THREAT-LED AI PENTESTING

How the ELTON AI pipeline tests a medical device

And why device testing does not look like web-app autonomous pentesting.

KEY TAKEAWAYS

- Autonomous pentesting has arrived, but the public state of the art is built for web applications: an agent is pointed at a URL, explores from the outside, and reports what it can exploit. A medical device is firmware, an RTOS, service processes, proprietary protocols, physical interfaces, and clinical workflows, most of which an outside-in agent never reaches.
- ELTON runs the opposite direction. It builds a digital twin of the device first, threat models the twin with MITRE EMB3D, generates a test case for every threat pinned to a specific component, then runs a proprietary toolchain against the real device. The pipeline starts with ground truth, not reconnaissance.
- Every finding is exploitable, conditionally exploitable, an unexploitable weakness, or already mitigated, each backed by an executed test case and traced to the manufacturer's own documentation, in the report format that has been through more than 1,000 FDA submissions since 2013.
- The engineering edge is the toolchain, not the model. More than five agentic loops write and refine device-testing tools continuously, the tools are deterministic and reusable, and customer data never becomes a prompt.

1. The shift, and where it stops short

The cost of finding a vulnerability has collapsed. In 2024, published CVEs jumped 38 percent to roughly 40,000, the largest single-year increase on record, and the curve is projected past 70,000 by 2026. The reason is AI: frontier models now read code, reason about it, and write working exploits, and the skill floor for doing so has fallen through the floor.

The clearest public proof points set the bar and define the gap:

- **XBOW** reached number one on HackerOne's US leaderboard in mid-2025 with roughly 1,060 submitted reports over about 90 days, the first time an autonomous system outranked human researchers on that board. On its own 104-scenario benchmark it matched an experienced pentester's 85 percent, finishing in 28 minutes what took the human 40 hours.
- **Google's Big Sleep**, from Project Zero and DeepMind, found a real zero-day in SQLite (CVE-2025-6965, CVSS 7.2) before it was exploited in the wild, compressing detect-to-patch to about 48 hours.

- **Academic teams** have shown a single GPT-4 agent exploiting 87 percent of a set of one-day CVEs from their descriptions, and hierarchical agent teams extending that to zero-day web vulnerabilities.

Every one of those results is real, and every one is about software an agent can reach from the outside: a web app, a parsing library, a service with a described CVE. The honest reading of the 2025 and 2026 benchmarks, including the CVE-Bench zero-day setting where the best framework exploited only around 13 percent of real web-app vulnerabilities, is that autonomous discovery is genuinely competitive and genuinely narrow. Competitive on reachable software surfaces. Narrow everywhere else.

A medical device is everywhere else. The exploitable surface of an infusion pump, a patient monitor, an imaging console, or a dispensing cabinet is mostly not a web endpoint. It is a bootloader trust chain, a firmware update path, a proprietary service on a bench cable, an authentication scheme wired to a fleet certificate store, a real-time process that behaves one way in emulation and another on the board. Point a web-app agent at that and it maps the small part it can see and declares the rest out of scope. The scope was decided by what the agent could reach, which is exactly the failure mode a regulator now asks about.

2. Outside-in versus inside-out

The public autonomous-pentest architecture is consistent across vendors and papers. A recent systematization of 13 open-source frameworks describes them along six axes (agent architecture, plan, memory, execution, external knowledge, benchmark), and the commercial systems follow the same shape: an attack-surface mapper, a coordinator that prioritizes what to test, a fleet of short-lived parallel agents with an offensive toolkit, and a validator that reproduces the exploit to kill false positives.

It is a good architecture, and it is an outside-in architecture. It begins with no knowledge of the target and spends its first effort discovering the target, because for a public web app that is the only option. That single design choice is what does not transfer to medical devices, for three reasons:

- **The surface is not discoverable from outside.** A firmware update manifest trust gap does not appear in a port scan. You find it by reading the update path, which means you need the update path.
- **The verdict standard is different.** A web-app pentester proves exploitability to a security team. A manufacturer has to prove exploitability, or non-exploitability, to the FDA, with evidence a reviewer will accept. A proof-of-concept in a bug bounty does not satisfy FDA Section 524B. A structured, traceable finding with a dismissal rationale does.
- **The obligation is continuous.** A pentest is an event. Postmarket cybersecurity is a standing duty: every new CVE against every SBOM component has to be dispositioned for the life of the device.

ELTON is inside-out. It starts from a model of the device, threat models that model, tests against the threats, and keeps the model alive release after release.

Stage	Outside-in autonomous pentest	ELTON threat-led pipeline
Start	No prior context. Agents spend first effort on recon.	Digital twin pre-loads architecture, SBOM, interfaces, trust boundaries, countermeasures. Zero recon.
Scope	Whatever the agent can reach from outside.	Every component, interface, data flow, and asset in the twin, threat modeled with EMB3D.
Probe	Web endpoints, forms, APIs, DOM, HTTP.	Network, firmware, code, hardware, and clinical workflow, on the real device.
Verdict	Binary: exploitable or not.	Graduated: directly exploitable, conditionally exploitable, weakness, or mitigated, each with evidence.
Output	Pentest report. Engagement ends.	Living system of record. VEX, MDDT-qualified ratings, FDA-traceable evidence that compounds.

3. Stage one: the digital twin

Everything starts with the twin, and the twin is not a binary inventory rebuilt from a firmware dump. It is a model of the device's security architecture: the level at which exploitability is actually decided. It is assembled from artifacts the manufacturer's quality system already produces, supplemented by source, firmware, or runtime scanning where available.

- **Architecture and data flows.** System diagrams, data-flow maps, and deployment context define what the device is, what talks to it, and where it sits on a hospital network.
- **SBOM and components.** The bill of materials binds every component and version into the model, so a new CVE resolves to a real location instead of a keyword match against a product name.
- **Interfaces and countermeasures.** DICOM, HL7, MQTT and proprietary listeners, trust boundaries, authentication schemes, encryption, segmentation, and documented mitigations become claims the platform can test, not lines in a PDF.
- **Assets and trust levels.** What is worth reaching, and what stands between an entry point and it.

It answers before anyone touches hardware.

The twin is Level 1 of exploitability verification. Reachability and control analysis against the model typically resolves 30 to 40 percent of findings as Not Affected, each with reasoning a reviewer can follow, because a component that is present but architecturally unreachable is not a risk on this device regardless of its CVE score.

It compounds.

The twin is not rebuilt per engagement. Verified paths, confirmed behavior, and dependency structure feed back into it, assessment after assessment, release after release. A CVE that drops in year three lands in a model that already knows the component, the interface, and everything

between them. This is the structural opposite of an outside-in agent, which starts from zero every run because it has nowhere to keep what it learned.

4. Stage two: threat analysis, mapping the twin to threats

A twin is a description. To test it, ELTON turns it into a set of concrete, scoped threats using MITRE EMB3D, the knowledge base MITRE, Red Balloon Security, and Narf built specifically for embedded devices. EMB3D is the right base precisely because it is not a web-app model: it enumerates threats to firmware, bootloaders, processors, buses, and embedded network stacks, the surface a medical device actually has.

EMB3D is organized around three linked identifiers, and ELTON walks them in order:

- **Device Properties (PIDs).** What the device is: it has a bootloader, it accepts firmware updates, it exposes a network service, it stores credentials. The twin's components map to properties.
- **Threats (TIDs).** What can go wrong given those properties. EMB3D maps each threat to the properties that make it relevant, so a device only inherits the threats its own architecture admits.
- **Mitigations (MIDs).** The technical controls that address each threat: hardware root of trust, signed manifests, continuous attestation, and so on.

The mechanism is deterministic where it can be and AI-driven where it must be. Component types map to properties, properties map to threats, and ELTON extends both where EMB3D stops, because no published catalog covers every proprietary protocol or vendor-specific update scheme on a real device. The AI's job here is not to guess vulnerabilities. It is to reason from the twin and the catalog to the set of threats that actually apply to this device, and to add the device-specific threats the catalog does not yet name. Every threat that comes out is pinned to a specific component, which is what makes the next stage precise instead of speculative.

> **A worked example, representative of a firmware update path on a networked bedside device.** Property: the device includes a bootloader and accepts firmware over a mobile update path. Threat: an adversary bypasses the integrity-protection chain on that path, enabling unauthenticated firmware or manifest substitution, mapping to CWE-693, Protection Mechanism Failure. Applicable mitigations: hardware-backed bootloader authentication, hardware root of trust, and continuous integrity measurement with remote attestation. The threat is now a scoped, testable claim against a named component, with the controls that would refute it already identified. Nothing about it required guessing where to look.

5. Stage three: AI test case generation, per threat

For each pinned threat, ELTON generates test cases against exactly that scope, using the EMB3D property identifiers plus test logic tailored to the specific vulnerability and device architecture. A test case is not a prompt asking a model whether something is exploitable. It is a concrete procedure with a scope, a pass criterion, a fail criterion, and an execution target.

For the firmware example: the pass criterion is that extracted paths are canonicalized and remain under the upgrade base directory, and the manifest signer chains only to a pinned signing root with a code-signing EKU enforced. The fail criterion is any archive entry with traversal characters landing outside the base directory, or any certificate chaining to a broad fleet trust store accepted as a manifest signer. The execution target is the tier at which it runs: twin, code and firmware, or real hardware.

Because each test case descends from a threat that descends from a property of the twin, coverage is auditable in both directions. A reviewer can start from a component and trace forward to the threats and test cases that examined it, or start from a finding and trace back to the documentation it came from. That two-way traceability is the artifact FDA reviewers have started asking for, and it is the thing a findings list without a coverage record cannot produce. The same machinery validates a newly published CVE the day it appears, because a CVE is just another threat against a component the twin already models.

6. Stage four: the proprietary pipeline

Generating test cases is not the hard part. Running them against a real medical device is, and this is where the proprietary engineering lives. AI builds the tools. The pipeline does the testing.

The distinction ELTON draws is between a prompt and a toolchain. Pointing a general model at a device and asking it to find vulnerabilities is a prompt: it has no device context, it verifies nothing, it returns something different every time, and getting an answer meant handing your source code to a third-party model. A tool is deterministic. It runs the same way twice, against the real device, and produces evidence. ELTON uses AI to write and select the tools, then runs the tools.

The pipeline has five parts working together:

- **The twin**, providing pre-loaded context so no run starts from zero.
- **An orchestrator** that holds the global view, selects which tools and specialized agents to dispatch for a given threat and device, and applies deterministic logic to reconcile their results.
- **Specialized capabilities on the device**, short-lived and focused: network and protocol testing (DICOM, HL7, MQTT and proprietary services), firmware and binary analysis, authentication and session testing, SBOM cross-referencing, and clinical-workflow testing under adversarial conditions.
- **TestLink**, which carries the whole pipeline onto physical hardware over an out-of-band private 5G uplink, so a bench instrument reaches the ELTON lab without ever touching the enterprise network.
- **A human gate.** No finding is passed to the customer before a qualified tester (the team holds OSCP, OSCE3, OSWE, GPEN, GXPN, and GICSP among other credentials) has reviewed it.

The part that is genuinely hard to copy is the toolchain's growth. More than five agentic loops run continuously, every hour of every day, finding gaps in ELTON's own coverage and writing new tools to close them. Building a debugger for one stubborn process, or a man-in-the-middle for one proprietary protocol, used to consume an entire assessment budget. Now it takes minutes, the cost is paid once, and the tool is reused on every device afterward. A prompt starts from zero

every time it is opened. The toolchain does not, and it deepens on medical devices specifically, so the gap widens rather than closes.

The data boundary is structural, not a policy promise.

Customer source code, firmware, documents, and findings stay inside the pipeline. What goes to a model is ELTON's own engineering work: what a tool needs to do, and how a documented protocol behaves. The data path and the development path never meet. When a vendor says they use AI on your product, the question that matters is where your data goes, and if the answer is a general chat client, it left their control the moment they pressed send.

This is also the sharpest architectural contrast with the outside-in systems. A web-app validator re-exploits a finding in a controlled environment to confirm it is real, which is the correct move for a web app and a genuine advance over scanners. ELTON does something different in kind: it does not only confirm the positives, it manufactures the negatives, generating and executing the test that lets a manufacturer defensibly dismiss the 99 percent of findings that are not exploitable on their device. Confirming a true positive helps a security team. Evidencing a true negative is what a regulator requires, and it is where most of a manufacturer's cost actually sits.

> **"Autonomous web-app pentesters point a model at a target and prove what they can reach. ELTON builds the device first, threat models it, tests every threat with tools the AI writes and a human reviews, and proves both what is exploitable and what is not, in evidence a regulator will accept."**

7. Stage five: verification, and the distinction that matters

Every test case runs at the deepest tier the manufacturer has opened, and the tier determines the evidentiary weight of the result:

- **Level 1, digital twin.** Reachability and control analysis against the model. Typically clears 30 to 40 percent of findings as Not Affected before anyone touches the device.
- **Level 2, code and firmware.** Custom harnesses built from the provided code, static analysis of whether vulnerable functions are actually called, firmware emulation where possible, cross-architecture builds. Typically 50 to 65 percent Not Affected, because implementation reveals what architecture alone cannot.
- **Level 3, real hardware.** On-device exploitation through TestLink, in the device's real runtime with its real peripherals and real-time constraints. Typically 70 to 85 percent Not Affected, the strongest evidence there is: an executed test case, pass or fail, on the actual product.

The tiers create an incentive that runs in the manufacturer's favor. Deeper access does not surface more alarms; it dismisses more findings with stronger evidence, which is why the Not Affected rate climbs from L1 to L3. Knowing more is only a burden if everything gets called a vulnerability.

That is why the vulnerability-versus-weakness distinction is enforced, not cosmetic. A vulnerability is exploitable today: reachable from the real attack surface and substantiated by a test case

executed against the target, and worth a developer's time. A weakness is latent: real, but with no path on this device today, tracked and shipped dismissed with its reasoning, and re-evaluated automatically as the product changes. It does not flood the backlog as if it were urgent.

And it is why the dependency graph sits on top of verification. ELTON rates each vulnerability in isolation, then computes an attack graph over the twin: entry vectors produce conditions, findings require them. A low-severity information leak, plus a reachable authentication bypass, plus a memory-safety bug is not three medium problems. It is one critical path. The graph models the whole path, scores the chain the way CVSSv4 subsequent-system impact intends, and identifies the single root fix that collapses the most exploitable severity. Hundreds of findings distill to a handful of root causes.

8. What comes out

The pipeline produces evidence as a byproduct of doing the work, not as a separate reporting project:

- **A finding in one of four states**, each with an executed test case behind it: directly exploitable, conditionally exploitable, unexploitable weakness, or mitigated weakness.
- **A device-contextual rating** using the MITRE Rubric for Applying CVSS to Medical Devices, qualified under FDA's MDDT program (Q171974), for CVSSv3, with CVSSv4 automation. A generic NVD 9.8 becomes a defensible 2.4 when a documented TLS control removes the network path, and the reasoning is attached.
- **VEX output** in CycloneDX, machine-readable, generated for every disposition.
- **An unbroken traceability chain** from CVE or threat to component to test case to result to rating to VEX status, in the report format this team has carried through more than 1,000 FDA submissions since 2013.

13 yrs testing medical devices, since 2013	**1,000+** devices tested	**10,000+** device tests run
1,000+ FDA submissions	**6 of 10** of the top medical device manufacturers	**2 days** to first findings

About ELTON. ELTON Cyber delivers agentic medical device pentesting and exploitability management as one subscription. AI finds every vulnerability, verification eliminates the noise, and the fix that ships fastest goes straight to your developers. Built by a team behind more than 1,000 FDA submissions and 1,000 devices tested since 2013, serving 6 of the top 10 medical device manufacturers.
eltoncyber.com

Sources for the external comparisons in sections 1 and 2: XBOW platform and benchmark materials (xbow.com); HackerOne leaderboard reporting; Google Project Zero, From Naptime to Big Sleep, and the Big Sleep SQLite disclosures (projectzero.google, thehackernews.com); LLM Agents can Autonomously Exploit One-day Vulnerabilities and Teams of LLM

Agents can Exploit Zero-Day Vulnerabilities, Fang, Bindu, Gupta, Zhan and Kang, University of Illinois (arxiv.org/abs/2406.01637); Hackers or Hallucinators, a systematization of LLM-based automated penetration testing (arxiv.org/abs/2604.05719); and MITRE EMB3D (emb3d.mitre.org).