ISSUE 2 · AI VULN DISCOVERY HARNESS ARCHITECTURE

# Chatbot or Vulnerability Discovery Tool?

Issue 2: How AI discovers vulnerabilities

## KEY TAKEAWAYS

- The model is not the tool; the harness is the tool. The model provides reasoning and the harness turns reasoning into work against a real target, deciding which tools the model can call, what context it sees, when to spawn subagents, when to compact a filling context window, where to write findings to durable memory, and how to recover from failure.
- When someone says they are 'using AI' for vulnerability discovery, the question is not which model (models are interchangeable) but what their harness looks like; if they cannot draw that picture they have a chatbot with a prompt, like owning MS Word does not make you an author.
- The Claude Code harness leak (via the Claude Agent SDK and circulated system-prompt and tool-definition leaks) exposed not a prompt but a recipe: the exact tool surface, subagent spawning with narrower context, memory compaction, todo/plan files as durable state, delegation logic, handoff patterns, retry logic, and orchestrator goal-holding.
- Once the scaffolding pattern was public it was portable to any frontier model, giving everyone the blueprint and breaking the offensive cost curve because the hard part, the harness, became reference material.
- Medtech needs harnesses, plural: a code-scanning harness is a weekend project, but a real-device harness must drive physical interfaces (USB, Bluetooth Low Energy, proprietary wireless, IR), attach a debugger to a process on real hardware or an RTOS, and coordinate static firmware analysis with live hardware behavior in real time.
- A concrete five-agent run on a network-connected infusion pump: the scanner reads code and lists candidates, the process agent attaches a debugger and confirms which are reachable, the runtime agent speaks the management-port protocol and produces crashes, the exploit agent turns a crash into a PoC or documents why it cannot, and the validation agent assembles the evidence package and re-runs the PoC from a clean image.

> **"A finding without reproducible evidence is an opinion."**

## The Claude Code harness leak, and why it mattered more than people realized

The press treated the leak as a curiosity, but what got exposed was a recipe, not a prompt: the tool surface, subagent patterns, compaction, durable state, delegation, handoffs, retries, and the orchestrator's goal-holding. That recipe was the moat, and once public it became portable to any frontier model, breaking the offensive cost curve.

## Why medtech needs harnesses, plural, and why they look different

A code-scanning harness over a filesystem of source is a weekend build. A medical device harness is not: it must drive physical interfaces, attach debuggers to processes on real hardware or RTOSes that cannot be emulated, and coordinate static firmware analysis with live hardware behavior in real time. Teams that spent a decade testing devices by hand built these harnesses over the last couple of years because it was the only way to scale.

## Real Example: the infusion pump and its five agents

A tester kicks off a run with a one-paragraph goal (find an exploitable vulnerability reachable from the management network with a working PoC and evidence). The scanner reads code and produces file/function/line candidates; the process agent attaches a debugger and kills most of them by testing reachability; the runtime agent speaks the protocol and crafts inputs until it gets crashes; the exploit agent tries to turn a crash into a working PoC or records why it cannot; and the validation agent assembles and re-runs the evidence package from a clean device image.

## How the harness holds it all together

No agent shares a context window with the others. They communicate through short structured handoffs and a shared filesystem holding the run's long memory, the crash corpus, coverage map, findings list, and notes. When a context fills the harness writes state, compacts, and re-reads; the orchestrator watches budget and progress and can spawn a narrower instance or kill a line of investigation.

## What to take away

Ask which agents exist, what tools each has, how they hand off, what durable state they keep, whether the harness can drive a physical device or only read source, and what the validation agent produces. If a vendor, consultant, or tool cannot draw that picture, they have a chatbot with a prompt, and the gap between that and a real harness is the entire story.

Five specialized agents: scanner, process agent, runtime agent, exploit agent, validation agent.

Worked example candidate: 'Function parse_dicom_header at firmware/svc/dicom.c:412 reads a 16-bit length field from the network and passes it directly to memcpy without bounds checking.'

Reachability funnel: of 47 scanner candidates, only 6 were reachable on the running firmware.

Concrete exploit detail: length field at offset 0x4A reached only when magic byte 0x73 is set in the header.

Example negative result: reachable crash blocked by a stack canary, primitive is denial of service, not RCE.

Physical interfaces a device harness must drive: USB, Bluetooth Low Energy, proprietary wireless, sometimes IR; protocols 'designed in 2008 and never updated.'

**About ELTON.** ELTON is the leading AI vulnerability discovery and cybersecurity verification platform for medical devices. ELTON builds a digital twin of each device, continuously identifies new vulnerabilities across the full stack, and uses AI to determine which are actually exploitable, with every finding and every dismissal evidenced for regulatory review. ELTON meets FDA premarket (524B) and postmarket cybersecurity vulnerability testing and management requirements. Read the newsletter and talk to us at **eltoncyber.com**.