XBOW

WHITE PAPER

Autonomous Offensive Security Testing, Built for Enterprise Trust

How XBOW turns frontier model capability into governed, validated offensive-security execution

Frontier models like Mythos and GPT-5.5 have sparked a wave of conversation across the security industry, raising an understandable question:

If an organization can point a powerful language model at an application and unearth security findings, is it effectively running a penetration test?

Homegrown systems can create the appearance of autonomous pentesting very quickly. A motivated engineer can wire a model to a browser, connect it to a few tools, point it at a web application, and produce something that finds real issues. This highlights that offensive AI is real, and that frontier models are changing the way vulnerabilities are found, by both attackers and defenders.

The problem is, attackers only need to find one exploitable vulnerability. Defenders need to find all of them.

A penetration test needs to be methodical and comprehensive, with consistent planning and logic, from scoping and reconnaissance to report writing. The nature of LLMs, which excel at pattern matching, make them great at finding vulnerabilities, but their design makes them less ideal for deep, sustained, and safe testing.

XBOW was designed to harness AI's incredible strengths, while also providing teams with what they need from an enterprise security tool: predictability, safety, and trust.

Frontier Models Will Find Vulnerabilities, Then Stop

Frontier models are already very good at important pieces of offensive security work. They can:

- **Recognize** patterns across large volumes of application behavior, HTML, source code, screenshots, logs, and tool output.
- **Generate** and refine payloads quickly, especially when they receive feedback from a failed attempt.
- **Summarize** technical findings, explain impact, and turn messy evidence into readable remediation guidance.

However, a penetration test is a sustained, adaptive investigation across an attack surface, not a single clever exploitation attempt.

It requires coverage, prioritization, authentication handling, safe execution, reproducible validation, and the discipline to keep testing after the first promising lead appears. This is where raw LLMs start to break down.

LLMs are trained to produce plausible next answers, and guessing is rewarded over uncertainty. In the context of penetration testing, this means they often give up too easily and are overconfident in their own work.

Additionally, a major issue with LLLM-driven security testing is that the model may prove a vulnerability in a way that harms the target system. A careless system might access, download, or modify production database records, rather than safely proving exploitability, such as with a harmless sleep command.

Testing Has a Cost, Especially When It's Ungoverned

While LLM costs are falling, high-capability inference is still not cheap, especially when the task requires the kind of reasoning, tool use, context retention, and persistence needed for vulnerability assessment.

Security is a discipline of tradeoffs. Teams are given finite budgets and asked to reduce as much risk as possible with the resources they have. Every dollar spent on testing one path is a dollar not spent testing another.

A system that relies on LLMs alone, without deliberate control over scope and execution, can quickly become inefficient. It may burn tokens revisiting the same behavior, chasing low-value leads, exploring application areas that are out of scope, or spending too much time on paths that do not materially improve coverage.

LLMs will find vulnerabilities, but won't produce complete and sustained testing. They:

- Are great at recognizing patterns.
- Can generate and refine payloads quickly.
- Excel at summarizing information.
- Give up too easily.
- Are too confident in their work.
- Don't operate safely without guardrails.

Ungoverned Testing Is Risky

Ungoverned AI testing also creates safety risk. A system that is not explicitly constrained by scope, policy, and execution controls can wander into unauthorized areas, apply techniques that are inappropriate for the target, generate unnecessary load, or take actions that disrupt production systems. In offensive security, capability without guardrails is a liability. Enterprise testing requires systems that can pursue findings aggressively while still respecting boundaries, preserving stability, and ensuring that every action is authorized, auditable, and aligned with the customer's intent.

XBOW

A Governed Offensive Security System

XBOW is a platform designed to help enterprise organizations scale offensive security. Enterprise testing requires the ability to assess large, complex applications end-to-end; to run many assessments in parallel; to sustain testing without constant human intervention; and to do all of this within clearly defined boundaries.

A platform built for this environment has to be:

- Reliable under load
- Disciplined about scope
- Methodical enough to be both efficient and thorough
- Safe enough to operate continuously across real enterprise environments
- Able to sustain progress through the practical hurdles that define real testing, including authentication, session management, and complex workflows

Frontier models provide extraordinary reasoning and execution capability, but raw model capability is only one part of the system. XBOW wraps that capability in a governed control plane that manages scope, orchestrates testing, validates findings, preserves evidence, and keeps execution aligned with enterprise policy.

How an XBOW Test Works

An XBOW assessment begins when a customer defines the target, scope, and objectives through the platform or API. Customers can also provide useful context, such as authentication details, source code, documentation, prior pentest reports, threat models, known-risk areas, or findings from other security tools.

XBOW then maps the attack surface, identifying entry points, workflows, roles, authentication flows, APIs, and trust boundaries. This reconnaissance model evolves throughout the test as agents discover new behavior.

From there, XBOW coordinates specialized agents across the application. Different agents focus on different tasks or vulnerability classes, such as discovery, authentication, injection, access control, business logic, validation, or safety monitoring. They execute real offensive-security techniques in parallel, share important context, and adapt as the application responds.

Before anything is reported, XBOW validates exploitability through controlled, reproducible checks. Where possible, validation is deterministic. For more contextual issues, such as business logic flaws or IDORs, XBOW validates findings against the threat model built during reconnaissance.

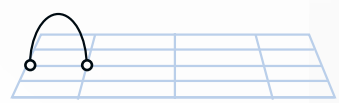
XBOW delivers validated findings with reproducible evidence, impact explanation, and remediation guidance that security and development teams can act on.

Bringing in Context

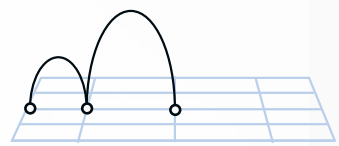
Human penetration testers do better work when they understand the system they are testing. The same is true for autonomous testing.

XBOW can incorporate context supplied by the customer, including documentation, source code, previous penetration test reports, architectural diagrams, threat models, business-priority instructions, and findings from other security tools such as SAST, SCA, API security, or cloud-security platforms.

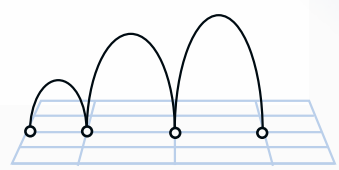
First, it improves prioritization. If the customer knows which systems handle regulated data, which workflows are business-critical, or which areas have produced incidents in the past, XBOW can use that knowledge to focus effort where risk matters most.



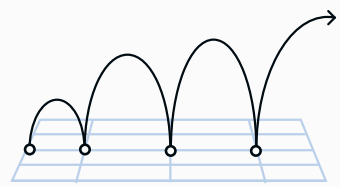
Second, it improves reconnaissance. Documentation and source code can help agents understand intended behavior, hidden routes, API structure, authorization patterns, and role boundaries faster than black-box exploration alone.



Third, it enables variant analysis. A SAST finding, a prior pentest result, a bug bounty write-up, or an internal incident report may describe a known weakness. XBOW can use that clue as inspiration to search dynamically for related exploitable variants in the running application. Static findings often describe potential risk, but exploit validation proves whether that risk is reachable and meaningful in practice.



Fourth, it improves remediation. When findings are grounded in application context, developers receive sharper reports. They get proof of exploitability, affected workflows, impact, reproduction steps, and guidance that maps more directly to how the system actually works.



The more relevant context the system receives, the more precise its testing and reporting can become.

Coordination and Orchestration: The Control Plane for Offensive AI

What separates XBOW from a prototype is that it orchestrates AI-driven testing into a sustained, disciplined process: coordinating agents, managing coverage, and allocating effort toward the highest-value paths.

At the center of the system is a coordinator.

The coordinator manages the overall testing plan, decides where effort should be spent, and adapts priorities as new information emerges. If one area of the application appears low-risk or well-protected, the coordinator can shift effort elsewhere. If another area shows signs of sensitive data exposure, unusual authorization behavior, or high-value business logic, the coordinator can direct more agent activity toward that area. Because XBOW has been trained by real expert pentesters, its AI will mirror pentester judgment and prioritize effort where it is most likely to uncover meaningful risk.

Security testing is an economic decision. Not every application receives the same level of effort, and not every part of an application deserves equal attention. A crown-jewel application may justify a longer, deeper assessment. A lower-risk application may require faster, broader coverage. XBOW's coordinator helps ensure that the available test budget is used to maximize meaningful coverage and validated risk reduction.

Coordination also makes long-running assessments possible. Some tests may run for hours. Large or complex applications may require slower, more careful testing over days or weeks. Maintaining control, context, and precision over that duration is a hard systems problem. Agents need to avoid duplicating work, stay within scope, preserve useful state, and continue progressing after early success. The coordinator gives the system the persistence and discipline that the raw model lacks.

Many Models

The XBOW platform is designed to use multiple frontier models in combination, selecting or alternating models based on task fit, availability, cost, speed, performance, and safety characteristics.

Models have different strengths. A model that performs best at exploit crafting may not be the best at logging in to a complicated enterprise application. A model that is strong at reasoning through business logic may not be the best independent judge of whether a proposed action is safe.

XBOW uses this diversity deliberately. In some cases, the system can select the model best suited to a task. In others, XBOW can use what the team calls an "alloy" approach: combining models from different families to counterbalance each other's weaknesses. The effect is similar to pair programming. One model may drift toward a poor assumption; another model, trained differently and behaving differently, may pull the trajectory back on course.

This is especially important in offensive security because overconfidence is dangerous. A model can convince itself it has succeeded. A second model with different behavior may be more skeptical. A model can propose an action that appears useful but is unsafe. A separate judge model can evaluate that action from a different perspective.

Validation: From Interesting Signal to Proven Exploit

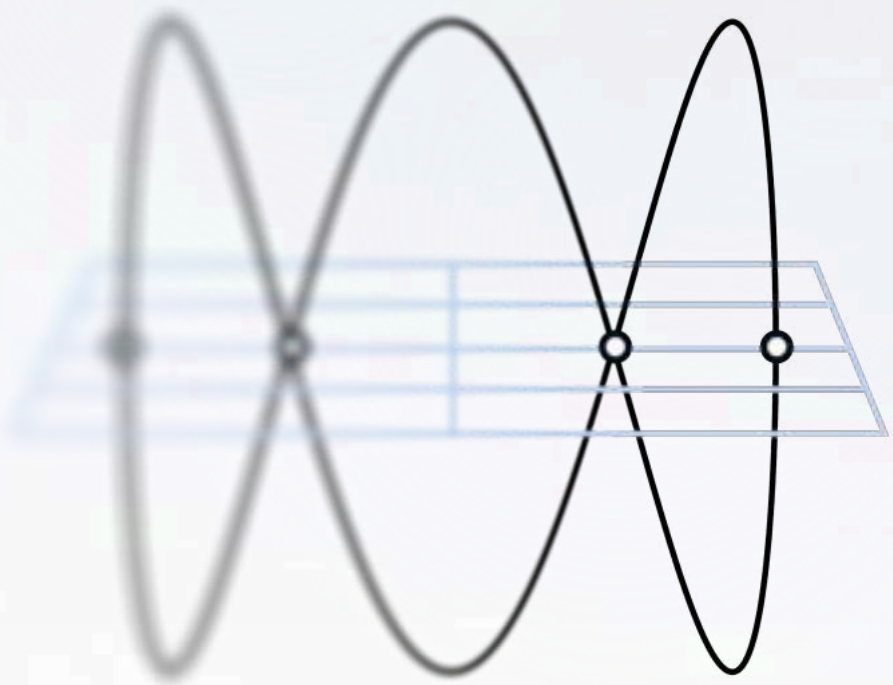
Security teams do not need more noisy findings. They need evidence.

One of XBOW's most important responsibilities is to **separate "this looks suspicious" from "this is exploitable."** The platform is designed to surface validated vulnerabilities, not speculative model output.

Where possible, XBOW validates with deterministic checks. That means using code and repeatable procedures, not model confidence, to verify that an exploit works. If an agent believes it has found a vulnerability, XBOW attempts to reproduce the exploit in a controlled way and preserve evidence that the behavior is real.

For some vulnerability classes, deterministic validation is straightforward. A SQL injection, for example, can often be validated through safe timing behavior rather than extracting or modifying data. Other vulnerability classes require more contextual reasoning. Business logic flaws, authorization issues, and IDORs often depend on what the application is supposed to allow. In those cases, XBOW's reconnaissance process builds a threat model that helps the system evaluate whether observed behavior represents a true security violation.

Validation is where XBOW's approach becomes especially valuable for developers. A report that says "this endpoint may be vulnerable" creates triage work. A report that shows the exploit path, affected role, reproducible evidence, impact, and remediation guidance creates action.



Safety: Proving Exploitability Without Harming the Target

Autonomous offensive security has to be held to a high standard. Human penetration testers already operate under rules of engagement. Autonomous systems need even stronger guardrails because they can execute at scale, in parallel, and for long periods of time.

XBOW's safety model is layered.

LAYER 1

The first layer is safe exploitation technique. XBOW is designed to prove exploitability without taking destructive actions.

LAYER 2

The second layer is independent action review. XBOW uses a separate guardian model to evaluate whether a proposed action is safe before execution. While the same model proposing an action may be biased toward approving its own plan, a separate judge gives the system an independent check.

LAYER 3

The third layer is target-health monitoring. XBOW observes the application during testing for signs of stress: degraded response times, failed logins, instability, or other indicators that the system may be affected by testing activity. If the target appears unhealthy, agents can back off and allow the system to recover.

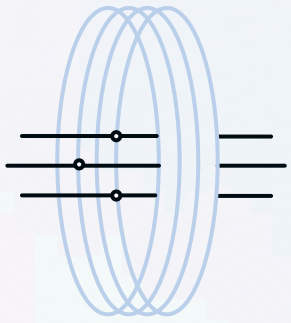
LAYER 4

The fourth layer is orchestration. Safety should go beyond blocking individual dangerous commands to control concurrency, pacing, scope, and the interaction of many agents against the same environment. XBOW's coordinator manages agent behavior so that parallel testing remains deliberate rather than chaotic.

LAYER 5

XBOW also adds a fifth layer: network layer access controls and traffic inspection via proxies.

No serious security practitioner should claim that autonomous testing eliminates all risk. The right standard is continuous risk reduction: safer exploit techniques, independent review, runtime monitoring, clear scope enforcement, and a product architecture designed for real production environments.



Enterprise Deployment Flexibility

Enterprise security teams have different requirements for data handling, residency, model access, and operational control. A deployment model that works for a fast-moving SaaS company may not satisfy a global bank, healthcare organization, or government contractor.

For many organizations, a standard SaaS deployment provides the fastest path to value. For customers with regional requirements, XBOW can support in-region deployment patterns. For organizations that require stronger isolation, single-tenant deployments provide dedicated environments. For highly regulated organizations, XBOW can support self-hosted deployment where application data, findings, and inference paths remain under the customer's control.

Customers may also choose how model access is handled. Some may use XBOW-managed model access. Others may use their own provider agreements, zero-data-retention configurations, or private infrastructure such as models available through enterprise cloud platforms.

XBOW's Expertise: The Harness Around the Models

AI is only as strong as the data it learns from, the context it receives, and the expertise used to shape it.

XBOW's leadership and technical roots play a key role in ensuring XBOW delivers truly expert-level pentesting. The team includes people who helped build major developer-security and AI-assisted development products, including GitHub Advanced Security and GitHub Copilot. That background shaped XBOW's view of the market: model capability improves quickly, but turning that capability into a reliable product requires systems engineering, evaluation, workflow design, and a deep understanding of how developers and security teams actually work.

XBOW also brings offensive-security expertise into the agent layer. Specialized agents are not generic prompts. XBOW hires worldclass hackers and vulnerability researchers and encodes their practices into XBOW. They are built to pursue specific vulnerability classes, handle practical testing obstacles, and validate results in ways that developers can trust.

The platform also improves through operational learning. **As XBOW runs more assessments across diverse applications,** the system accumulates knowledge about which strategies work, which paths waste effort, which validation techniques are reliable, and how to better coordinate agents across complex environments. This creates a compounding advantage: better playbooks, validation, prioritization, and safety controls over time.

The Future: Continuous, Validated Security

Beyond just a faster version of today's manual test, autonomous penetration testing changes the operating model for application security.

As software development becomes more AI-assisted, applications will change faster. Code will be generated more quickly, release cycles will compress, and traditional periodic testing will struggle to keep up. In that world, runtime validation becomes more important.

Static tools, design documents, threat models, and prior findings can provide valuable context. But the decisive question is whether a vulnerability is exploitable in the running application. XBOW is built around that principle: use every available signal to guide testing, but validate risk through controlled execution.

Over time, this creates a feedback loop. Context guides testing. Testing produces validated findings. Findings inform remediation. Remediation can be verified. Lessons from each assessment improve future assessments. Eventually, this points toward a more continuous model of application security: applications are tested as they evolve, findings are proven before they reach developers, and security teams gain assurance from persistent, governed offensive validation, rather than a single point-in-time report.

Key Takeaway: From AI Models To Governed Offensive Security

Frontier models are changing offensive security. They can reason, adapt, generate payloads, and recognize vulnerability patterns in ways that would have seemed impossible only a few years ago. But model capability alone is not a penetration test.

A real penetration test requires persistence, scope discipline, coverage, authentication handling, prioritization, validation, safety, reporting, and governance. It requires a system that does not stop after the first plausible success, proof that findings are real, and controls that allow testing to run safely in enterprise environments.

XBOW turns frontier model capability into governed offensive-security execution by:

- ↳ Coordinating specialized agents across complex attack surfaces
- ↳ Using multiple models rather than depending on one
- ↳ Bringing in customer context to improve reconnaissance and prioritization
- ↳ Validating exploits before reporting them
- ↳ Layering safety controls around autonomous execution
- ↳ Giving enterprises deployment options that match their governance needs

And it is built by a team with deep experience in AI, developer security, and offensive security systems.

That is the difference between pointing a model at an application and running an autonomous penetration test that an enterprise can trust.

To learn more about how XBOW is adding orchestration and validation to autonomous offensive security, schedule a demo.

SCHEDULE A DEMO