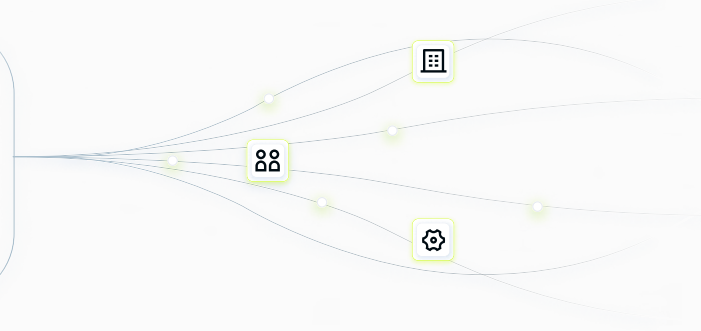
EXECUTIVE BRIEF

The Build-Versus-Buy Decision

for Autonomous Offensive Security



Building an internal AI-powered offensive security tool can seem appealing, especially for teams that want more control over data, models, vendors, cost, and compliance. Those are valid reasons to consider building.

But before attempting to build their own, teams should be clear-eyed about what they are actually taking on. An LLM is not the same thing as a penetration testing platform. A model can help identify possible vulnerabilities, reason through attack paths, and review context. But a production-grade security tool must also validate findings, enforce safety boundaries, manage runtime environments, integrate with workflows, control costs, and earn trust from both security and engineering teams.

10 Key Questions to Ask Before Building

01 How will we handle ongoing maintenance?

An internal AI security platform is not a one-time build. It requires ongoing ownership of infrastructure, models, prompts, evaluations, safety controls, attack tooling, browser infrastructure, sandboxing, credentials, logs, storage, and integrations.

- Who owns this after the first version ships?
- Who updates it when models change?
- Who maintains infrastructure as applications evolve?
- Who handles false positives, auth failures, safety exceptions, and support requests?
- What happens when the original builders leave or move teams?

02 How will we keep testing safe?

Autonomous offensive security systems are goal-directed. If safety is not built into the execution layer, the system may test the wrong asset, cross scope, create unnecessary load, or take actions that are unsafe in production.

- How is scope enforced?
- What techniques are allowed or blocked?
- How does the system know when to stop?
- Is there an audit trail of what happened and why?
- Can we prove activity stayed within authorized boundaries?

03 How much will it cost?

Model costs can grow quickly when testing requires deep reasoning, tool use, long context, repeated attempts, browser interaction, and validation. Even if token prices fall, inefficient agent behavior can create unnecessary spend.

- How will we estimate and monitor model token usage?
- How will we make sure model token usage is efficient?
- How will we prevent agents from repeating work?
- Can we route different tasks to different models?
- How will we control costs across many applications?

04 How will we validate findings are real?

AI can generate a large number of possible findings. Without strong validation, it can create more work for AppSec and developers instead of reducing risk.

- How will the system reproduce and prove exploitability?
- What evidence will it collect?
- How will it remove duplicates and false positives?
- How will it distinguish suspicious behavior from confirmed risk?
- Will developers trust the output enough to act on it?

05 Are we solving a tool problem or a platform problem?

A prototype may show that an LLM can find interesting issues. That doesn't mean it will operate across many applications, teams, environments, and edge cases.

- Will this work across our full application portfolio?
- Can it handle authentication, sessions, MFA, permissions, APIs, rate limits, and fragile workflows?
- What happens when an assessment fails?
- Who debugs failures across the model, orchestration layer, environment, and application?

06 Can we orchestrate agents at scale?

Launching agents is easy compared with coordinating them. At enterprise scale, the system must prevent duplicated work, preserve shared state, track coverage, assign tasks, and redirect effort intelligently.

- How will agents avoid testing the same paths repeatedly?
- How will the system track what has already been tested?
- How will it manage parallel assessments across many applications?
- How will it control compute costs and avoid wasted exploration?

07 How will model progress be managed?

Better models can improve testing, but they also create new work. Each model change may require evaluation, routing, regression testing, validation updates, governance reviews, and workflow changes.

- Are we locked into one model provider?
- Can we use different models for different tasks?
- When new models are released, will we be able to switch to take advantage of the progress?
- What breaks when model behavior changes?

08 Do we have the right headcount and expertise?

A serious AI security tool requires a team to build and maintain it. It's important to consider costs and expertise for proper headcount planning.

- Do we have the right mix of offensive security, AI, platform, and infrastructure expertise?
- Who will maintain the system when applications, browsers, login flows, and models change?
- Who will investigate false positives, failed assessments, authentication issues, and safety exceptions?
- What other security work will this team not be doing because they are maintaining this platform?

09 Will results fit into existing workflows?

A tool only reduces risk if findings reach the people and systems responsible for remediation. If output becomes a PDF or manual handoff, the bottleneck remains.

- Can findings flow into ticketing, CI/CD, dashboards, and developer workflows?
- Do results include evidence, impact, reproduction steps, severity, ownership, and remediation guidance?
- Can remediation be tracked and verified?
- Will this shorten the path from discovery to fix?

10 What governance do we need?

Offensive security tools need strong access control, auditability, data protection, and policy enforcement.

- What controls do we need in place (e.g., Who can launch assessments? Against which environments? Under what approvals?)
- How do we ensure findings and sensitive data have secure storage and correct access permissions?
- How are permissions, retention, audit logs, and encryption handled?

XBOW: Autonomous Offensive Security Testing, Built For Enterprise Trust

XBOW gives customers the best of both worlds: the power of frontier models applied to offensive security, with the control, governance, and operational maturity enterprises require. Customers can benefit from autonomous testing without inheriting the full burden of building and maintaining the platform internally.

XBOW gets better over time because it learns from the work: which attack methods are effective, which checks are reliable, how to stay safe, how to manage changing models, and what kind of evidence security teams trust.

SCHEDULE A DEMO