion. Every status that carries weight was signed by a named person in a stated role.

3.1 System overview

Raqib runs as a Docker Compose stack. Four services form the auditor: a FastAPI backend, a worker, a PostgreSQL database and a React web client. The remainder of the stack is the simulated laboratory.

Two Docker networks separate the auditor's own backend traffic from the network being audited. This is not a diagram convention — it is enforced by Docker, and it is why the worker is the only component that can reach a device. The API never executes a command; it creates and reads `scan_jobs` rows, and the worker polls for them, resolves the target, re-validates it and runs the collector.

Schema is managed by seventeen numbered SQL migrations plus an `init.sql` that seeds a fresh volume. Migrations are idempotent and safe to re-run against a live database, because `init.sql` only executes on a first-time volume and an existing installation needs a separate path. Judgments are preserved rather than overwritten: compliance assessments and remediation blueprints each carry a `superseded_by` pointer, so a revision links to the row it replaces instead of destroying it, and verdicts and evidence are only ever inserted. The history of what was believed, and when, therefore survives by construction — with two exceptions, noted here rather than in a footnote. Compliance exceptions are mutated in place, since approval and rejection write the status onto the existing row; and a remediation blueprint's human review is likewise written onto the blueprint rather than superseding it. Both are human sign-offs, which makes them exactly the rows whose history the design elsewhere takes most care to preserve.

Sequential record identifiers are still derived by counting the rows already issued for the day, but that count now runs under a database advisory lock, which closes a race in which two concurrent requests could count the same total and mint the same identifier — a defect that would have corrupted the append-only history the rest of the design depends on.

One structural asymmetry caused a real incident and is worth recording. The API's Python source is baked into its image at build time, while the `policies/` tree is bind-mounted. A change to a policy file therefore takes effect on restart, but a change to an API route requires a full rebuild.

The full stack is seventeen containers, listed in Table 2; sixteen run by default, the certificate generator being a one-shot service started only when the laboratory is first initialised. The topology is easier to state than to draw: everything being audited sits on `audit-network` (172.30.0.0/24); the auditor's own state sits on `internal-network` (172.31.0.0/24), which Docker marks `internal` so it has no route off the host; and exactly one container, the worker, is attached to both. Nothing else crosses.

The asymmetry in that table is the design. The container that touches deliberately vulnerable devices holds no database credentials, and the container that owns every record cannot reach a device at all. A compromised fixture therefore cannot reach persisted evidence without first traversing a component that validates the request — and that component is not reachable from the audit network in the first place.

One operational consequence is worth recording because it misleads on first reading. The dashboard declares a published port, but a container whose only network is marked `internal` cannot be reached by Docker Desktop's port-forwarding proxy, so on macOS and Windows that publication silently never binds. A development overlay exists for exactly this reason: it relaxes the isolation and publishes the fixtures individually. It is never used in the base configuration, and the isolation the rest of this section relies on is the base configuration's.

3.2 The simulated laboratory

Evaluation inputs are containerised device fixtures rather than physical hardware. Three represent security postures — insecure, partially hardened and fully hardened — and provide the graded baseline against which a collector's discrimination can be

Table 2 Container inventory. *audit* denotes `audit-network` and *internal* denotes `internal-network`. No container is given a static address; Docker's embedded DNS resolves service names within each network.

Container	Network	Role
Device fixtures — the estate under audit		
`device-insecure`	audit	Camera, insecure profile
`device-partial`	audit	Camera, partially hardened
`device-hardened`	audit	Camera, hardened
`device-smartlock`	audit	Smart lock
`device-plc-gateway`	audit	Industrial gateway, Modbus
`device-router-gw`	audit	Router/gateway, UPnP
`device-nvr`	audit	Video recorder, RTSP
`device-speaker`	audit	Smart speaker, mDNS
Supporting infrastructure		
`telnet-sim`	audit	Telnet-only host for the discovery classifier
`mqtt-broker-insecure`	audit	Plaintext, anonymous broker
`mqtt-broker-secure`	audit	TLS and password-authenticated broker
`traffic-capture`	audit	Packet-capture sidecar
`cert-init`	—	One-shot certificate generator
The auditor		
`auditor-worker`	audit + internal	The only executor and the only container on both networks
`auditor-api`	internal	Owns all persisted state; executes nothing
`auditor-database`	internal	PostgreSQL
`auditor-web`	internal	Dashboard; reads from the API only

checked: a correct check should fail the first, partially pass the second and pass the third.

Five further fixtures provide protocol and domain variety rather than posture variety: a smart lock, an industrial Modbus gateway, a router/gateway exposing UPnP, a network video recorder speaking RTSP, and a smart speaker announcing over mDNS. Each presents a real manufacturer's vendor and model strings and a self-reported hardware address carrying that manufacturer's registered prefix, which is what makes the identity and vendor-consistency checks meaningful: a fixture claiming to be nothing in particular cannot exercise a check that looks up what it claims to be. The realism has a stated boundary — the address a fixture reports about itself carries the real vendor prefix, but the address network discovery actually observes on the wire is the virtual one Docker assigned, so discovery-time vendor enrichment sees nothing meaningful inside the laboratory.

Five supporting services complete the environment: two MQTT brokers (one open, one TLS-protected), a Telnet-only host that exists so the discovery classifier's `uncertain` branch can be exercised against something real, a traffic-capture sidecar and a certificate-initialisation service. One certificate is deliberately weak, to give the TLS checks something real to fail against.

Table 3 gives the configuration each fixture carries. The vendor and model strings are real products, and the two cameras that share a manufacturer carry different models rather than one identity between them. Seven of the eight were chosen because they carry a publicly documented vulnerability of the kind the fixture is built to exhibit, so a finding against a fixture corresponds to something that has actually happened to that class of device rather than to an invented flaw. The eighth, the hardened camera,

was chosen for the opposite property: a manufacturer publicly associated with signed firmware and HTTPS-only management, to give the graded comparison a realistic upper end rather than a merely configured one.

Table 3 Device fixtures and the configuration each carries. The three camera profiles are built from one image and differ almost entirely by environment; the hardened and partial profiles additionally mount the certificate directory, and each profile's healthcheck differs to match the transport it serves. That makes them a near-controlled comparison rather than a perfectly controlled one.

Fixture	Identity presented	Configuration and distinguishing exposure
Camera, insecure	Hikvision DS-2CD2143G2-I	Plaintext HTTP, default credentials, Telnet enabled, hard-coded API key, anonymous MQTT
Camera, partial	Hikvision DS-2CD2143G2-IU	HTTPS with a deliberately weak certificate (1024-bit, SHA-1); credentials changed but still weak; administrative endpoint still unauthenticated; MQTT still plaintext
Camera, hardened	Axis M3216-LVE	HTTPS with a strong certificate, unique credentials, authenticated MQTT over TLS
Smart lock	Yale Conexis L1	Default PIN; tamper detection present but not wired
Industrial gateway	Schneider Modicon M221	Unauthenticated Modbus TCP on port 502
Router/gateway	Netgear R7000	SSDP and UPnP exposure
Video recorder	Dahua NVR4108-8P	Unauthenticated RTSP stream. No public identifier exists for this product, so it is left out of the mapping table rather than given an invented one
Smart speaker	Sonos One (Gen 2)	mDNS announcement; voice log unencrypted at rest

3.3 Evidence collection

Thirty-two collectors gather every observation the platform produces. The catalogue groups them into the five categories below and tags a subset with a second axis, `pipeline_phase`, identifying which stage a test belongs to. The full catalogue is listed in Appendix B.

Almost every observation is a fact about a device's current runtime state or about the contents of an image it shipped with. The one exception is the device-intelligence collector, which attaches cached public advisory data to a device's declared identity and so reports a fact about what is published rather than about the device. That is the honest boundary of what a network and firmware scan reaches, and it is the boundary the procedure in Section 3.16 measures against.

3.4 Stage 02 — Network discovery

Discovery runs before device registration and answers the question the rest of the pipeline depends on. The sweep is deliberately split in two. Stage A runs a ping and ARP sweep only — no ports, no reverse DNS — across the configured scopes. Stage B then runs a service-version scan against only the addresses that answered, restricted to eight ports and accompanied by two nmap broadcast scripts for UPnP and DNS service discovery.

Table 4 Evidence collection surface. Collectors return structured observations; the policy engine reads only those fields, never a collector's prose.

Category	Count	Representative signal
Network and protocol	16	Open ports and service versions; Modbus, RTSP, UPnP and mDNS probes; TLS configuration; cleartext traffic capture; device identity and MAC vendor
Web and authentication	6	Default credentials, session handling, anonymous access
Firmware	8	Embedded secrets, private keys, component manifest
Network discovery	1	Whole-scope sweep and host classification
Device intelligence	1	Published vulnerabilities for a device's declared vendor and model, with no firmware image
Total	**32**	

The split matters at scale: a single-command sweep spends most of its time probing addresses that are not there. If Stage A finds nothing, Stage B is skipped and the scan still succeeds with an empty host list — recorded in the code as a real and honest outcome rather than an error.

Each live host is classified from its open ports. Six ports are treated as IoT signatures (80, 443, 1883, 8883, 502, 554) and two as ambiguous (22, 23), yielding `iot_device` at high confidence, `uncertain` at medium (Telnet) or low (SSH only), or `unknown`. The important property is that `unknown` is a reachable outcome rather than dead code: an earlier version passed `--open` to nmap, which silently omitted hosts with no matching port and made that branch unreachable.

Scan tuning is deliberately gentle — timing template 3, a maximum rate of 50 packets per second, reduced version-detection intensity. IoT network stacks are fragile, and an aggressive scan that knocks a device over makes the audit itself the incident.

3.4.1 Configurable scan scope

The audit range was originally a hardcoded constant in two independent modules. It is now a database-backed allowlist: a `network_scopes` table holding a label, a CIDR, whether it is active, who added it and why. Both consuming modules stay free of database and HTTP imports on purpose and are reconfigured in-process — the API directly after every write, the worker by polling a small endpoint every fifteen seconds, since it has no database access of its own. A failed poll retains the last known good configuration.

Creation is guarded: a new scope must parse strictly with no host bits set, be IPv4, be no larger than a `/16`, fall inside private or link-local address space, and not overlap the auditor's own backend network or loopback. The laboratory's range survives as a named preset that can be deactivated but never hard-deleted, so returning to lab-only scanning is one action away. Deactivation *is* the audit trail.

3.5 Stage 03 — Device fingerprinting

Fingerprinting establishes what a registered device is and what it exposes: a full-range service-version scan plus protocol-specific probes. Nine collectors carry the fingerprinting phase tag.

Two of them are recent and change what the later stages can do. The first reads a device's self-declared identity — vendor, model and firmware version — and the platform will adopt it automatically, but only under a rule worth stating: the fields are filled only when all three are empty and no person has set any of them, the row is locked while it happens, and the device is marked as auto-detected rather than manually entered. Editing any identity field by hand flips the record back to manual, and a record already marked auto-detected is never re-filled. The mechanism is a column on the device row holding one of two values, and it is deliberately excluded from the set of fields a client may patch: it is only ever set by the system, because a provenance flag a client can write is a provenance flag that can be made to lie. An identity the machine guessed and an identity a person asserted therefore remain distinguishable in the database, which is the same principle the attestation design follows.

The second resolves the vendor behind a device's MAC address and compares it with the vendor the device claims to be, so a device presenting itself as one manufacturer while carrying another's hardware address is visible. This is also the one place the platform departs from its own no-network-at-scan-time rule, and that departure is discussed in Section 6.

The most careful code in this stage is not the scanning but the decision about what may be scanned. A host is rejected if it parses as an address outside the configured scopes, is a loopback alias, looks like an address but is not a valid one, names one of the auditor's own services, or fails a strict name pattern.

One guard deserves specific mention. A bare integer or hexadecimal string contains no dot or colon, so it evades a textual address check, and it matches an ordinary hostname pattern — but the C library resolves it as an IPv4 address anyway, so nmap and curl will connect to it. Any value that parses as an integer in any base is therefore rejected, and the source carries an explicit warning not to delete the check as redundant. Separately, the name pattern's leading-alphanumeric requirement prevents a host value from being interpreted as a command-line flag.

Validation runs twice, independently: the API validates on write, and the worker re-validates the row it reads back before building any command, on the stated principle that the database is untrusted input. Commands are always constructed as argument lists rather than shell strings, so even a bypassed check leaves no shell-injection surface.

3.6 Stage 04 — NCA compliance assessment

This is the largest module and the one the project is named for.

3.6.1 The catalogue

The 81 guidelines are parsed deterministically from a page-by-page transcription of the published document [1] into a committed JSON catalogue — a reviewed artefact, not

something regenerated at request time. Four domains hold 27, 41, 2 and 11 guidelines across 27 subdomains.

Each guideline carries nineteen fields. The canonical requirement is the NCA's own wording, verbatim. Everything evaluative — scope, assessment type, severity, whether it blocks — is the project's own judgment and is labelled as such in the source. That separation matters: a reader must be able to tell the framework's text from our interpretation of it.

One defect in that catalogue was found while preparing this report and is recorded rather than quietly corrected, because it illustrates the limits of calling something a reviewed artefact. The parser has no terminator for the final guideline, so the last entry, 4-2-5, does not stop at the end of its requirement: it absorbs the remainder of the document — three appendices and the back cover — into its canonical text, which runs to some six thousand characters where every other entry is a sentence or a short list. No evaluation reads that field, so the entry's classification, mappings and score are unaffected. It is not harmless, though: the canonical requirement is displayed verbatim on the control page, in the assessment dialog and in the executive summary, and it is passed into the remediation prompt as the requirement text — so for this one guideline a reader, and a language model, are handed six thousand characters of appendix. But the field is not what it claims to be for one guideline in eighty-one, and a transcription nobody re-read end to end is a weaker artefact than the word "reviewed" implies.

3.6.2 The evaluator

Status and score come from one module of pure functions, so a route handler and a dashboard cannot disagree about a device's position. A verdict follows a fixed precedence: no applicable controls fails outright; a blocking failure fails; a scope in which no applicable required control has yet produced any result fails; a score below the partial threshold of 50 fails; critical failures, untested controls or controls awaiting review each reduce a pass to partial; a score below the pass threshold of 85 is partial; otherwise it passes.

Three guidelines are marked blocking — default or hard-coded credentials, unencrypted sensitive data in transit, and unnecessary exposed services. Any one failing fails the whole assessment regardless of score.

Two details are easy to get wrong and were handled deliberately. A device with nothing assessed returns `not_tested` from the status function and a score of `None`, not partial and not zero — a fresh device must not read as partially compliant on the strength of nothing. The readiness classification deliberately goes the other way for the same device and records it as failed, on the principle the risk model also applies: absence of proof of compliance is not proof of safety. The two are not in conflict, because they answer different questions — what is known about this device, and whether it may be signed off — but the difference is worth stating, since a device can read `not_tested` in one view and *failed* in another. And evidence expiry can only downgrade a pass to partial; it never improves a status and never touches a failure.

3.6.3 Two routes to a status

The module must cover a framework in which most requirements are not technical, and does so with two mechanisms sharing one evaluation vocabulary.

The **finding-mapping layer** turns observations into suggestions. Fifty rules key off generic observation field names rather than device identities and reach seventeen distinct guidelines, all device-scope and all in Domain 2. A guard ensures a rule fires only against evidence that actually carries the field it tests; without it, an absent field compares not-equal to almost anything and a mapping would fire on unrelated evidence.

Each rule carries a verdict hint, and the distribution is the part worth reading. Twenty-eight fire on an insecure condition and suggest a fail. Four surface evidence a human must interpret and suggest review. The remaining eighteen fire on a genuinely clean reading — TLS negotiated at a current version, MQTT carried over TLS, Telnet absent from the open-port set — and suggest a pass. That third category is a deliberate reversal: for most of the project's life the schema would not admit it, because every rule fired on a defect and a match could only ever mean bad news. The consequence was asymmetric in a way that showed up in use. A device whose evidence was genuinely clean sat at *not tested* indefinitely, because nothing in the system could represent the observation that a check had run and found nothing wrong — which is a different statement from a pass, and the platform had no way to say it. Where several rules match one guideline, precedence resolves fail above review above pass: one real problem outweighs any number of clean checks.

Admitting a suggested pass introduces a weakness, and the design bounds it rather than denying it. A guideline with several independent aspects can be suggested as a pass on the strength of one clean reading while the others were never exercised; guideline 2-4-3 carries six independent fail-rules, so a scan touching only one of the two aspects its pass-rules cover would still produce a suggestion. Every mapping-derived suggestion therefore carries the set of observation fields that actually produced it, and the signing dialogue displays that set whenever such a suggestion is a pass, so the auditor confirming one sees whether it rested on a single check or on several. This does not remove the weakness; it makes it visible at the moment it matters, which is the moment a person puts their name to it. The distinction the whole module rests on holds throughout: a suggestion is not an assessment, and only a person records one.

The **checklist mechanism** covers the other fifty-nine guidelines — precisely the non-device set, with no overlap with the seventeen. Each is a small set of questions derived from the guideline's own text, built from three reusable templates that capture the framework's recurring phrasings: define–approve–implement, periodic review, and practice-with-evidence. Answers are evaluated top to bottom and the first matching rule wins; if nothing matches, the result is nothing at all, never an implicit pass.

No mapping targets the governance domain or the supplier and cloud groups, and an invariant test enforces it. A scan cannot demonstrate that a policy was approved or that a contract says what it should.

3.6.4 Attestation and sign-off

Recording an assessment requires a control, exactly one scope, an applicability with a reason where something is excluded, a status, a severity, a test method and a named assessor. Where the status is anything other than not-tested, three further fields are mandatory: confirmation, the certification text displayed at signing, and the role the signer acted in. A database check constraint enforces part of that rule — a status other than not-tested cannot be stored without the confirmation flag — so an entirely unattested judgment cannot exist even if the API is bypassed. The role and the certification text are not themselves constrained at the schema level, so a direct write could still record a confirmed assessment with neither. The API is what makes all three mandatory, and the schema is the weaker of the two guarantees.

An assessment's judgment is never updated in place. A new row is inserted, the previous row's supersede pointer is set, and an audit event records the before and after values with an actor and a reason. The only fields ever written onto an existing assessment row are that supersede pointer and the list of attached evidence records, which is itself append-only. Exceptions require a reason, a requester and a mandatory expiry, and count only while approved and unexpired. Override is the most guarded write path: it requires a justification and a named person, re-runs full validation, records a distinct audit event, never mutates the original row, and offers an optional concurrency guard that rejects the request if the row has changed since the caller loaded it.

3.7 Stage 05 — Vulnerability intelligence

Firmware arrives as an uploaded archive and is analysed without ever being extracted to disk. Uploads are capped at 20 MB and rejected if any member name is absolute or contains a parent-directory traversal. Two checks govern the type, and the order matters: the API rejects anything whose filename does not end in a recognised archive extension, and the worker then dispatches on magic bytes rather than on that extension, so a renamed archive is refused at the door but never mis-parsed once inside. Members are read into memory under a per-member cap and a total budget, with the running total breaking the loop rather than skipping — the difference between a bounded read and a zip bomb.

Eight checks run against the archive: three YARA rules (hard-coded passwords, embedded API keys, private key material) and five structural checks, one of which is the post-quantum inventory described later in this section. The component manifest is converted to a CycloneDX SBOM and scanned by Grype [8], pinned to a fixed version and installed in the worker image. The scan is entirely offline; the vulnerability database is refreshed by a separate scheduled operation and persisted in a named volume.

Findings are cross-referenced against a locally cached CISA Known Exploited Vulnerabilities feed [9], written atomically so a partial file can never be read. Where exploitation status changes anything, it changes ordering: findings sort by exploitation record first and severity second. It does not flip a verdict.

A second, independent path reaches vulnerabilities without any firmware image at all. Where a device's vendor and model are known, they are mapped through a small hand-verified table of product identifiers and matched against a locally cached extract of the National Vulnerability Database, refreshed on the same scheduled, never-at-scan-time basis as the Grype database. This matters because a firmware image is the hardest evidence to obtain in practice — an auditor assessing a deployed estate frequently cannot get one — so a device-level path widens what is assessable considerably.

Two honesty properties carry over. A device whose identifier is not in the table is reported as unmatched rather than as clean, and the interface renders that as its own state rather than as an empty result. The laboratory contains one such device by circumstance rather than design: the national vulnerability data holds no identifier for that recorder at all, and rather than invent a plausible-looking one the team left it unmapped — which has the useful side effect of keeping the unmatched path visible in the product itself. And device-level findings are excluded from the risk score, which reads package-level evidence only — so this path informs a reader without silently moving a number it was never validated against.

Built on top of that path is a version comparison that answers a sharper question than *does a vulnerability exist for this product*. A device reports its own firmware version as a vendor-flavoured string, and a comparator extracts the dotted-numeric core from those shapes and tests it against each advisory's affected range, returning one of four states per vulnerability: affected, not affected, affected with no fix available, or unknown. The third state is the one that earns its place. It means the advisory's range has no upper bound, so the finding is real but no upgrade resolves it — a distinction an operator needs and that a single *outdated* flag erases. It should be read as a property of the advisory data rather than as a vendor end-of-support declaration, which the platform has no source for and does not claim.

One vendor feed sits alongside the national database for the same reason. The industrial gateway fixture's manufacturer publishes machine-readable advisories in a standard format that the national database does not fully carry, so those are fetched on the same scheduled, offline-at-scan-time basis and consulted as a second and more authoritative opinion for the products they cover. The feed currently carries one hand-verified advisory, which is stated plainly rather than described as vendor integration.

Beneath Grype sits a small static reference table scoped by its own documentation to the packages this laboratory's firmware ships. Its honesty rules are the notable part: an unknown component returns null rather than a guessed clean result, and the exploitation field is null on that path because the static table is not cross-referenced at all — never a guessed false.

3.8 Stage 06 — Dynamic risk assessment

Risk scoring answers a different question from compliance: not whether a device meets a requirement, but which device to deal with first. The direction is inverted from every other number in the platform — higher means riskier.

The model carried a seventh factor until recently, a violation count, and its removal is worth a sentence because the reasoning generalises. That factor counted failing

Table 5 The six risk factors. Weights are asserted at import time to sum to one, and every score returns a full per-factor breakdown, so the number is never a black box.

Factor	Weight	Derivation
Compliance	0.30	100− NCA score; never-assessed scores the maximum
Highest CVSS	0.20	Severity score × 10, from firmware evidence
Exploit availability	0.20	100 if any CVE is on the CISA list, else 0
Device criticality	0.15	Auditor-set, four levels
Network exposure	0.10	Auditor-set, three levels
Insecure services	0.05	25 points per enabled insecure service, capped

verdicts from an earlier five-control policy-as-code pilot together with failing NCA assessments, which made it a genuinely independent signal while both frameworks were live. When the pilot was retired as a pipeline stage the factor would have counted only NCA failures — the same failures the compliance factor already reflects — so it was removed rather than left to double-count one framework under two names. Its weight was folded into compliance rather than redistributed, which is why that factor now carries 0.30 and not 0.25: a deliberate increase in what a compliance position contributes to risk, not an accounting artefact.

The never-assessed rule is argued explicitly in the source: absence of proof of compliance is not proof of safety, so a device nobody has examined scores the same on that factor as one that failed everything.

Two inputs cannot be determined by any scan and are set by the auditor. Criticality defaults to high where the device runs a broker and medium otherwise, and is never guessed down to low. Exposure is always set to internal-only at registration and deliberately not inferred from a published port, since a published port reflects a developer's convenience rather than real internet reachability.

Nothing is persisted or cached; the score is recomputed on every read. Crucially, a risk score never feeds back into a compliance verdict, never auto-fails a control and never blocks a passing classification.

3.9 Stage 07 — AI-assisted remediation

This is the one stage that calls a language model, and the design question was how to use one without letting it decide anything. The answer has three parts.

The **prompt** states that the finding has already been determined by deterministic scanning and policy evaluation, that it is a fact rather than something to question, and that no additional vulnerabilities or facts may be invented. The **response schema** constrains output to five required fields: root cause, ordered remediation steps, priority, estimated effort and caveats. The **human gate** requires a named reviewer before a blueprint counts as reviewed.

The engine never raises. Every failure path — missing key, network error, unparseable body, malformed structure, missing field, invalid priority, a steps array containing a non-string — returns nothing, and the route surfaces that as an explicit error. Nothing is fabricated to fill a gap, which is the property that makes the stage safe to have at all.

Blueprints are stored append-only with the model name recorded per row. Regenerating supersedes rather than overwrites, and nothing in this stage ever modifies a verdict or an assessment.

3.10 Stage 08 — Executive summary

The executive summary is fleet-wide and read-only, and its notable property is that it recomputes nothing. It imports the other stages' own functions — the report model, the risk computation, the compliance evaluator — and aggregates their output. It therefore cannot drift out of sync with the dashboards it summarises, because there is no second implementation to drift.

It produces no model-generated narrative text. Every sentence is a template filled with an already-computed value; the aggregation of the previous stage's blueprints is what the stage name refers to.

3.11 Stage 09 — Security dashboard

The web client is a React single-page application with 28 routes, nineteen of them surfaced in a five-group navigation — Start, Settings, Pipeline, Records and Organisation-wide — and the rest detail pages, two redirects and a not-found route. The Pipeline group's order is written to match the pipeline's own order, and a test enforces it: the navigation array is still a separate hand-maintained list, but one test asserts that the Pipeline group's links appear in exactly the pipeline's own sequence, and a second pins the phase list itself. Navigation drifted from the architecture it represents in an earlier build, which is why the guard exists.

The guided pipeline is the organising idea. A shared module defines five tracked phases and computes, live, which a device has reached — from its evidence, compliance status, vulnerability data and risk record. Nothing is read from a stored pointer, so a device's position cannot become wrong through a missed update.

There were six phases until the five-control policy-as-code pilot was retired as a stage of its own. The retirement is a small change with a wide blast radius, and three things in this report follow from it: the phase count above, the removal of a risk factor from Table 5, and the rehoming of the device identification signal. That collector had been written for the pilot's first control, `SA-IOT-001`, whose pass and fail conditions read exactly the field it returns, and until the retirement that was the only *compliance* rule consuming it — the identity auto-fill described under Stage 03 had been reading the same observation for its own purposes for some time. The pilot's verdict path still evaluates it; what changed is that the same observation is now *also* mapped to NCA guideline 2-1-1, which is where the evidence had always belonged. What was retired is the stage, not the machinery: the policy engine is generic and still evaluates whatever control definitions are present, and the pilot's five control files — identified `SA-IOT-001` through `SA-IOT-005`, and referenced by that name in Appendix A — remain in the repository. The route that presented it redirects to the NCA compliance page rather than returning a dead link.

3.12 Stage 10 — Continuous monitoring

This stage is not built, and that is confirmed by search rather than assumed: there is no scheduler library, no recurring re-scan, no trend table in any migration, no chart plotting a metric over time, and no alerting code. The worker's poll cycle does carry five age-gated refreshers that keep the vulnerability, vendor-advisory and hardware-address caches current, so the platform is not entirely without time-based work — but nothing re-assesses a device on a schedule, which is the part that matters. Every assessment still begins with a row a person or the API created.

What makes this the most tractable of the remaining gaps is that the data already accumulates. Every assessment, verdict and evidence row is append-only, so the raw material for trend analysis is written on every run — it is simply never read back as a series.

3.13 Bonus stage — Post-quantum readiness

A stage outside the reference pipeline's ten assesses whether a device's cryptography is positioned for the migration to post-quantum algorithms. It sits between risk assessment and remediation, and it is informational by construction: it never contributes to a risk score, never produces a compliance verdict, and never becomes a fourth framework requiring sign-off.

That constraint is deliberate and worth defending, because the obvious design would have been to make it a compliance check. There is no enforceable post-quantum regulation for IoT to assess against — nothing that stands to CGIoT-1:2024 as CGIoT-1:2024 stands to a device. Presenting the stage as compliance would therefore have meant inventing a standard and then scoring against it, which is precisely the failure this project set out to avoid. The three criteria it checks are instead named technical properties grounded in the NIST post-quantum standards, stated as such.

Two collectors carry the stage. The first performs two TLS handshakes: one offering hybrid post-quantum key-exchange groups, to see what the device negotiates, and a second to obtain the certificate itself so its signature algorithm can be read. The second inspects a firmware image's cryptographic inventory against a small reference table of library versions known to support the standardised algorithms — a table whose one entry was confirmed against the worker's own image rather than assumed, with other libraries left out rather than guessed at.

The stage also produced the clearest example in this project of why live verification is not optional. An earlier version of the handshake offered a fourth hybrid group name, arrived at by analogy with the three that were verified. That group does not exist in the installed cryptographic library, and offering it caused the library to reject the entire argument — so every TLS-capable device in the laboratory reported a connection error rather than a result. Unit tests did not catch it, because they exercised the parser against recorded output rather than the command against a real device. A single live run did.

One defect remains open and is reported rather than fixed: the handshake collapses "the device speaks only TLS 1.2" and "the device was unreachable" into a

single indeterminate result, so an absence of post-quantum support is not currently distinguishable from an absence of an answer.

3.14 Cross-stage — the Fully Automated Run

A single action drives the pipeline end to end: discovery and registration, fingerprinting and control evaluation, then vulnerability intelligence — the firmware path for devices carrying an image, and the device-level path for any device whose vendor and model are known, so a device nobody has firmware for is still assessed — then post-quantum readiness and NCA compliance. Risk needs no step because it is computed on read. Each stage boundary is a checkpoint that doubles as the cancellation point.

Fault handling is honest but uneven. Individual failures inside a stage are absorbed and recorded, and the run continues; but all stages sit inside a single outer exception handler, so an unhandled error anywhere aborts the remainder. There is no per-device isolation, and execution is sequential and single-threaded.

3.14.1 The attestation problem

A genuine tension arises here. The database requires every non-trivial assessment status to be attested by a named person in a stated role. An automated run produces statuses with no human involved. Either the constraint is relaxed, or the run misrepresents itself.

Neither happened. The run sets attestation using a system role and a statement that explicitly discloses that no human reviewed it, and additionally sets a separate flag whose entire purpose is to record that fact structurally. That flag is what the interface reads to badge the row differently and offer a review action, which supersedes the automatic row with a human-signed one; the original is never deleted. The constraint stays intact, the automation stays honest, and the difference between the two remains machine-readable rather than a matter of reading the assessor's name.

One asymmetry completes the design, and it is the clearest single statement of the principle the whole platform is built around. The run records suggested failures and suggested reviews automatically under that system attestation. It never records a suggested pass. Passes are counted instead and left in the per-device workspace, so they surface in the run summary as work waiting for a person rather than disappearing into a clean-looking result. A machine may flag a suspected problem unattended; only a person may certify compliance. The asymmetry is not a safety margin bolted on afterwards — it is what the distinction between a suggestion and an assessment means when the suggesting party is not present to be accountable for it.

3.15 Evaluation procedure

3.15.1 Tools and frameworks

The platform is built on FastAPI and PostgreSQL with a React and TypeScript front end, orchestrated by Docker Compose. Evidence collection uses nmap for port and service discovery, tcpdump for traffic capture, OpenSSL for TLS inspection, YARA for firmware pattern matching, and Grype [8] for vulnerability matching, and Google's

Gemini API for remediation blueprint generation. The software bill of materials Grype scans is built in process from the firmware's own package manifest rather than by a separate tool.

3.15.2 Test inputs

Test inputs are the containerised fixtures described above. The three graded security postures are the primary evaluation instrument: a collector that cannot distinguish the insecure fixture from the hardened one is not measuring what it claims to measure. The five protocol fixtures test generalisation — whether a check written against a camera still behaves correctly against a lock or an industrial gateway.

3.15.3 Verification

Three mechanisms verify the system, at different levels.

- **Automated tests.** 1,433 test definitions across 132 files — 1,096 Python test functions across 80 files covering the policy engine, API routes, worker and device fixtures, and 337 front-end cases across 52.
- **Live verification.** Every feature is exercised against the running laboratory before being recorded as complete. Two phase acceptance documents record the result item by item, each entry naming the artefact or the command that demonstrated it. The runbook for the manual collector pass is the weak point in this mechanism: it names eight collectors, one of which no longer exists, so it covers seven of the thirty-two — and it has fallen behind the catalogue in a second way, still describing an external tool the build no longer invokes.
- **Adversarial code audit.** An internal audit of the live codebase was conducted in multiple independent passes, each required to cite file and line and explicitly forbidden from trusting the project's own changelog. Its findings are reported in Section 4 rather than quietly fixed.

3.15.4 Metrics

The platform is evaluated on guideline coverage — how many of the 81 are reachable by an evidence mapping or by an authored checklist — on stage completeness against the ten-stage reference pipeline, and on the audit's own findings. Coverage is computed live from the database rather than asserted.

3.16 Measuring the automatability ceiling

The coverage metric above describes this implementation. A separate procedure was used to judge what a network-and-firmware scanner could settle at all, independently of what this project happened to build.

One qualification governs everything that follows and belongs in the definition of the exercise rather than in its limitations. The result is not a property of CGIoT-1:2024 alone. It is the reachability of CGIoT-1:2024 *under the observability model and decision rules defined below* — specifically: evidence must be obtainable from a device, its firmware image or its network traffic; management-plane sources such as cloud

consoles and fleet APIs are excluded; and full automation requires evidence sufficient to settle a requirement in both directions. A different access model yields a different number. Wherever this report says *ceiling*, that model is what bounds it, and the phrase should be read as shorthand for *the ceiling under this observability model*.

3.16.1 The procedure

The full procedure was written before any guideline was classified, and every subsequent change to it was logged with a date and a reason. It defines an observable signal as any deterministic data point extractable programmatically from a device, its firmware image or its network traffic, without subjective human interpretation; management-plane sources are excluded.

Each guideline is then processed in a fixed order: restate the requirement as a proposition that is true or false of a specific system; identify who bears the obligation, and stop at manual if it is anyone other than the device or its software; decompose into atomic sub-requirements; name the evidence that would convince, and check it against the scope; decide whether observation settles each part in both directions, one direction, or neither; combine; and record one sentence of justification.

Three categories are used. **Automated** means observation settles the verdict in both directions. **Hybrid** means it reaches part of the requirement. **Manual** means it yields nothing decisive.

3.16.2 The rules that decide hard cases

Four rules govern the difficult decisions. The first attracts the most attention, but on the recorded justifications it is the second that decides most cases.

- **The one-direction clause.** A guideline where observation can establish non-compliance but not compliance is hybrid, never automated. Technical evidence refutes far more readily than it confirms.
- **The mixing rule.** Every part automated gives automated; no part automated gives manual; anything else gives hybrid. This replaced an earlier "weakest link" phrasing, which would have concealed genuine partial automation.
- **The hedge rule.** Conditional phrasing is disregarded for the core evaluation unless the condition deciding applicability cannot itself be established programmatically. A hedge alone is never a sufficient reason.
- **The tie-break.** Where a classification cannot be decided, take the more human-reliant category and record confidence as low. An unnecessary manual check costs an auditor time; a false pass reports a device as compliant when it is not.

A classification was never settled by reference to how the platform classifies the same guideline, which would have made the exercise circular.

3.16.3 Coding

All 81 guidelines were classified against the written procedure, each with a one-sentence justification recorded at the time the decision was made. The classification was then locked, and every subsequent change to it was logged with a date and a reason.

Nine entries were revised following a review of the first pass. Eight changed classification and one kept its classification with the justification rewritten. The revisions fall into four groups: three guidelines whose requirement is a periodic review — a process fact rather than a device state — moved from hybrid to manual; two moved down under the one-direction and mixing rules; two had used a hedge as the whole justification, which the procedure forbids; and two had simply misjudged the observable set, one in each direction.

This is a single reading of the framework by one project team, and it is reported as such. It has not been through a formal reliability exercise, and Section 6 treats that as the main limitation on how the resulting figures should be read.

4 Results

Results are reported at three levels: what the platform implements against the reference pipeline, how much of the framework it brings under assessment, and what the classification procedure established about how much of the framework is assessable at all. Every figure below was derived from the running system or the locked classification dataset rather than from project documentation.

4.1 Pipeline completeness

Nine of the ten stages of the IoTGuard reference pipeline are implemented, together with one bonus stage outside the original ten and a cross-stage orchestration layer. The baseline for this measure is the reference pipeline itself.

4.2 Framework coverage

The compliance module brings 76 of the 81 guidelines under a guided assessment workflow through two disjoint mechanisms: 17 guidelines carry automated or hybrid evidence via 50 finding-mapping rules, and 59 carry hand-authored checklist questions. The sets do not overlap — the 59 are precisely the non-device guidelines — so the total is a sum rather than a union requiring deduplication.

The boundary between the two mechanisms is not fixed, and one guideline crossing it during development shows why the total alone is a poor progress measure. Device identification, 2-1-1, was answered by a two-question checklist until its collector's evidence was mapped to it; the catalogue then reclassified it from organisation-scope to device-scope, its checklist was withdrawn, and it entered through the mapping layer instead. That guideline's answer changed from something a person typed to something a machine gathered — a real improvement — while the count of guidelines under the workflow did not move at all. Most of the remaining headroom has this shape: better answers rather than more of them.

The five uncovered guidelines are 2-9-2 (timely patching), 2-14-2 (application allowlisting), 2-15-3 (end-of-life and disposal planning), and 3-1-1 and 3-1-2 (fail-safe and resiliency capability). Both facts that put them there are facts about the platform's own catalogue rather than about the framework: the catalogue records all five

Table 6 Stage completeness against the ten-stage reference pipeline. "Generic" indicates the implementation carries over to a different device set; "lab-bound" indicates it depends on this laboratory's fixtures; "prototype only" indicates the stage exists in code but does not function in the reference deployment.

Stage	Status	Note
01 Platform	Built, generic	Two enforced network boundaries, 17 migrations
02 Discovery	Built, lab-bound	Two-stage sweep; scope now configurable
03 Fingerprinting	Built, mixed	Identity auto-detect; one live third-party lookup
04 NCA compliance	Built, generic	Pure-function evaluator, dual coverage mechanism
05 Vulnerability intel	Built, mixed	Offline; firmware and device paths; CPE table incomplete
06 Risk assessment	Built, generic	Six weighted factors, full breakdown
07 AI remediation	Prototype only	Never fabricates; cannot reach the network in the base configuration
08 Executive summary	Built, generic	Pure aggregator; cannot drift
09 Dashboard	Built, generic	28 routes; accessibility partial
10 Continuous monitoring	**Not built**	No scheduler, no trend, no alerting
Bonus: post-quantum	Built, mixed	Informational; never affects risk
Fully Automated Run	Built, lab-scale	Sequential; no per-device isolation

Table 7 Framework coverage. The baseline is the full framework: 81 guidelines. The five uncovered guidelines are named rather than absorbed into a percentage.

Mechanism	Guidelines	Share of 81
Automated or hybrid evidence	17	21.0%
Authored checklist questions	59	72.8%
Under guided assessment	**76**	**93.8%**
Neither mechanism	5	6.2%

as device-scope, so no checklist was authored for them, and its `assessment_type` field marks all five as manual, so no mapping rule targets them either.

The classification in Appendix A is a different instrument and reads three of the five differently. It judges 2-9-2 and 2-14-2 reachable — they are the two device-scope entries among the eight reachable-but-unbuilt guidelines of Section 4.6 — and records 2-15-3 and 3-1-1 as organisation-scope, which on that reading would have made them eligible for a checklist. The disagreements are recorded rather than reconciled, because the two instruments were deliberately kept apart and neither was allowed to correct the other.

4.3 Verification

The test suite comprises 1,433 test definitions across 132 files: 1,096 Python test functions across 80 files covering the policy engine, API routes, worker and device fixtures, and 337 front-end cases across 52. More cases are collected at run time than there are definitions, because parametrised suites expand one function into several: a full run collects 1,482.

The suite was executed end to end for this report, and every case passes — 1,145 Python cases and 337 front-end cases, with four skipped because an optional PDF-rendering library is absent. Reproducing that result requires the environment the containers provide, and the requirement is worth stating because it is a real limit on portability rather than a detail: the API suite provisions its own PostgreSQL instance, several modules resolve paths against the image's working directory rather than the repository root, and one device fixture pins a protocol library to a version whose successor moved the symbol it imports. Outside that environment the suite does not fail so much as decline to start, which is the more accurate way to describe it.

The project maintains 33 written defect postmortems, each recording cause, fix and prevention. Their distribution is itself a result: 21 of the 33 were written in the first week, and the dominant category there is environment failure rather than application logic — container tooling, package availability, healthcheck semantics. The split is not clean in either direction, and it would be convenient rather than accurate to present it as one: the first week also records two design defects, a process defect and two genuine code defects, and several of the later twelve are environment failures too, among them a Docker DNS corruption that survived a reboot. One entry remains open, its root cause not conclusively determined.

> **Attention**
>
> No continuous integration exists. The suite passes locally but nothing runs it automatically before a merge, so its protective value depends entirely on discipline.

4.4 Internal code audit

An adversarial audit of the live codebase was conducted in multiple independent passes, each required to cite file and line and forbidden from trusting the project's own changelog. Its findings are given below by name rather than as a count, so that each can be checked against the source directly; the audit's working register is not reproduced here, and no total is claimed for it.

Two findings cut across every stage. First, at the time of the audit every scan-based stage was gated by a single hardcoded subnet constant, written independently in two modules — meaning discovery, fingerprinting, device-scope compliance and vulnerability intelligence were not merely untested outside the laboratory but code-level incapable of targeting a real network. Second, the API has no authentication of any kind: every sign-off, override, exception approval and destructive action is reachable without credentials, and every reviewer identity is an unverified free-text string.

The first of these has since been closed. A migration replaced the constant with a database-backed allowlist, and the laboratory range survives as a named, deactivatable preset. Four items the audit rated critical remain open at the time of writing — authentication, continuous integration, Stage 10, and the vulnerability CPE override table — and each is verifiable in the source as stated: there is no credential handling anywhere in the API, no continuous integration configuration in the repository, no scheduler, trend or alerting code, and the override table holds a single entry.

The audit also surfaced defects that a narrative review would not have found: a control's first-ever assessment writes no audit event, because the event is emitted only when a prior row exists; exception expiry is evaluated at query time and never written back, so a stored status can outlive its own effect; and the post-quantum handshake collapses "speaks only TLS 1.2" and "unreachable" into a single indeterminate result.

4.5 The automatability ceiling under the defined observability model

Classifying all 81 guidelines against the written procedure produced the result in Table 8. Every figure in it is bounded by the observability model stated in Section 3.16 — device, firmware and network traffic only, no management plane — and is an estimate of reachability under that model rather than a property of the framework in itself.

Table 8 Automatability classification of CGIoT-1:2024. "Reachable" denotes automated or hybrid as a proportion of the domain.

Domain	Guidelines	Automated	Hybrid	Manual	Reachable
1 Governance	27	0	1	26	3.7%
2 Defense	41	0	21	20	51.2%
3 Resilience	2	0	0	2	0%
4 Third-party and cloud	11	0	1	10	9.1%
Total	**81**	**0**	**23**	**58**	**28.4%**

No guideline in the framework is fully automatable. Twenty-three (28.4%) are partly automatable and 58 (71.6%) require a human. The distribution is markedly uneven: 21 of the 23 reachable guidelines lie in Domain 2, Cybersecurity Defense, which is also the only domain where reachable guidelines form a majority. Figure 2 shows the same result against domain size, which is where the unevenness is most visible.

Two structural facts explain most of the result. First, 51 of the 81 guidelines address the organisation rather than a device — approving a policy, training staff, reviewing something on a schedule — and no scan of any device can observe those. Second, every remaining guideline that a scanner can reach at all turns out to be a compound requirement: it pairs something observable with something that is not.

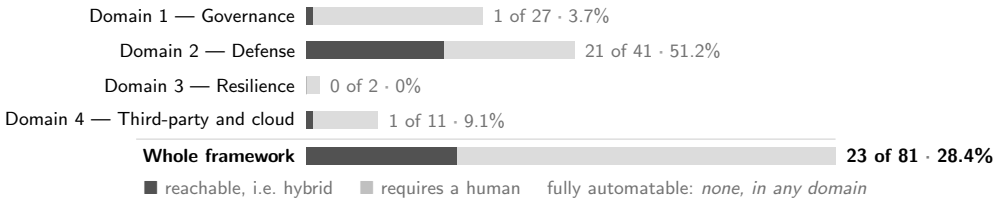


Fig. 2 The automatability ceiling by domain, under the observability model of Section 3.16. Bar length is proportional to the number of guidelines the domain contains, so the figure shows both how large each domain is and how much of it a network and firmware scanner could ever settle. Twenty-one of the 23 reachable guidelines lie in Domain 2; no guideline anywhere in the framework is fully automatable, so the chart needs only two shades.

4.6 What the platform reaches, against what is reachable

Setting the classification beside the platform's own coverage closes the loop between the two halves of this report. Of the 23 guidelines the classification marks reachable, the platform implements machine evidence for 15. Eight are reachable in principle but unbuilt — six are currently answered by checklist, and two, 2-9-2 and 2-14-2, are device-scope and so covered by neither mechanism. Two guidelines the platform reaches, 2-11-2 and 2-13-2, the classification marks manual, which is a real difference of reading rather than an artefact of timing.

Those are three different numbers for what is loosely called coverage: 76 of 81 under a guided workflow, 17 of 81 carrying machine evidence, and 23 of 81 reachable at all under the observability model. They are not interchangeable, and Section 5 takes up why the distinction matters more than any one of them.

4.7 Evaluation summary

The results above establish the main coverage figures. The following tables consolidate the evaluation evidence that is otherwise distributed across the implementation description, test report and adversarial audit. They report measured or explicitly observed outcomes only; no result below extends the claims beyond the simulated laboratory.

Table 1: Automated verification footprint and executed result.

Evaluation item	**Reported quantity**	**Outcome / interpretation**
Test definitions	1,433 across 132 files	Broad backend and frontend verification surface.
Python definitions	1,096 functions across 80 files	Policy engine, API routes, worker logic and device fixtures.
Frontend definitions	337 cases across 52 files	Dashboard and interaction behaviour.
Collected cases in a full run	1,482	Parameterised tests expand definitions into additional cases.
Executed Python cases	1,145 passed	All collected Python cases passed in the evaluated container environment.
Executed frontend cases	337 passed	All collected frontend cases passed.
Skipped cases	4	Optional PDF-rendering dependency absent; reported rather than counted as passes.
Written defect postmortems	33	21 were produced in the first week; one recorded issue remained open.
Continuous integration	Not implemented	Passing suites are local evidence; no merge gate automatically re-runs them.

4.8 Objective completion

Table 5 closes the Results section by assessing the objectives stated in Section 1.6. “Achieved” means that the stated prototype objective is supported by reported evidence; it does not imply production readiness. “Substantially achieved” marks an objective whose central function is demonstrated but whose stated reference scope contains an explicit unbuilt or bounded element.

Table 2: Compliance-assessment mechanisms and what each result means.

Mechanism	**Scale**	**Framework reach**	**Evaluation result**
Automated collectors	32 collectors	Evidence for 17 guidelines	Collectors return structured observations; they never record a compliance verdict.
Evidence mapping	50 rules	Same 17 guidelines	28 rules suggest fail on insecure evidence, 4 suggest review, and 18 can suggest pass on a clean observation; precedence is fail > review > pass.
Human checklist path	148 questions across 59 guidelines	59 non-device guidelines	Organisational/supplier/cloud obligations are guided and recorded rather than misreported as automated.
Neither current mechanism	5 guidelines	5 of 81	Uncovered items are named explicitly rather than absorbed into a percentage.
Human attestation	Required for recorded non-trivial assessments	All signed assessments	System-generated automatic runs are structurally distinguished from human review; automatic passes are withheld.

Table 3: The four coverage quantities and their evaluation significance.

Quantity	**Result**	**Share of 81**	**What it establishes**
Guided workflow coverage	76 guidelines	93.8%	The platform prompts and records an assessment through either machine evidence or a human checklist.
Machine-evidence coverage	17 guidelines	21.0%	The implementation currently collects technical evidence that can support a guideline assessment.
Framework reachability	23 guidelines	28.4%	Under the fixed network-and-firmware observability model, at least part of the requirement contains scanner-observable evidence.
Full automation	0 guidelines	0%	No guideline has scanner evidence sufficient to establish both compliance and non-compliance for the entire requirement.
Reachable but not implemented	8 of the reachable 23	—	Engineering headroom under the classification; six are currently checklist-routed and two are covered by neither mechanism.

Table 4: Laboratory evaluation dimensions and observed outcomes.

Evaluation dimension	**Controlled input / method**	**Observed result**
Security-posture discrimination	Three camera fixtures configured as insecure, partially hardened and hardened.	Provides a graded baseline for checks that should distinguish clearly insecure, intermediate and clean technical conditions; the associated automated suites passed.
Protocol/device diversity	Smart lock, Modbus gateway, UPnP router, RTSP NVR and mDNS speaker in addition to the camera profiles.	Exercises protocol-specific collectors rather than evaluating only one device class.
Discovery branch coverage	IoT-signature ports, Telnet-only host and hosts without a matching signature.	IoT, uncertain and unknown classifications are reachable; an earlier use of `-open` that hid the unknown branch was removed.
Vulnerability evidence	Firmware SBOM/Grype/KEV path plus a seven-entry device product map.	Vulnerabilities can be reported with exploit prioritisation; an unmapped NVR remains explicitly unmatched rather than being labelled clean.
Post-quantum live verification	Hybrid TLS group negotiation plus firmware cryptographic inventory.	A live run exposed an invalid group name missed by parser unit tests; it was corrected. TLS 1.2-only and unreachable remain conflated as one open indeterminate case.
End-to-end orchestration	Sequential Fully Automated Run across stages.	Fail/review suggestions can be system-attested; pass suggestions are left for a person. An unhandled exception can still abort later stages because per-device fault isolation is unbuilt.
Adversarial source audit	Source/configuration review independent of feature happy paths.	Surfaced material limitations including lack of production authentication, no continuous monitoring and the former inability to target a real network.

Table 5: Objective-completion matrix.

Obj.	Assessment	Key evidence	Completion statement
O1	Substantially achieved	9 of 10 reference stages implemented; bonus PQ stage built	End-to-end assessment functions are present, but Stage 10 continuous monitoring remains unbuilt.
O2	Achieved for the prototype scope	76/81 guided; 17 machine-evidence; 59 checklist	The framework is operationalised through two explicitly different mechanisms; five residual guidelines are named.
O3	Achieved with an identity limitation	Structured evidence, attestation, supersession, audit events	Decision provenance is preserved, but signer identity is still unverified free text because authentication is unbuilt.
O4	Achieved	76 workflow, 17 machine, 23 reachable, 0 fully automated	The report measures implementation coverage and framework reachability separately and explains why they are not interchangeable.
O5	Achieved within the simulated laboratory	8 device fixtures; 32 collectors; 1,433 definitions; all executed cases passed	Reproducible controlled validation is demonstrated; external validity to physical hardware remains future work.
O6	Achieved as comparative/informational analysis	Formal OWASP mapping; NIST FIPS 203–205-based PQ criteria	The comparisons are grounded in recognised references and are explicitly not presented as additional certification.
O7	Achieved	33 postmortems plus adversarial audit and limitation/next-step pairs	Material gaps are documented rather than hidden, including authentication, CI, monitoring, real-hardware validation and classification reliability.

6 Discussion

6.1 Why coverage stops at 76

The platform brings 93.8% of the framework under a guided workflow, which sounds close to complete and is not. The figure needs unpacking, because two very different things are being counted.

Seventeen guidelines are covered by evidence a machine gathers. Fifty-nine are covered by questions a human answers. Both are real coverage in the sense that an auditor working through the platform is prompted, guided and recorded on all 76—but only the first seventeen represent work the machine does. Reporting 93.8% without that distinction would be the exact failure this project set out to avoid.

The remaining five are instructive precisely because they are not a random residue. All five are device-scope in the platform's catalogue, which excluded them from the checklist mechanism by construction, and the same catalogue marks all five manual, which excluded them from the mapping layer. They fell through a gap between two mechanisms whose boundary was drawn by scope rather than by difficulty. That is a design consequence, not an oversight, and the fix is to allow device-scope guidelines to carry checklists too—an increment rather than an architectural change.

5.2 Why the ceiling is zero

The result that no guideline is fully automatable is the one most likely to be read as an artefact of strictness — as a restatement of how automation was defined rather than a finding about the framework. The objection deserves a direct answer.

The rule usually blamed is the one-direction clause: a guideline where observation can establish non-compliance but not compliance is hybrid. Consider the prohibition on default credentials. The platform tries ten documented credential pairs. If one succeeds, the device is conclusively non-compliant. If all ten fail, the platform has established that those ten failed — not that no default credential exists. The set that can be tested is finite; the claim the guideline makes is universal. No amount of engineering closes that gap, because it is a gap in what evidence of this kind can support, not in the quality of the check.

That argument is sound, but it carries less of the result than the attention paid to it suggests. The clause is not what holds most of the 23 reachable guidelines short of full automation — each of those separately contains a component that no observation settles in either direction, which is the mixing rule's doing rather than the one-direction clause's. Even the default-credentials guideline that motivates the clause requires periodic password changes no scan can see. So the objection lands on the clause, but the clause is not where the weight sits.

We retain the strict rule anyway, because the asymmetric cost favours it. An unnecessary manual check costs an auditor an hour. A false pass tells an organisation a device is compliant when it is not. But the reported figure does not depend on that preference.

What the figure does depend on is the scope boundary. Excluding management-plane sources is a modelling choice, not a fact about the framework, and a different choice would produce a different ceiling — which is why the number should always be read with the observability model attached to it.

> **Key insight**
>
> The practical reading of a zero is not that automation is futile. It is that *no compliance scanner should ever report a bare pass.* Every automated positive is provisional, and a tool that presents it otherwise is making a stronger claim than its evidence supports.

The platform tests that principle against itself rather than merely stating it. Eighteen of its fifty mapping rules fire on a clean reading and suggest a pass, so the obvious objection is that the tool does exactly what the insight forbids. It does not, and the difference is worth being exact about, because it is the difference between the principle being a design constraint and being a slogan. The platform suggests a pass; it never records one. A person records it, and at the moment of recording sees the set of observation fields that actually produced the suggestion — so a pass resting on one clean check is visibly different from one resting on three. The Fully Automated Run, which records fails and reviews without a human present, is forbidden from recording a pass at all. What the zero rules out is not the machine forming a positive opinion,

which is useful; it is the machine issuing a positive verdict, which its evidence cannot support. Keeping those two apart is most of what the compliance module does.

5.3 Where the framework is reachable, and why

The concentration of 21 of 23 reachable guidelines in a single domain was not anticipated, and is the most useful finding for anyone planning an audit budget.

The explanation is structural rather than technical. Domain 2 covers technical defensive measures — configuration, encryption, authentication — and asks predominantly about the state of a device. The other three domains ask about governance, resilience planning and supplier management, which are obligations placed on an organisation. Fifty-one of the 81 guidelines are addressed to the organisation rather than to a device at all.

Automation potential in a security framework is therefore not spread thinly across it; it clusters where the framework happens to talk about machines. An organisation deciding how much human auditing to budget for cannot use an average — it needs the distribution, because a domain-by-domain plan and a whole-framework plan give very different answers.

5.4 The framework subsumes the OWASP IoT Top 10

A reader who works in IoT security will reach for a familiar reference point, and the natural one is the OWASP IoT Top 10. Reading all 81 guidelines against those ten categories produces a clean result: the framework addresses every one, and several almost verbatim.

Guideline 2-2-2 requires preventing "default and hard-coded passwords"; 2-15-2 requires "disabling certain features or functionalities that would not be used"; 2-4-6 requires that update files be "transmitted over a secure connection"; 2-7-2 requires encrypting data "both at rest or in transit"; 2-13-2 requires "hardware tamper protection and detection". Those are the OWASP categories for weak passwords, insecure network services, insecure update mechanisms, insecure data transfer and storage, and lack of physical hardening, written as regulatory requirements. Ecosystem interfaces are covered more thoroughly than the scanner can reach, across the device (2-14-1), the companion mobile application (2-5-1) and the cloud service (4-2-1 to 4-2-5). Device management, which OWASP treats as a single category, is spread across asset inventory (2-1-1 and 2-1-2), event logging to a security operations centre (2-11-1) and the three lifecycle guidelines of subdomain 2-15. Insecure defaults are reached twice over, by 2-2-2's prohibition on shipped credentials and by 1-2-2's requirement for "hardening / minimum baseline security standards". Even the software bill of materials appears, as 4-1-3's requirement to obtain "a list of hardware and software components".

One category is covered with a qualification. Data protection and data minimisation are explicit — 2-6-3 requires preventing devices from collecting sensitive data "that is not needed or cannot be adequately protected" — but consent and data-subject rights appear nowhere in the framework's text, and the only retention rule it states — 2-11-1's twelve-month minimum — governs how long cybersecurity event logs are kept rather than how long personal data may be held. A keyword sweep of

all 81 requirements finds no mention of consent, personal data, or a data subject. The nearest the framework comes is 2-6-1's instruction to classify and label IoT data "as per related laws and regulations", which points outward without saying to what; in the Kingdom the natural referent is the Personal Data Protection Law. Whether that silence is a deliberate division of labour between the two instruments or simply a gap, the framework's own text does not say, and this report does not assume.

Key insight

The two references are not alternatives. CGIoT-1:2024 is a superset: everything the OWASP IoT Top 10 names, the framework already requires, and it then adds governance, third-party and resilience obligations the Top 10 does not touch.

This matters because it closes off an objection this project would otherwise invite. A reader could reasonably ask whether the low automatable share reflects a framework that is unusually documentary — whether a more technically-minded standard would have yielded a higher ceiling. The comparison says otherwise. The framework covers all ten of the weakness classes the security community considers canonical for IoT. Assessed against the platform's own collectors, three of those ten — weak credentials, insecure network services and insecure defaults — are reached as fully as a scanner can reach them; for the other seven, part of the category lies outside what any network and firmware scan can observe — the storage behind an encrypted transport, whether an update is signed rather than merely present, whether a log is monitored rather than merely emitted. The limit is not the framework's technical seriousness. It is observability: what can be seen from outside a closed appliance.

That is a consistent reading of the same limit from a second direction, stated in a vocabulary a practitioner recognises without needing the classification procedure of Section 3.16 explained first. It should not be read as an independent validation of the ceiling: the same team performed both readings, against the same notion of what a scanner can observe, so the agreement shows the two accounts are consistent rather than that either confirms the other.

Formal OWASP IoT Top 10 mapping

To make the comparison in Section 5.4 reproducible, Table 1 maps each category of the OWASP Internet of Things Top 10 (2018) [10] to the CGIoT-1:2024 requirements used in this report and states what a network-and-firmware scanner can actually observe. OWASP is used as a security-weakness taxonomy, not as a compliance standard; the mapping therefore demonstrates thematic coverage and observability, not equivalence of the two instruments.

Table 1: Formal mapping of the OWASP IoT Top 10 (2018) to CGIoT-1:2024 and Raqib's evidence boundary.

ID	OWASP category [10]	Principal CGIoT-1:2024 mapping	Raqib / scanner observability
I1	Weak, guessable, or hard-coded passwords	2-2-2 (default and hard-coded passwords)	Strong technical reach for tested default/hard-coded credentials; exhaustive absence of all weak credentials cannot be proven.
I2	Insecure network services	2-15-2 (disable unused features/functions); related service-exposure controls in Domain 2	Strong reach through port/service discovery and protocol probes; whether a service is organisationally "necessary" can still require context.
I3	Insecure ecosystem interfaces	2-14-1 (device interfaces), 2-5-1 (mobile application), 4-2-1–4-2-5 (cloud)	Partial. Device-facing interfaces are observable; mobile/cloud configuration, governance and contractual obligations largely fall outside the scanner vantage point.
I4	Lack of secure update mechanisms	2-4-6 (secure transport of update files), 2-9-2 and related update/patch obligations	Partial. Firmware/update artefacts and transport properties can be inspected; approval, process quality and complete signing/lifecycle assurance are not settled externally.
I5	Use of insecure or outdated components	4-1-3 (hardware/software component list) plus device/firmware vulnerability evidence	Partial-to-strong technical evidence from SBOM, Grype/NVD/KEV and device intelligence; supplier completeness and lifecycle governance remain human evidence.
I6	Insufficient privacy protection	2-6-1–2-6-3 (classification, protection and minimisation of IoT data)	Limited. Some transport/storage signals are observable, but consent, data-subject rights and organisational privacy decisions are outside this framework/scanner comparison.
I7	Insecure data transfer and storage	2-7-2 (encryption at rest or in transit) and related cryptographic requirements	Partial. TLS, MQTT transport and cleartext traffic can be observed; at-rest storage and key-management practice may not be externally visible.
I8	Lack of device management	2-1-1/2-1-2 (inventory), 2-11-1 (security logging), 2-15-1–2-15-3 (lifecycle)	Mixed. Device identity, exposed logging endpoints and some lifecycle evidence are technical; inventory governance, monitoring practice and end-of-life planning require human evidence.
I9	Insecure default settings	1-2-2 (hardening/minimum baseline), 2-2-2 (default credentials), 2-15-2 (unused features)	Strong for measurable defaults such as credentials/services; baseline definition and approval are organisational and therefore manual.
I10	Lack of physical hardening	2-13-2 (hardware tamper protection/detection)	Limited/partial. An exposed tamper-status signal can be checked in the laboratory, but physical resistance and hardware behaviour require physical-device validation.

The table supports the qualitative conclusion already drawn in Section 5.4: CGIoT-1:2024 addresses all ten OWASP categories, but the scanner's strongest practical reach is concentrated in weak credentials, exposed/insecure services and insecure defaults. For the remaining categories, one or more material parts depend on firmware internals, physical hardware, cloud/mobile interfaces, organisational processes or human judgment.

5.5 What the checklist mechanism buys

The 59 authored checklists are, on paper, the least sophisticated component in the platform: three templates and a first-match rule. In practice they are what makes an honest reading of the ceiling survivable as a product.

Without them, a tool that automates 17 of 81 guidelines has two options. It can report 21.0% coverage and leave the auditor to handle the rest unaided, or it can quietly expand what it counts as coverage until the number looks respectable. The checklist mechanism is a third option: it accepts that 59 guidelines need a human, then does the useful thing anyway—asks that human the right question, records the answer against the guideline, and carries it into the same evaluator that handles machine evidence.

That is why the classification work and the build are not separate projects. The classification identifies which guidelines will never yield to a collector; the checklist mechanism is what the platform does about them.

5.6 Where the results miss expectations

Three results fell short of what was expected at the outset, and the gaps are worth naming.

The classification procedure turned out to be less decisive than expected at one specific joint: the boundary between hybrid and manual for guidelines with a small observable fragment. Nine of the 81 entries were revised on review, and the revisions cluster there rather than spreading evenly, which points at an underspecified rule rather than at careless reading. That location is the most useful thing the exercise produced about its own method, and it is what a revision of the rubric should operationalise first.

Stage 10 was expected to be a stretch goal and became the single largest gap. Its absence is more visible than it would otherwise be precisely because every assessment, verdict and evidence row is append-only: the data for trend analysis has been accumulating since the first day and is simply never read back.

Finally, the audit found that the platform could not target a real network at all — a limitation nobody had stated, because nobody had looked. That it took an adversarial pass over the source to surface something so fundamental is itself a finding about how a project's self-knowledge decays when it is recorded in a changelog rather than verified against the code.

5.7 Three numbers, three different questions

The three coverage figures in Section 4 are the practical argument of this report, and they are worth stating together because each is routinely reported as though it were the others.

Seventy-six of 81 is what the platform brings under a guided workflow. It is the largest number and the least informative one about automation, since 59 of those 76 are questions a person answers. Seventeen of 81 is what the machine actually collects evidence for. Twenty-three of 81 is what any network-and-firmware scanner could reach under the observability model of Section 3.16, whether or not we built it — so the gap between 17 and 23 is our remaining engineering headroom, while the gap between 23 and 81 is not headroom at all.

> **Takeaway**
>
> How much of a framework a tool implements and how much of that framework is reachable at all are different quantities, and they behave differently: the first rises as collectors are written, the second does not move. An automation percentage that does not say which of the two it reports cannot be interpreted.

This is why the classification was kept deliberately separate from the platform, and why no guideline was ever classified by reference to how the platform treats it. Had the two been allowed to inform each other, the exercise would have measured our own engineering back to us and called it a property of the framework — which is precisely the failure the separation exists to prevent.

6 Limitations and Future Work

One statement governs this whole section and belongs before the detail. **Raqib is a research prototype for controlled IoT assessment workflows, not a production compliance platform.** Nothing in this report should be read as a claim of deployment readiness: the system has no authentication, has never been run against physical hardware, and does not monitor anything continuously. Its value is as a working demonstration that the two coverage quantities can be separated and reported apart — not as software an organisation could put in front of a real estate today.

The limitations below are grouped into those affecting the platform and those affecting the classification. Most were found by the internal audit rather than volunteered, which is the point of running one. Each is paired with a concrete next step.

6.1 Platform limitations

6.1.1 No authentication or authorisation

The API has no login, no session concept and no credential handling of any kind, and permits requests from any origin. Every reviewer identity recorded against an assessment, an exception approval, an override or a remediation review is an unverified free-text name. This is the single largest obstacle to any deployment beyond a personal laboratory, and it makes the attestation chain — otherwise the most carefully built part of the compliance module — unenforceable in practice, since nothing establishes that the named signer is the person who signed.

Next step. Introduce authentication with a role model distinguishing assessor from approver, and bind the existing attestation fields to an authenticated identity rather than a typed string. The schema already carries the right fields; only their provenance is missing.

6.1.2 No continuous integration

The suite runs only when someone runs it. Nothing gates a merge, so a regression can enter the codebase and remain until a human happens to execute it.

Next step. A single workflow running the four suites on every push. Of the outstanding critical items this is by a wide margin the cheapest, and it protects every other improvement made afterwards.

6.1.3 Stage 10 is entirely unbuilt

Continuous monitoring does not exist: no scheduler, no recurring re-scan, no trend view, no regression detection, no alerting. A compliance position is therefore always

a snapshot, and the platform cannot answer whether a device's posture has improved or decayed.

Next step. The data already accumulates — every assessment, verdict and evidence row is append-only — so the first increment is read-only: a query returning a device's compliance score over time, and a diff between two assessments. Scheduling can follow; the analysis does not depend on it.

6.1.4 Vulnerability matching under-reports silently

Vulnerability identification relies on synthesised package identifiers, which only match public vulnerability data when a component's vendor name happens to equal its product name. This was confirmed true for the two components the laboratory's firmware actually ships, and false for a third package tested in the same spike — the MQTT broker, whose registered vendor is not its product name, and which matched nothing at all. The override table correcting this holds one entry, so for any firmware beyond the laboratory's own fixtures, silent under-matching is the expected behaviour rather than an edge case — and it fails quietly, producing a clean-looking result rather than an error.

Next step. Expand the override table against a real corpus of embedded components, and — more importantly — surface a warning when a component is matched by synthesis alone, so a clean result and an unverified result are visually distinguishable.

6.1.5 Remediation cannot run in the base configuration

The remediation stage issues an outbound request to a language model API, but it runs inside the API container, whose only network is the one marked `internal` and therefore has no route off the host. In the base configuration that call cannot succeed; the stage works only under the development overlay, which relaxes the isolation. Two decisions that were each sound alone — put all persisted state behind an isolated network, and generate remediation from the component that holds the finding — are jointly inconsistent, and the report records that rather than presenting the stage as working.

Next step. Move the outbound call to the worker, which already sits on both networks and already performs every other external fetch. The stage's interface does not change; only which container makes the request does.

6.1.6 One collector breaks the no-network-at-scan-time rule

Every other evidence path is offline by construction: the vulnerability databases are refreshed on a schedule and read from a local cache, so a scan never waits on a third party and two scans of the same target against the same snapshot agree. The MAC vendor collector is the exception. It tries a local per-address cache, then a cached copy of the public hardware-address registry, and only then queries a third-party web service during the scan. How often the third tier is actually reached in this laboratory is not established, and the distinction matters enough to state rather than assume: the source records that all eight fixtures resolved against the *web service*, which is a measurement of the third tier, not evidence that the second one would have answered

first. What is certain is that on a fresh installation neither offline tier resolves until the first registry refresh completes, and it is exactly then that the network tier fires.

The consequences are worth being precise about, because they are smaller than they sound and larger than nothing. The collector degrades rather than fails: with no network it records the lookup error and drops its own confidence, and no compliance verdict depends on it. But the recorded observation includes which tier answered, so the same device scanned twice can produce different evidence — the first run naming the web service, the second the cache it populated. That breaks strict reproducibility for this one collector, and reproducibility is a property the rest of the platform treats as non-negotiable.

Next step. Ship the hardware-address registry with the worker image so the offline tiers resolve from the first scan, and make the network tier opt-in. The lookup function already takes a flag to disable it, but nothing in production passes that flag and no environment variable controls it, so turning it off today means editing the source — which is a smaller job than it sounds and a larger one than the flag's existence suggests. Measuring how often each tier answers would cost one counter and would settle the open question above.

6.1.7 A suggested pass can rest on a partial check

The mapping layer suggests a pass when a rule fires on a clean reading, and one matching rule is enough — symmetric with the treatment of a fail. A guideline carrying several independent aspects can therefore be suggested as a pass on the strength of one of them while the others were never exercised. Guideline 2-4-3 carries six independent fail-rules against two pass-rules; a scan touching only one of the two aspects those pass-rules cover still produces a suggestion.

The design discloses rather than withholds: every suggestion carries the set of observation fields that produced it, and the signing interface shows that set for a suggested pass, so the person recording the assessment can see the basis is thin. That is a real mitigation and it is not the same as a guarantee. A tired auditor confirming a suggestion without reading its basis is a foreseeable failure mode, and the platform does not prevent it.

Next step. Two options were considered and neither is obviously right, which is why this is recorded as a limitation rather than a defect. The suggestion could be withheld until every aspect of a guideline has been observed — safer, but it would suppress useful signal on guidelines the laboratory can never fully exercise. Or the interface could rank suggestions by how much of the guideline was actually covered, making a thin pass visibly weaker than a thorough one without hiding it. The second is preferred and is not built.

6.1.8 The device-level vulnerability path is bounded by a seven-entry table

Vulnerability lookup without a firmware image depends on resolving a device's vendor and model to a product identifier, and that resolution is a hand-verified table of seven entries covering this laboratory's fixtures. A device outside the table produces an

honest miss rather than a wrong answer, which is the right failure mode, but it means the path does not generalise beyond these fixtures without further curation. One of the eight fixtures is already unmatched, because the real product it corresponds to has no public identifier at all.

Next step. The table is the wrong long-term shape: a lookup against the public product dictionary at registration time, with the hand-verified table retained only for the cases where the dictionary is wrong or absent, would scale where curation does not.

6.1.9 The laboratory is simulated

All evidence comes from containerised fixtures. Containers reproduce protocol behaviour but not firmware diversity, hardware roots of trust, physical interfaces or the failure modes of constrained devices under load. Results therefore establish that the collectors work against a controlled target set, not that they work against real hardware.

Next step. Validate a representative subset of collectors against two or three physical devices. The realistic goal is not to replace the laboratory but to identify which checks behave differently outside it.

6.1.10 The Fully Automated Run has no fault isolation

Individual failures inside a stage are absorbed, but all stages sit inside a single outer exception handler, so an unhandled error anywhere aborts every remaining stage. Execution is sequential and single-threaded, and the run is not idempotent: re-running accumulates superseding assessment rows rather than converging.

Next step. Wrap each device in its own error boundary so one bad device cannot end a fleet run, and skip suggestions that already have a live assessment, which makes a re-run converge.

6.2 Limitations of the classification

6.2.1 The classification has not been through a reliability study

This is the largest limitation on the classification, and it is a limitation of scope rather than of care. All 81 guidelines were coded by one project team against a procedure written before coding began, with a justification recorded for each. What has not been done is the formal exercise that would establish how reproducible that reading is: a second coder with no connection to the project, working from the framework text and the written procedure alone, coding the full set so that agreement can be measured rather than asserted.

Until that exists, the figures in Section 4 should be read as one team's careful reading of a public document, published in full so that it can be disputed — not as a measured quantity with a known error. The 28.4% in particular is an estimate whose uncertainty is not quantified.

Next step. The full study is planned as separate work, and the material makes it unusually cheap to run: because the object of study is a public document rather than a system or a set of human subjects, an independent coder needs no access to the

project, the platform or the laboratory. The procedure, the 81 codings and the per-guideline justifications are all released with this report (Appendix A), so the exercise can be replicated by anyone from the published material alone.

6.2.2 The scope boundary is a choice, not a fact

Excluding management-plane sources — cloud consoles, fleet APIs, device management platforms — was fixed before coding began and is defensible, but a different choice would produce a different ceiling. The reported figure is the ceiling for a network-and-firmware scanner specifically, not for all technical means.

Next step. Re-run the classification with management-plane access assumed. The difference between the two ceilings is itself a useful quantity: it measures how much of the framework is reachable only through privileged access.

6.2.3 One framework, one team

The classification covers CGIoT-1:2024 alone, and was produced by the same team that built the platform. Neither the shape of the result nor the 28.4% figure can be assumed to generalise to another framework.

Next step. Apply the same procedure to a second framework — IEC 62443 or ETSI EN 303 645 are the natural candidates — with a coder unconnected to this project. The two arguments that produce this shape — compound requirements, and single-direction evidence — are about the nature of technical evidence rather than about this framework, so a similar shape would be expected; that expectation is currently untested.

6.3 Promising directions

Three extensions follow naturally from the work rather than merely being desirable.

- **Publishing the classification in a machine-readable carrier.** All 81 rows and their reasoning are already released with this work and reproduced in Appendix A, so the result can be checked rather than taken on trust. What they are not yet in is a form another tool can consume: expressed in a standard carrier — OSCAL [7] is the obvious one — they would let another team recompute the ceiling under different rules automatically, rather than by re-reading a table.
- **Coverage-aware audit planning.** Since automation potential clusters by domain, a planning tool could estimate human auditor hours for a given estate from the distribution rather than from an average.
- **Closing the loop from checklist to evidence.** The classification marks 23 guidelines reachable, of which the platform implements evidence for 15. The remaining eight have an observable fragment that no collector reaches — six are currently answered by checklist, and two, 2-9-2 and 2-14-2, are device-scope and so covered by neither mechanism, which makes them the highest-value two of the eight. Building those collectors would convert human questions into partial machine evidence, which is the most direct route to raising the automated share honestly.

7 Conclusion

We built Raqib, a platform implementing nine of the ten stages of the IoTGuard reference pipeline against NCA CGIoT-1:2024, and used it to ask a question the platform alone could not answer. The system brings 76 of the 81 guidelines under a guided assessment workflow — 17 through machine-collected evidence from 32 collectors and 50 mapping rules, 59 through hand-authored checklists — verified by 1,433 automated tests and by an adversarial audit of our own source that found, among other things, that the platform could not target a real network at all. Separately, classifying all 81 guidelines against a written procedure judged that 23 (28.4%) contain evidence a network-and-firmware scanner could observe under the stated observability model, 58 (71.6%) do not, and none meets the strict criterion for full automation. That classification is a single team's reading, released in full so that it can be disputed rather than taken on trust; putting it through a formal reliability study is the natural next piece of work and is described in Section 6.

The contribution a reader should carry away is not the platform, and not the number 28.4%. It is the distinction the two halves of this work make visible. This project produced three different figures that all get called coverage — 76 of 81 under a guided workflow, 17 of 81 carrying machine evidence, 23 of 81 reachable at all — and they answer different questions. The first two describe what we built and rise as we write more collectors. The third describes the framework under a fixed observability model and does not move when the software does. **How much of a framework a tool implements and how much of that framework is reachable at all are different quantities, and they must be measured separately.** Any published automation percentage that does not say which of those questions it answers cannot be interpreted, and the gap between them is not measurement noise. It is the entire distance between an engineering status report and a property of the framework.

Code availability. The platform source is available at https://github.com/MohamadTalagi/KAUST-IoT.

Data availability. The classification dataset underlying Sections 3.16 and 4 — all 81 guidelines with their coding, the one-sentence justification recorded for each, and the log of every change made to the coding — is released with this work. Appendix A reproduces the coding and the justifications in full; the change log is in the released dataset. NCA CGIoT-1:2024 itself is a public document of the Saudi National Cybersecurity Authority, published under Traffic Light Protocol: White, and is cited in the bibliography. No third-party or personal data was used.

Appendix A Complete classification of CGIoT-1:2024

All 81 guidelines with the classification recorded against the written procedure of Section 3.16, together with the one-sentence justification the procedure requires for every row. **H** denotes hybrid and **M** manual; no guideline was classified automated. Justifications are reproduced verbatim from the locked dataset.

ID	Class	Recorded justification
Domain 1 — Cybersecurity Governance		
1-1-1	M	Define, document and approve are all documentary; requires sight of a strategy document and a record of its approval, neither of which is in scope.
1-1-2	M	Requires sight of a written IoT cybersecurity plan showing prioritized actions; documentation only, no device signal.
1-1-3	M	Two parts: defining KPIs and tracking them. Defining is management judgment. Tracking requires evidence that the organization monitors them, not that values exist.
1-1-4	M	Requires evidence that a review took place at planned intervals; no scan shows whether a review happened or when.
1-2-1	M	Define, document, approve and disseminate — all four need policy documents and distribution records.
1-2-2	M	Requires sight of written technical standards. The subject matter is technical but the requirement is that the standards exist and are documented.
1-2-3	M	Updating policies and standards involves human administrative review and legal/regulatory tracking.
1-3-1	M	Requires records showing which roles were assigned and to whom; organizational structure is not visible from any device.
1-3-2	M	Requires an assessment of whether people performed their assigned duties; no technical signal corresponds to this.
1-4-1	M	Implementing risk management practices relies on organizational frameworks rather than API calls.
1-4-2	M	Defining theoretical risk scenarios is a subjective, human-led analytical exercise with no observable output on a device.
1-4-3	M	Maintaining a risk register requires manual data entry and qualitative human evaluation.
1-4-4	M	Conducting risk assessments involves interpreting potential threats and calculating business impact.
1-4-5	M	Determining corporate risk appetite and appropriate mitigations requires business-level executive logic.
1-4-6	M	Requires evidence of periodic review against current regulations; neither the review nor the regulatory tracking is observable.
1-5-1	M	Secure-by-design principles are broad architectural practices requiring qualitative human review.
1-5-2	M	Verifying planning and design phase considerations requires reviewing project documentation.
1-5-3	M	Defining change management procedures is a purely administrative and documentation-based task.
1-6-1	M	Assessing compliance with legal frameworks inherently requires human legal and audit interpretation.
1-7-1	M	Reviewing implementation holistically requires human audit functions and subjective evaluation.
1-7-2	M	Independent third-party audits are inherently subjective, human-led validation exercises.
1-7-3	M	Defining non-compliance management processes and assigning roles is an administrative task.
1-8-1	M	Defining and approving HR requirements is a document-based policy creation task.
1-8-2	**H**	A script can check if an account is active, but human HR context is needed to verify if the person was actually terminated.
1-8-3	M	Periodically reviewing and updating policies is an administrative and legal compliance task.
1-9-1	M	Integrating IoT into a training strategy is a document-driven human resources activity.
1-9-2	M	Promoting awareness relies on human training and corporate communication methodologies.
Domain 2 — Cybersecurity Defense		
2-1-1	**H**	A scan builds the device list and reads hardware/software capabilities, but sensitivity, classification and third-party assets are not observable.
2-1-2	M	'Review periodically' implies a human audit and validation process against the documented inventory, generating no technical signal.
2-2-1	**H**	Access controls can be queried programmatically, but determining if the correct privileges are assigned requires business context.
2-2-2	**H**	Trying documented default pairs proves failure but not success — ten rejections does not prove no default exists. The periodic-password-change clause is also not observable.
2-2-3	M	The requirement is that a periodic review takes place, not that permissions are correct. Whether a review happened is a process fact.

ID	Class	Recorded justification
2-3-1	M	Defining, documenting, and approving email security policies is a purely document-based administrative task.
2-3-2	**H**	Encryption can be verified automatically, but ensuring it maps to organizational email protection rules requires context.
2-4-1	M	Defining and approving secure connectivity requirements is an administrative documentation task.
2-4-2	**H**	Peer authentication such as mTLS is scannable, but determining which peers should communicate requires a human baseline.
2-4-3	**H**	Encryption in transit is scannable, but 'secure the underlying infrastructure' is a broader requirement a device scan cannot reach.
2-4-4	**H**	Subnets and VLANs can be checked via code, but validating the segregation against a risk assessment is manual.
2-4-5	**H**	Gateway existence is scannable, but verifying it successfully secures all internet-facing connectivity requires human architectural review.
2-4-6	**H**	Reading an HTTPS update URL from firmware proves failure if absent, but not that the update server enforces TLS or that its authentication is adequate.
2-5-1	**H**	Communication between the IoT device and mobile devices is partly observable, but secure coding practices for the mobile app and secure erasure on loss are not.
2-6-1	M	Defining and documenting data classification and labeling schemes is a subjective organizational governance activity.
2-6-2	**H**	Transit protection is observable; data at rest inside device storage is not reachable without host access, so one half is unobservable.
2-6-3	**H**	Data flows can be tracked programmatically, but determining if collected personal data is 'not needed' requires legal/business judgment.
2-7-1	M	Documenting and approving alignment to a National Cryptographic Standard is a manual compliance task.
2-7-2	**H**	Packet capture and TLS checks settle the in-transit half. Data at rest inside device storage is out of scope with no host access.
2-8-1	M	Defining and approving backup policies requires human administrative governance.
2-8-2	**H**	A stored recovery image may be observable, but validating it as a 'trusted and tested' version requires human context.
2-8-3	M	The requirement is periodic review and testing of backups. Whether testing occurred on schedule is a process fact.
2-9-1	**H**	Vulnerability scanning is automated, but the 'mitigate' requirement often involves manual patching or human risk acceptance for IoT hardware.
2-9-2	**H**	Checking component versions for missing patches is automated, but the 'timely manner' aspect requires human procedural context.
2-10-1	M	Penetration testing is a subjective, methodology-driven human exercise, even when automated scanning tools are used.
2-10-2	M	Red team exercises require human-led methodology including social engineering and physical intrusion.
2-11-1	**H**	Log-forwarding configuration may be readable, but SOC monitoring, alerting on unauthorized change and a 12-month retention period are not observable.
2-11-2	M	Conducting regular examination of diagnostic information implies human analysts actively reviewing logs for incidents.
2-12-1	M	Developing an incident response plan is a documentation and strategy exercise requiring human input.
2-12-2	M	Establishing an incident management plan is purely a policy and documentation exercise.
2-12-3	M	Establishing a post-incident analysis capability requires heavy human analytical and forensic expertise.
2-12-4	M	Defining threat management requirements and sharing threat intelligence with the NCA are documentary and procedural.
2-13-1	M	Verifying physical detection systems such as cameras and badge readers requires on-site physical inspection.
2-13-2	M	Hardware tamper protection and detection require in-person physical inspection of the device to verify.
2-14-1	**H**	Interface security mechanisms can be scanned, but validating their effectiveness against unauthorized access requires contextual evaluation.
2-14-2	**H**	Execution restriction configurations such as allowlisting can be checked in code, but the list of authorized software must be defined by humans.
2-14-3	M	Secure development practice and source code review require access to source, which is outside the observable set.
2-14-4	M	The requirement is that the whitelist is kept updated over time. Whether updates happen periodically is a process fact.

ID	Class	Recorded justification
2-15-1	**H**	Root of trust and secure boot are not observable, but embedded certificate or private key material in the firmware image is — TEST-FW-CERTKEY maps here with a fail verdict. Mixing rule gives hybrid.
2-15-2	**H**	Two halves: defining installation steps is documentary, but applying secure configuration and hardening is observable. SA-IOT-003 maps here via TEST-NET-PORTSCAN.
2-15-3	M	Developing an end-of-life plan and withdrawal strategy requires organizational governance and documentation.
Domain 3 — Cybersecurity Resilience		
3-1-1	M	Defining and approving resiliency requirements within a Business Continuity Plan is a subjective governance exercise.
3-1-2	M	Whether a gateway enforces security during a connectivity outage cannot be established by passive observation.
Domain 4 — Third-Party and Cloud Computing Cybersecurity		
4-1-1	M	Defining security requirements for third parties is a legal and procurement documentation task.
4-1-2	M	Requesting vendor demonstrations and evaluating secure-by-design principles requires subjective human vendor risk assessment.
4-1-3	M	Requesting hardware and software component lists from vendors is an administrative supply chain task.
4-1-4	M	Incorporating supply chain components into organizational risk assessments is a subjective, human-led analytical exercise.
4-1-5	M	Conducting audits and verifying software certifications against contractual requirements relies heavily on human legal review.
4-1-6	M	Periodically reviewing third-party risk mitigation procedures is a manual compliance and audit task.
4-2-1	M	Defining and approving cloud security policies requires human administrative governance.
4-2-2	**H**	Encryption and authentication settings can be checked programmatically, but determining if they are 'adequate' requires human architectural context.
4-2-3	M	Assessing a cloud provider's overall cybersecurity posture against internal policies requires subjective human vendor risk review.
4-2-4	M	Establishing procedures for audits and multi-tenant risk management is an administrative policy creation task.
4-2-5	M	Contractual provisions for vendor-neutral data return on provider exit are a legal and procurement matter with no technical signal.

Appendix B Collector catalogue

Every observation the platform produces comes from one of these 32 collectors. Each returns structured observations; none writes the finding text a human signs.

Test identifier	What it checks	Tool
Network discovery		
TEST-NET-DISCOVERY	Network discovery (VLAN sweep)	nmap
Network and protocol		
TEST-DEVICE-ID	Device identification	curl
TEST-DEVICE-MAC-VENDOR	MAC address vendor lookup	python3
TEST-MDNS-PROBE	mDNS discovery	python3 (socket)
TEST-MODBUS-PROBE	Modbus TCP discovery	nmap
TEST-MONITORING-ENDPOINT	Diagnostic monitoring endpoint	curl
TEST-MQTT-OPEN	MQTT anonymous access	mosquitto_sub
TEST-NET-HTTP-INSPECT	HTTP inspection	curl
TEST-NET-PKTCAPTURE	Packet capture	tcpdump
TEST-NET-PORTSCAN	Nmap service/port scan	nmap
TEST-NET-REACHABILITY	Host reachability	python3 (socket)
TEST-PQC-TLS-HANDSHAKE	Post-quantum TLS readiness	python3 (openssl)
TEST-RTSP-PROBE	RTSP stream authentication	nmap
TEST-SECURITY-LOG-ENDPOINT	Security/access log endpoint	curl
TEST-TLS-CLIENT-AUTH	TLS peer authentication	python3 (openssl)

Test identifier	What it checks	Tool
TEST-TLS-CONFIG	TLS configuration	openssl
TEST-UPNP-PROBE	UPnP/SSDP discovery	python3 (socket)
Web and authentication		
TEST-ADMIN-UNAUTH	Unprotected administrative endpoint	curl
TEST-AUTH-ANON-ACCESS	Anonymous access	curl
TEST-AUTH-DEFAULT-CREDS	Default credentials	curl
TEST-AUTH-SESSION	Weak session behavior	curl
TEST-HTTP-HEADERS	HTTP security headers	curl
TEST-PHYSICAL-TAMPER-STATUS	Hardware tamper detection status	curl
Firmware analysis		
TEST-FW-APIKEY	API keys	yara
TEST-FW-CERTKEY	Certificate or key file	yara
TEST-FW-CONFIG	Configuration files	python3
TEST-FW-MANIFEST	Packet manifest	python3
TEST-FW-SECRETS	Hard-coded password or secrets	yara
TEST-FW-UPDATESCRIPT	Update script	python3
TEST-FW-VERSION	Version file	python3
TEST-PQC-FIRMWARE-CRYPTO	Post-quantum firmware crypto currency	python3
Device intelligence		
TEST-DEVICE-CVE-LOOKUP	Device CVE lookup (NVD, no firmware required)	python3

References

[1] National Cybersecurity Authority. *Cybersecurity Guidelines for Internet of Things (CGIoT-1:2024)*. Kingdom of Saudi Arabia, 2024. https://nca.gov.sa/en/regulatory-documents/guidelines-list/cglot/.

[2] O. Briliyant, A. Javed, and Y. Cherdantseva. "Systematization of human-centered continuous audit for IoT security compliance." *Journal of Cybersecurity*, 12(1), 2026. https://doi.org/10.1093/cybsec/tyag021.

[3] I. Oranekwu, L. Elluri, R. Yus, and A. Kotal. "Scalable automation for IoT cybersecurity compliance: Ontology-driven reasoning for real-time assessment." *Computers & Security*, 161:104711, 2026. https://doi.org/10.1016/j.cose.2025.104711.

[4] B. Saha and S. K. Shukla. "An Automated Framework for Cybersecurity Policy Compliance Assessment Against Security Control Standards." arXiv:2605.07515, 2026.

[5] I. Cindrić, M. Jurčević, and T. Hadjina. "Mapping of industrial IoT to IEC 62443 standards." *Sensors*, 25(3):728, 2025. https://doi.org/10.3390/s25030728.

[6] T. Alsahafi, W. Halboob, and J. Almuhtadi. "Compliance with Saudi NCA-ECC based on ISO/IEC 27001." *Technical Gazette*, 29(6):2090–2097, 2022.

[7] National Institute of Standards and Technology. *Open Security Controls Assessment Language (OSCAL)*. https://pages.nist.gov/OSCAL/.

[8] Anchore. *Grype: Vulnerability Scanner for Container Images and Filesystems*. https://github.com/anchore/grype.

[9] Cybersecurity and Infrastructure Security Agency. *Known Exploited Vulnerabilities Catalog*. https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[10] OWASP Foundation. *OWASP Internet of Things Project - IoT Top 10 (2018)*. https://owasp.org/www-project-internet-of-things/.

[11] M. Fagan, K. Megas, K. Scarfone, and M. Smith. *IoT Device Cybersecurity Capability Core Baseline*. NISTIR 8259A, 2020. https://doi.org/10.6028/NIST.IR.8259A.

[12] National Institute of Standards and Technology. *IoT Device Cybersecurity Guidance for the Federal Government: Establishing IoT Device Cybersecurity Requirements*. NIST SP 800-213, 2021. https://doi.org/10.6028/NIST.SP.800-213.

[13] M. Fagan et al. *IoT Non-Technical Supporting Capability Core Baseline*. NISTIR 8259B, 2021. https://doi.org/10.6028/NIST.IR.8259B.

[14] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. FIPS 203, 2024. https://doi.org/10.6028/NIST.FIPS.203.

[15] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard (ML-DSA)*. FIPS 204, 2024. https://doi.org/10.6028/NIST.FIPS.204.

[16] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard (SLH-DSA)*. FIPS 205, 2024. https://doi.org/10.6028/NIST.FIPS.205.

[17] National Institute of Standards and Technology. *Foundational Cybersecurity Activities for IoT Device Manufacturers*. NISTIR 8259 Rev. 1, 2026.

[18] National Institute of Standards and Technology. *IoT Device Cybersecurity Guidance for the Federal Government: IoT Device Cybersecurity Requirement Catalog*. NIST SP 800-213A, 2021.

[19] R. Roman, J. Zhou, and J. Lopez. "On the features and challenges of security and privacy in distributed Internet of Things." *Computer Networks*, 57(10):2266–2279, 2013. https://doi.org/10.1016/j.comnet.2012.12.018.

[20] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini. "Security, privacy and trust in Internet of Things: The road ahead." *Computer Networks*, 76:146–164, 2015. https://doi.org/10.1016/j.comnet.2014.11.008.

[21] A. Mosenia and N. K. Jha. "A Comprehensive Study of Security of Internet-of-Things." *IEEE Transactions on Emerging Topics in Computing*, 5(4):586–602, 2017. https://doi.org/10.1109/TETC.2016.2606384.

[22] M. A. Khan and K. Salah. "IoT security: Review, blockchain solutions, and open challenges." *Future Generation Computer Systems*, 82:395–411, 2018. https://doi.org/10.1016/j.future.

2017.11.022.

[23] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga. "A survey of intrusion detection in Internet of Things." *Journal of Network and Computer Applications*, 84:25–37, 2017. https://doi.org/10.1016/j.jnca.2017.02.009.

[24] S. Hajiheidari, K. Wakil, M. Badri, and N. Jafari Navimipour. "Intrusion detection systems in the Internet of Things: A comprehensive investigation." *Computer Networks*, 160:165–191, 2019. https://doi.org/10.1016/j.comnet.2019.05.014.

[25] M. Miettinen et al. "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT." In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2177–2184, 2017. https://doi.org/10.1109/ICDCS.2017.283.

[26] Y. Meidan et al. "Detection of Unauthorized IoT Devices Using Machine Learning Techniques." arXiv:1709.04647, 2017.

[27] Y. Meidan et al. "N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders." *IEEE Pervasive Computing*, 17(3):12–22, 2018. https://doi.org/10.1109/MPRV.2018.03367731.

[28] R. Doshi, N. Apthorpe, and N. Feamster. "Machine Learning DDoS Detection for Consumer Internet of Things Devices." In *2018 IEEE Security and Privacy Workshops*, pp. 29–35, 2018. https://doi.org/10.1109/SPW.2018.00013.

[29] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster. "IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2), Article 46, 2020. https://doi.org/10.1145/3397333.

[30] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. "A Large-Scale Analysis of the Security of Embedded Firmwares." In *23rd USENIX Security Symposium*, pp. 95–110, 2014.

[31] D. D. Chen, M. Egele, M. Woo, and D. Brumley. "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware." In *NDSS Symposium*, 2016. https://doi.org/10.14722/ndss.2016.23415.

[32] M. Kim et al. "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis." In *Annual Computer Security Applications Conference (ACSAC)*, pp. 733–745, 2020. https://doi.org/10.1145/3427228.3427294.

[33] E. Fernandes, J. Jung, and A. Prakash. "Security Analysis of Emerging Smart Home Applications." In *2016 IEEE Symposium on Security and Privacy*, pp. 636–654, 2016. https://doi.org/10.1109/SP.2016.44.

[34] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn. "IoT Goes Nuclear: Creating a ZigBee Chain Reaction." In *2017 IEEE Symposium on Security and Privacy*, pp. 195–212, 2017. https://doi.org/10.1109/SP.2017.14.

[35] M. Antonakakis et al. "Understanding the Mirai Botnet." In *26th USENIX Security Symposium*, pp. 1093–1110, 2017.
https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[36] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. "SoK: Security Evaluation of Home-Based IoT Deployments." In *2019 IEEE Symposium on Security and Privacy*, pp. 1362–1380, 2019. https://doi.org/10.1109/SP.2019.00013.