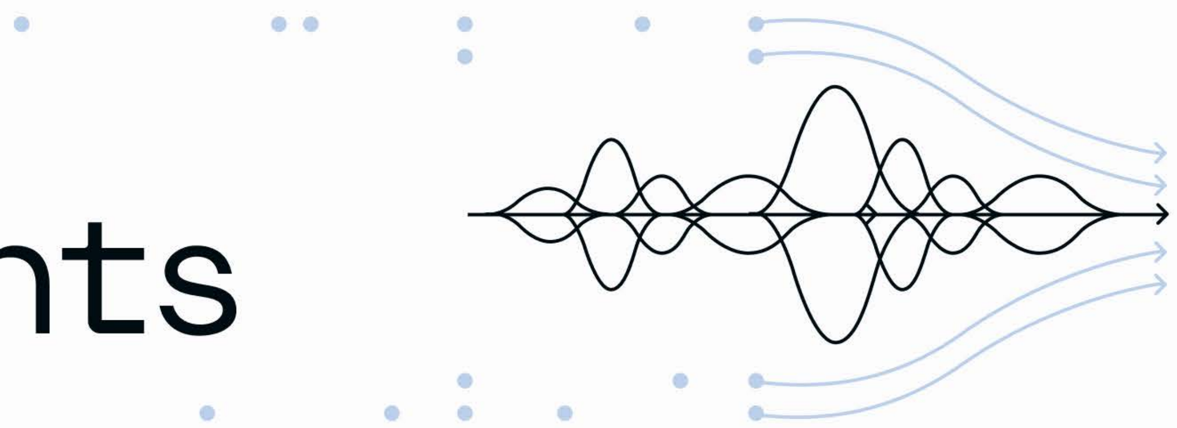
XBOW

How Autonomous Pentesting Works at Machine Speed, Without False Positives

Scaling Autonomous Exploitation with AI:
Inside XBOW's Autonomous Pentesting Architecture

Table of Contents



The Challenge:
Too Many Bugs,Too Much Noise

Application security teams face a daunting paradox. Modern apps (and now AI-generated code) hide countless zero-day vulnerabilities, yet traditional scanning tools bury teams in false positives and tedious triage. Running one vulnerability scanner after another often yields an endless **deluge of alerts**, many of them misleading or low-value. Meanwhile, real exploits slip through unnoticed.

For platform and DevOps leaders, this isn't just a numbers problem. Rather, it's an operational bottleneck. Every hour spent sifting out false alarms is an hour not spent fixing real exposures. And as AI accelerates development cycles, that gap only widens. What's needed is a new approach that finds the needle (the real exploitable bug) without the haystack of noise. Enter XBOW, an autonomous AI-driven penetration testing system designed to do exactly that. It has already identified over 200 zero-day vulnerabilities **with zero false positives**, proving that offensive security can scale without drowning in noise.

From AI Noise
to AI-Driven Results

Early attempts to apply AI (like naïvely prompting an LLM to "find vulns in this code") often created more problems than they solved. Sure, large language models would highlight possible issues, but they also hallucinated plenty of ghosts: phantom vulnerabilities that didn't truly exist. Security teams found themselves staring at long lists of "findings" with no clear way to tell real risks from mirages. In short, AI without guardrails traded one form of noise for another.

XBOW takes a very different path. It leverages AI agents not as unchecked vulnerability spotters, but as autonomous attackers that think and act like expert pentesters. The key distinction is that **every finding is proven through exploitation**. Instead of handing you a guess, XBOW attempts the exploit and validates the outcome so you only get a report when something is genuinely exploitable. No hypothetical "maybe" vulns. Instead, only confirmed, repeatable proof.

This approach flips the script on false positives. By building **exploit validation into the AI's workflow**, XBOW can let the AI run wild generating attack ideas, without creating extra work for humans. The agent may try dozens of potential attacks, but you only hear about the ones that succeeded. The result is an AI-driven pentest that produces real, actionable results (and nothing else). Platform leaders can scale security testing to hundreds of apps knowing that **no amount of AI "creativity" will burden their team with junk findings.**

Inside XBOW:
How Autonomous Offense Works

How did XBOW achieve the "holy grail" of security testing, and combine breadth of automation with depth of accuracy? It comes down to a clever blend of techniques, each augmenting the other:

Static + Dynamic in Sync

XBOW marries static code analysis (SAST) with dynamic testing (DAST) in one loop. It starts by reading the target's source code (if available) or observing its interface to map out an attack surface. This static insight gives the AI agent a head start; it knows where vulnerabilities might lurk and what patterns (SQL queries, file paths, auth logic, etc.) to probe. From there, it launches dynamic attacks against the running application, but with the precision of having read the "instructions" first. This fusion means XBOW can find subtle bugs that pure black-box scanners miss, while drastically cutting down the trial-and-error time. As one security researcher quipped, it's like having a hacker that **reads the source code to plan the break-in, then executes it live**, all in minutes and without human intervention.

Knowledge-Fueled Attacks

The system's AI isn't limited to the code in front of it; it's been trained on a wealth of prior vulnerability data, exploits, and security knowledge. This context becomes fuel for hypothesis generation. For example, when testing Apache Druid, XBOW "recalled" an old SSRF issue in a similar endpoint and **formed a hunch** that a new SSRF might exist in the Druid console's proxy feature. That intuition paid off, and the agent quickly discovered a brand-new SSRF zero-day (later assigned CVE-2025-27888) by trying variants of the known attack pattern until one slipped through. This knowledge-driven approach lets XBOW **turn past lessons into new exploits.** It's continually asking: "Have we seen a bug like this before? Could it exist here in a new form?" In practice, this means more zero-days found, faster. The AI isn't rediscovering everything from scratch, it's building on a collective memory of hack techniques.

Continuous Self-Reflection

Like a good human pentester, XBOW iterates and learns as it goes. Early attack fails? The agent pauses, **reviews the application's feedback**, and adjusts strategy. In one case, testing a known vulnerable app, XBOW tried a simple SQL injection payload and got no visible result. Instead of flagging a false positive or giving up, it dug into the source code, realized the injection was happening but needed a different syntax to exploit (due to how the query was structured), and then crafted a new payload that successfully triggered a time-based SQLi. This ability to **"read between the lines" and pivot** is what sets agentic testing apart from one-and-done scanners. XBOW essentially debugs its own attacks in real-time by combining what it sees at runtime with what it learned from static analysis and prior knowledge.

Built-In Exploit Validation

The final piece that eliminates false positives is the exploit validation engine. Every time the AI suspects a vulnerability (say, an open redirect or a file path traversal), it doesn't trust the initial test alone. Instead, it runs a **secondary verification** using a specialized method tailored to that flaw. For instance, if XBOW's agent thinks a Jenkins endpoint might be vulnerable to open redirect, it will follow up by sending a crafted URL and checking if the response actually **redirects to an external site** (confirming the issue). If the agent finds a possible blind SQL injection, it will automatically perform a time-delay test (e.g., SLEEP(5)) to see if the response is 5 seconds slower. This is a clear sign the injection is real. Only when a validator returns positive proof does XBOW mark the vulnerability as found. This means security teams get **zero "it might be vulnerable" guesses** – they get a proven exploit scenario every time.

The diagram below illustrates how these pieces come together in XBOW's architecture and workflow:

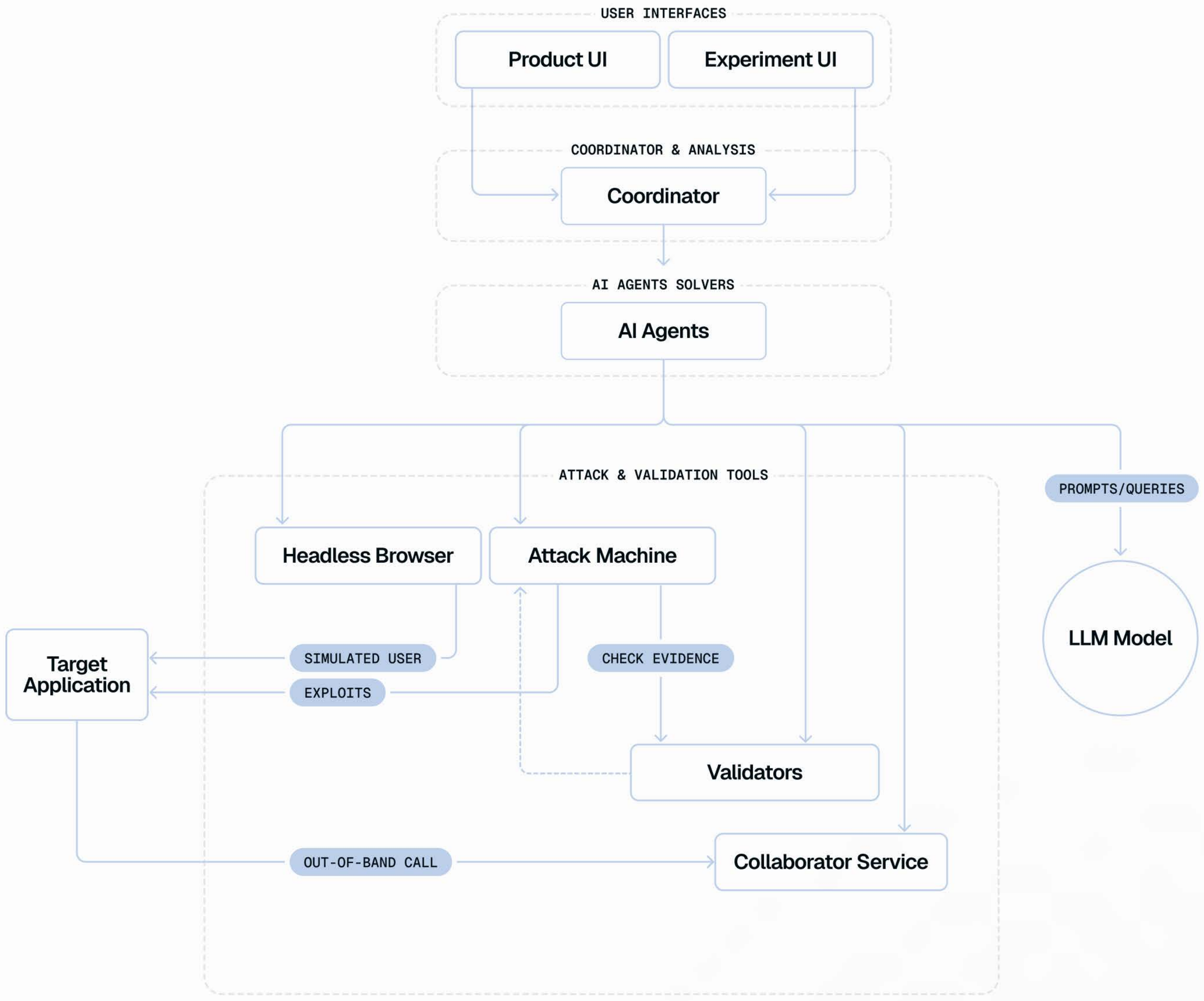


Figure: XBOW's autonomous pentesting workflow. The AI agent combines static code insights with dynamic attacks in a loop of hypothesis and testing. Every suspected finding goes through an exploit validator to ensure only real vulnerabilities are reported.

In practice, this loop runs at machine speed. XBOW can perform an in-depth security assessment from code review to live exploit in a matter of hours or even minutes, depending on the target. And it does so consistently, without fatigue or oversight. **The output is a high-fidelity report** that security engineers can trust instantly: each item is backed by evidence (like a working payload or captured response) showing the exploit in action. No more "vuln or not?" guessing game – XBOW provides the proof.

To appreciate how this looks in the real world, let's explore a few examples of XBOW's discoveries. →→→

XBOW in Action: Hunting Zero-Days with Precision

XBOW's methodology has been battle-tested across thousands of applications and containers. Here are a few highlights that show how it blends technical savvy with creative hacking all on its own:

Jenkins Open Redirect - from minor bug to major find

During one engagement, XBOW noticed a Jenkins endpoint that reflected a URL parameter in a 302 redirect. A trivial detail, but the agent zeroed in on it. It hypothesized an **open redirect** (where an attacker can redirect users to a malicious site) and verified this by instructing Jenkins to redirect to an external domain. The validation module confirmed it: Jenkins would indeed follow the attacker's URL. What a human might have overlooked as a "small issue," XBOW escalated into a concrete finding, complete with a demo link. The takeaway: the AI agent's thoroughness meant even a subtle misconfiguration turned into an actionable result, with **no doubt about its impact** (the proof was the redirected request).

Apache Druid SSRF - using history to make history

Apache Druid is a complex data system, but XBOW approached it like a seasoned hacker with a library of tricks. Remembering past SSRF flaws in similar platforms, the agent systematically tried various endpoints for server-side request forgery. Initial attempts at known vulnerable paths yielded nothing (the target had those locked down). Undeterred, XBOW tried the Druid console's proxy endpoint – an educated guess guided by documentation and prior knowledge. **Bingo.** It triggered an unexpected internal call, confirming a brand-new SSRF vulnerability that allowed access to internal network resources. This was later logged as CVE-2025-27888. Notably, every step was autonomous: XBOW formed the hypothesis from old CVEs, executed the attack, then validated it by seeing the internal response. It proved the SSRF by actually fetching an internal URL through the Druid server, demonstrating a textbook example of AI turning a hunch into a zero-day with zero human intervention.

GroupDocs Path Traversal via JWT - chaining the unthinkable

In another case, XBOW tackled a cloud document management app (GroupDocs) and uncovered a clever chain. First, static analysis of the app's code (and config) revealed that JSON Web Tokens (JWTs) were used for authentication, and the token verification code was loading keys from a file path constructed partly from the kid header of the JWT. This rang alarm bells for the agent: could an attacker supply a kid value that tricks the system into loading an arbitrary file (classic path traversal)? XBOW forged a JWT with a malicious kid (../../../../etc/passwd as part of the path, for example) and indeed got the system to load a sensitive file instead of a valid key. The validator component kicked in by observing the response and server behavior, confirming that the file contents were exposed (or the signature bypassed). In effect, XBOW **combined an authentication bypass with a directory traversal** in one go, demonstrating how an attacker could steal secrets or elevate access in GroupDocs. The finding was not just a hypothetical combo – XBOW provided the actual forged token and the evidence of unauthorized file access, making it an undeniable critical vuln.

Each of these cases shows a pattern: XBOW's agent operates like an expert security researcher at superhuman speed. It picks up one clue (a redirect here, a file path there), **relentlessly pursues the trail**, and doesn't stop at "this looks risky;" it goes all the way to "owned it." And because it validates every exploit, security teams don't have to second-guess the results. There are no false positives hiding among these gems – just genuine vulnerabilities that teams can prioritize immediately.

Results:
No-Noise Pentesting at Scale

The marriage of AI automation with rigorous validation isn't just a lab experiment – it's delivering real value in the field. Using XBOW, organizations have been able to run broad offensive security campaigns that were previously impractical. In one initiative, the system autonomously tested **thousands of containerized apps pulled from public repositories,** uncovering over 200 zero-day vulnerabilities (with corresponding CVEs assigned) across a spectrum of projects. Astonishingly, not a single false positive was reported out of those findings. Every one of those 200+ issues was real and required a fix. This level of signal-to-noise is unheard of in traditional AppSec tooling.

From a security leadership perspective, this shifts the paradigm. Instead of drowning in scanner output and spending days triaging, teams can focus on remediation and strategic improvements. It's the difference between **needle-in-haystack hunting** and a targeted treasure map of confirmed exploits. When an autonomous agent can comb through your entire portfolio and hand you a shortlist of verified critical bugs, the efficiency gains are dramatic. We're talking about compressing what might be months of manual pentesting (or endless scan-review cycles) into a scalable, continuous process, and without hiring an army of experts or burdening developers with false alarms.

Equally important, this approach fosters a more proactive security culture. Developers and platform engineers start to see security findings as actionable and trustworthy, not as "cry wolf" scanner noise. That builds confidence and accelerates fix cycles. When a developer gets an XBOW report showing exactly how a sensitive file was read via a path traversal, there's little debate, and it's a clear mandate to fix the issue. The clarity and credibility of results turn security from a nagging gatekeeper into a **collaborative problem-solver** that the team can rally behind.

Finally, autonomous pentesting addresses the scale problem head-on. Traditional security testing (with humans in the loop at every step) simply can't keep up with the explosion of apps, microservices, and updates in today's environments. By applying AI agents that never tire, always learn, and **validate as they go**, organizations can achieve coverage at a scale that was previously out of reach. You can schedule on-demand pentests for every build, spin up nightly security regressions, or test every third-party service before integration, all without adding headcount or clogging pipelines. Security testing becomes as continuous as development, but intelligently so, filtering its own output.

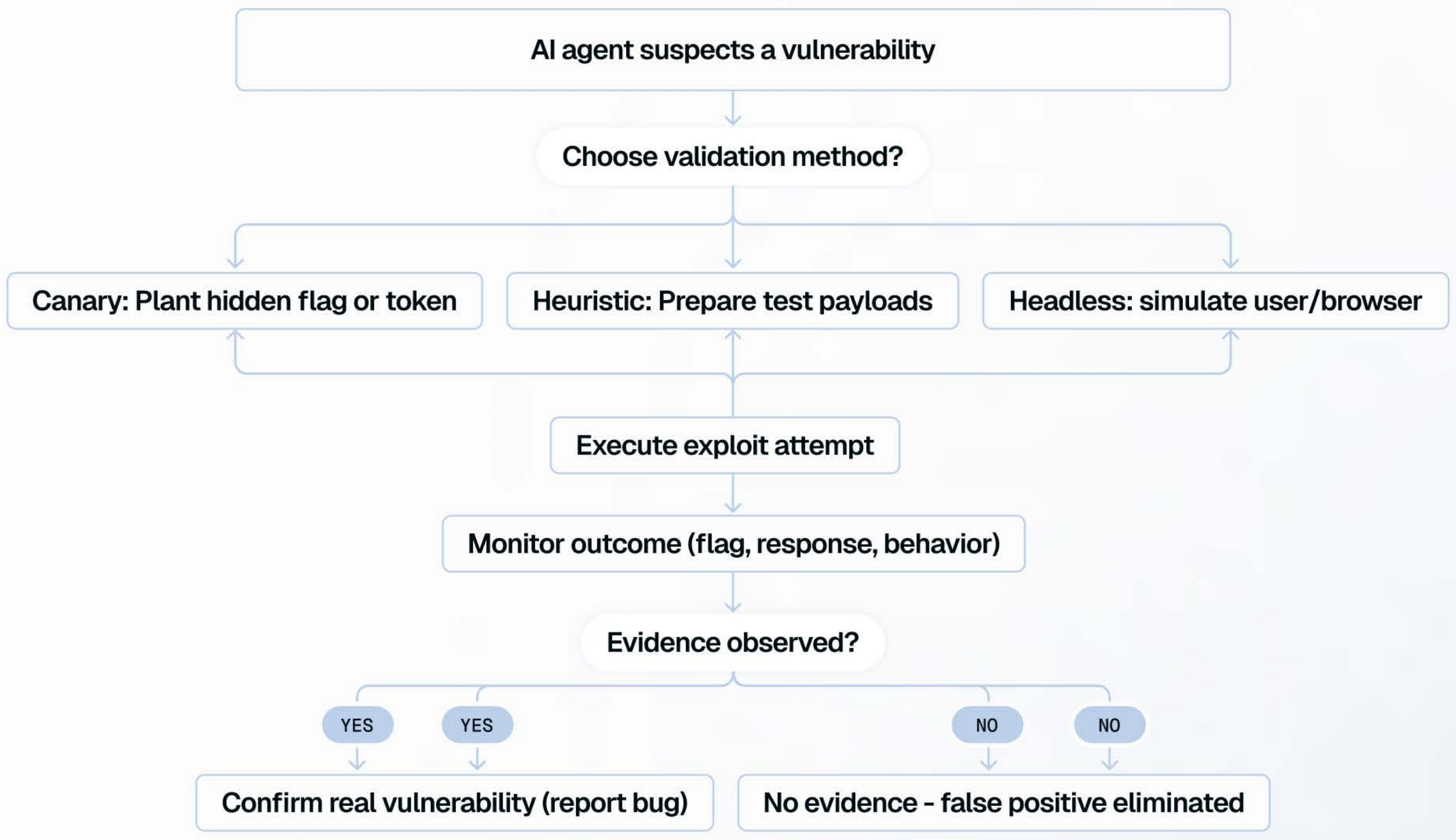

AI agent suspects a vulnerability
Choose validation method?
Canary: Plant hidden flag or token
Heuristic: Prepare test payloads
Headless: simulate user/browser
Execute exploit attempt
Monitor outcome (flag, response, behavior)
Evidence observed?
YES
YES
NO
NO
Confirm real vulnerability (report bug)
No evidence - false positive eliminated

Figure: Example autonomous validation loop in XBOW.

Conclusion:
Autonomous, Accurate, and Actionable

For security teams embracing AI-driven offensive testing, this new reality is reshaping how real vulnerabilities are found and fixed at scale without the drag of false positives. By letting an autonomous system handle the heavy lifting of vulnerability discovery and verification, organizations can **dramatically amplify their security coverage** without drowning in noise. This frees up human experts to focus on the vulnerabilities that truly matter, the creative fixes, and the strategic improvements in architecture and policy that harden systems long term.

When every result is vetted and reproducible, security leaders can champion these findings to developers and executives with hard evidence ("Here's how the exploit works"). It turns security from a guessing game into a data-driven conversation about risk and remediation. For platform owners, it means you can run continuous security assessments on your CI/CD and **not slow down delivery.** So, the AI is doing its thing in the background and only validated issues surface for your attention. No pipeline drag, no wild goose chases.

In a landscape where attackers (including their own malicious AIs) are moving faster than ever, XBOW demonstrates that defenders can too. Autonomous pentesting offers a way to **continuously probe your attack surface with creativity and rigor**, finding the cracks before adversaries do, and doing so at the scale of modern infrastructure. It's an empowering shift: instead of playing catch-up with endless alerts, your security team gets to go on offense, informed by an unfaltering AI ally that never sleeps and never sends a false alarm.

The era of AI-augmented security is here. By blending machine precision with the ingenuity of seasoned hackers, we finally have a path to **zero false positives** without missing the needles in the haystack. 200 zero-days later with XBOW, one thing is clear: when you arm an AI with the right knowledge, let it think like an attacker, and demand proof for every finding, you end up with an offensive security engine that is **fast, accurate, and battle-ready.**

What you can do next:

Turn proven exploits into fewer false alarms, faster fixes, and security that keeps pace with delivery, while giving you defensible security decisions and confidence at the executive level. Your developers – and your CISO – will thank you for the clear, impactful results.

It's time to put this security engine to work and reclaim the advantage in application security:
Speak with an XBOW security expert now.