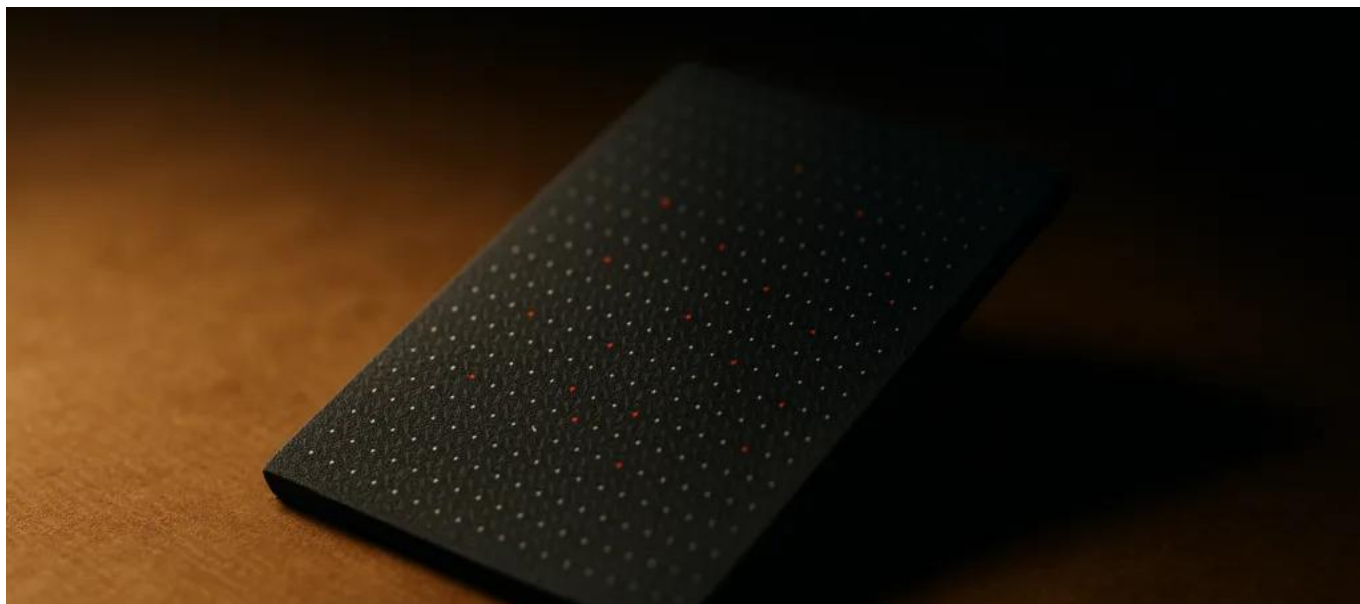




How to Wire Strix and Autonomous Remediation Into CI/CD Without Letting an AI Push Its Own Patch

What share of last year's CVEs actually got exploited? Two percent. Edgescan counted 40,009 CVEs published in 2024 and 768 reported as exploited in the wild.

THE SHORT VERSION

Of 40,009 CVEs published in 2024, Edgescan reports only 768 (about 2%) were publicly reported as exploited in the wild. Agentic testers like Strix validate findings with working proof-of-concept exploits, the filter queue-based scanning never had. Wire the finding and the exploit artifact into CI/CD, and keep the patch behind a human merge gate, especially before the EU Cyber Resilience Act's exploited-vulnerability reporting duty starts on 11 September 2026.



Forty thousand CVEs, 768 that mattered

The numbers from the [Edgescan 2025 Vulnerability Statistics Report](#) describe a triage problem rather than a detection problem. 40,009 CVEs published in 2024, a record annual total. 768 of them, roughly 2%, publicly reported as exploited in the wild, which was itself a 20% increase over 2023. CISA's Known Exploited Vulnerabilities catalog closed 2024 with 1,238 entries, 185 of them added that year.

Set that against remediation speed. Fluid Attacks, in its vendor-produced [State of Attacks 2025](#) dataset, puts overall mean-time-to-remediate at roughly 55 days, down from 68 the year before. Thirteen days of improvement is real work, and it is irrelevant if those 55 days go to findings nobody will ever exploit. A severity score is a prediction. A working exploit is a measurement.

That is the argument behind the agentic shift the publisher's brief (dated 2026-10-10) describes: security moving from passive scanning to active testing, where AI agents find a vulnerability and then validate it by exploiting it. [Strix](#), published on GitHub by usestrix and described in the brief as "AI hackers for your apps," is the open-source entry point. On the remediation side, the brief names three harnesses that act on findings instead of filing them: Microsoft's MDASH, a multi-model agentic security system that, per Microsoft's Security Blog post of 12 May 2026, [tops a leading industry benchmark](#); Harness, which per [Unite.AI's report](#) ships agents that scan, triage and patch "at machine speed"; and Wiz, named in the brief alongside the other two as a harness that triages and remediates autonomously.

My read, labelled as opinion: those three are grouped together in the brief, but they are not interchangeable inside a build pipeline. Finding-and-proving carries a different risk profile from patch-and-merge. The first produces evidence a human can check in minutes. The second produces a code change whose correctness you can only establish by reading it. I would adopt them on different timelines, and I say why below.

Why exploit validation beats a severity score

It converts a probabilistic claim into a falsifiable one. A CVSS 9.8 in your dependency tree



tells you the flaw is severe in some configuration. It does not tell you whether your authentication layer, your WAF rules, or the simple fact that the vulnerable code path is never reached make it unreachable in your deployment. An agent that produces a working proof-of-concept against your running application has answered that question empirically, for your build, on that day.

The regulatory side makes this concrete. Regulation (EU) 2024/2847, the Cyber Resilience Act, starts obliging manufacturers to report actively exploited vulnerabilities and severe incidents on 11 September 2026, with the main obligations following on 11 December 2027. The ceiling for breaches of essential requirements or the vulnerability-reporting duties reaches €15 million or 2.5% of worldwide annual turnover. The reportable category is “actively exploited,” not “we have a critical finding.” Switzerland has had a sharper version of this since 1 April 2025: under the revised Information Security Act, critical-infrastructure operators must report cyberattacks to BACS within 24 hours of discovery, with 14 days to complete the report. Twenty-four hours leaves no room to argue about whether a finding is real.

A severity score is a prediction. A working exploit is a measurement.

My reading, and this is inference rather than something any regulator has written down: a defensible internal record of which findings were proven exploitable, and when, will be worth more under both regimes than a complete scanner log. Both frameworks reward the ability to tell an exploited condition from a theoretical one, under time pressure. An agent that stores a reproducible PoC alongside the finding builds that record as a side effect.

Six steps to wire this in without handing over merge rights

The sequence below assumes you already run CI on pull requests and have some form of ephemeral or staging environment. Strix is open source, so the acquisition cost is your engineering time plus whatever model inference the agent consumes. The brief and the repository do not publish a pricing figure, and I will not invent one. The commercial harnesses (MDASH, Harness, Wiz) have costs I have no sourced numbers for, so treat



procurement as a separate exercise.

1. **Run the agent against a disposable environment, never production.** An agent that validates by exploitation is, by construction, attacking something. Point it at an ephemeral deployment of the PR branch with seeded test data. Blast radius is only half the reason: a PoC against production is itself an incident you may have to report, and in Switzerland you would have 24 hours to decide whether it was.
2. **Gate on exploit-validated findings only, at first.** Fail the build when the agent produces a reproducible PoC. Log everything else as informational. If you gate on the agent's unvalidated suspicions you have rebuilt the 400-ticket queue with a higher inference bill. The 2% exploitation rate is why this filter is worth more than the raw finding count.
3. **Store the PoC as a build artifact with a hash and a timestamp.** The exploit is the evidence. Attach it to the finding, version it, and keep it after the fix ships so you can replay it as a regression test. This is also the artifact you want when a CRA or BACS clock starts and someone asks when you first knew a vulnerability was exploitable.
4. **Let the remediation harness open a pull request, not merge one.** MDASH, Harness and Wiz are described in the brief as triaging and remediating autonomously. Scope that autonomy to proposing a diff. The agent writes the patch, the humans review and merge. A patch merged without review is an unreviewed code change in a security-sensitive path, which is the thing you were trying to prevent.
5. **Require the agent to prove its own fix.** Re-run the stored PoC against the patched branch and attach the failed-exploit result to the PR. This is the highest-value piece of automation in the chain, because it turns "the model says it fixed it" into a replayable negative result a reviewer can verify in one command.
6. **Treat the agent's own tooling as attack surface.** These agents read repositories, call tools, and write code. Prompt injection via untrusted repository content is a live class of problem in AI coding agents, as the eight [GitSpawn flaws Manifold Security found across seven agents](#) showed, with four still unpatched at the time of that writeup. Restrict credentials, isolate the runner, and consider a tool-call filter in front of it.

Autonomy creep, shrinking dashboards and the silence of



a clean run

! The autonomy creep

Every harness vendor's roadmap points toward merging without review, because that is where the time savings live. The first time an agent patches a flaw at 02:00 and the build goes green, someone will propose removing the merge gate for "low-risk" changes. The category "low-risk security patch" does not survive contact with a dependency bump that quietly changes an API contract. Keep the gate and accept the slower cycle.

A second failure is quieter. Exploit validation produces fewer findings, and fewer findings look like a weaker security program to anyone reading a dashboard. If your board or your auditor is used to scanner volume, dropping from hundreds of criticals to a handful of proven exploits reads as a regression. Reframe the metric before you change the tooling. The number to show is proven-exploitable findings and their time-to-fix against the roughly 55-day baseline Fluid Attacks reports.

The third one is about what a clean run means. A validated exploit proves a vulnerability is real. The absence of one proves only that this agent, with this configuration, in this environment, did not find a path on this run. Business-logic flaws, multi-step authorization bypasses and anything requiring domain knowledge of your data model are places where an agent's silence means nothing. There is no published figure for Strix's false-negative rate that I know of, and until someone publishes one, treating a clean agent run as an all-clear is unsupported.

When I would not wire this in yet

If you do not have a staging environment that can be stood up and destroyed per pull request, stop here and build that first. An exploitation agent without a disposable target is a liability, and the alternative (running it against shared staging with real data copies) creates exactly the incident-reporting ambiguity the Swiss 24-hour rule punishes.

If your team cannot review a security patch competently, an agent that writes them does not fix that. It moves the problem from "we have no patch" to "we have a patch nobody can evaluate," which is worse because it feels solved. Autonomous remediation assumes a reviewer on the other side.



And if you ship a product that falls outside the CRA's scope and outside Swiss critical-infrastructure obligations, the compliance argument above does not apply to you. The 2% exploitation rate still does. Your reason for adopting this is triage efficiency, not the 11 September 2026 deadline, and you should be honest with yourself about which one drives the budget.

One more gap worth naming: the market estimate the brief cites, \$1.6B in 2025 growing to \$9.8B by 2034 at a 21.3% CAGR, comes from MarketIntel (22 March 2026) and is a third-party projection, not an audited figure. I would not use it to justify anything. The Edgescan numbers and the regulatory dates carry the argument.

Start with Strix on one service, buy the harness later

Start with Strix, because it is open source and the failure mode of a finding agent is a wasted afternoon rather than a bad merge. Give it a disposable environment and one service, ideally your most externally exposed one. Measure two things over six weeks: how many exploit-validated findings it produces, and how that compares to your current critical count. If the ratio looks anything like 768 out of 40,009, you have found where your engineering hours were going.

Only then look at the commercial harnesses. MDASH, Harness and Wiz solve the second half of the problem, and the second half is worth automating only once the first half produces a signal you trust. Buying a patch-generation agent to feed it unvalidated scanner output is paying for faster movement in an unknown direction.

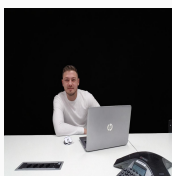
One thing I cannot resolve from the available data. The brief groups MDASH, Harness and Wiz as peers, but the only comparative evidence in front of me is Microsoft's own statement that MDASH tops a leading industry benchmark, published on Microsoft's security blog. That is a first-party claim on an unnamed benchmark. It may well be accurate. It is not a basis for choosing between the three, and anyone telling you otherwise is reading a press release as a bake-off. If you are evaluating these, insist on running them against your own stored PoCs from step three. You will have the regression suite by then anyway.



How to Wire Strix and Autonomous Remediation Into CI/CD Without Letting an AI Push Its Own Patch

WHAT TO DO WITH THIS

Of 40,009 CVEs in 2024, Edgescan reports 768 were seen exploited. Exploit validation is the filter that turns that ratio into working hours you get back, and the CRA's 11 September 2026 reporting duty makes a record of proven-exploitable findings worth keeping. Run the finder, gate the patcher. [Strix on GitHub →](#)



WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →