FireCompass

FIRECOMPASS TECHNICAL WHITE PAPER

Why LLMs Are Not Enough for Enterprise-Grade Pentesting

Using harness engineering to bridge the gap.

A frontier model can reason about an attack. It cannot execute one. This paper explains why agentic AI pentesting needs more than a capable model, and how harness engineering turns model intelligence into validated, safe, auditable penetration testing.

Published by FireCompass | Version 2.0 | June 2026

1 Executive Summary

Agentic AI pentesting is being sold on the strength of the model. The pitch is that a capable enough large language model, pointed at your environment, can now find and prove vulnerabilities on its own. It cannot, and the reason has almost nothing to do with how smart the model is. The gap is everything the model does not do, and closing that gap is an engineering problem.

A frontier large language model is raw intelligence. It reasons about targets, generates exploit strategies, and drafts attack narratives with real skill. What it cannot do is act. It cannot send an HTTP request, hold state across a hundred-step attack chain, enforce a scope boundary, or produce a tamper-evident audit log. A penetration test is not a reasoning exercise. It is a live interaction with production infrastructure, and plausibility is not proof. This is why an LLM alone, however capable, is not enough for enterprise-grade pentesting.

Harness engineering is the discipline that bridges the gap. It is the models, the tools they drive, and the orchestration that decides what runs when, holds the results, feeds them back, and keeps the whole process inside scope and safety limits. FireCompass runs multiple large language models, has built its own small language models, and drives them through its own orchestration engine and a network of specialized agents. In FireCompass testing, this harness produces about 3X to 10X the result of a plain frontier model used as a wrapper with tools connected. For reference, Mythos is about 2X its own previous model. On that basis, FireCompass already delivers Mythos-grade results for dynamic application security testing.

The remainder of this paper sets out the four problems that make an LLM insufficient on its own, defines the harness engineering that solves each one, and describes how FireCompass has built it. It closes with an honest scope boundary: FireCompass does dynamic testing, not static analysis, and that distinction is deliberate.

2 The Gap, and the Discipline That Bridges It

An LLM is not enough for enterprise pentesting because the model is only the reasoning part of the job. Everything that makes a test real, taking action, holding state, proving findings, staying safe, lives outside the model. The discipline of building that surrounding system is harness engineering, and it is the most underestimated part of any autonomous pen testing platform.

A harness has three moving parts. First, the models: not one, but several, each chosen for the task it handles best. Second, the tools those models drive, from crawlers and request engines to browser automation and credential handlers. Third, the orchestration that sequences the tools, holds their output, feeds it back into the reasoning loop, and enforces the boundaries the model itself has no concept of. How good a result you get depends far more on how well these three are engineered together than on which model sits at the center.

This is why the industry's focus on model leaderboards misses the point, and why a more capable LLM does not close the gap. A better model raises the ceiling on reasoning. It does nothing for execution, state, validation, or safety, because those are not model properties. They are harness properties. The model is necessary but not sufficient. The harness is what makes the intelligence usable.

2.1 The engine and the vehicle

The clearest way to hold this is the relationship between an engine and a vehicle. An engine generates power, but it has no wheels, no steering, and no brakes, so on its own it goes nowhere. A frontier model is the engine. It reasons, plans, and generates, and a powerful one is genuinely impressive. The harness is the vehicle built

around it: the drivetrain that turns reasoning into motion, the steering that keeps it on course, and the brakes that stop it before it does harm. A vehicle without an engine is inert. An engine without a vehicle is uncontrollable. FireCompass builds the vehicle, and treats the engine as a component it can swap as the market improves.

3 Four Reasons a Model Is Not Enough

Frontier models bring real capability to offensive security. They recognize vulnerability patterns, reason about exploitation approaches, decompose objectives into testable steps, parse application context, generate proof-of-concept code, and synthesize findings into readable reports. These are the engine's horsepower, and they are the reason AI-driven pen testing is possible at all.

But four problems stand between that capability and an enterprise-grade result. None is solvable by a smarter model, because none is a reasoning problem. Each is an engineering problem the harness has to close, and together they are why an LLM on its own cannot deliver an enterprise pentest.

3.1 Execution: intelligence that cannot act

Models generate text. They cannot execute code, open network connections, send HTTP requests, or interact with file systems. When a model produces an exploit payload, it has produced a string, not an exploit. Delivering that payload to a live target, observing the response, and determining whether exploitation actually succeeded requires a runtime the model has no connection to. And that runtime needs real-time awareness the model lacks: a model cannot see that an endpoint that returned 200 now returns 403, that a firewall blocked a payload mid-chain, or that a session token expired. Multi-step chains need continuous feedback between action and system response, wired directly into the decision loop.

3.2 State: coherence across long chains

Enterprise attack chains run 30 to 100 or more discrete steps: discovery, enumeration, credential extraction, authentication, privilege escalation, lateral movement, data access. Holding coherent state across a chain that long is not a context-window problem that a larger model solves. Within a conversation context, models hallucinate prior steps, forget discovered credentials, and contradict earlier decisions, and the longer the chain the worse it gets. Coherence requires a structured state machine that records what was tried, what succeeded, what was blocked, and which credentials are held, independent of the model's conversation history.

3.3 Validation: hypotheses are not vulnerabilities

Models are probabilistic. Asked to find vulnerabilities, they produce confident, syntactically correct findings that are sometimes simply wrong: vulnerabilities that do not exist, exploitation paths that are blocked, findings that cannot be reproduced. Without a validation layer that tests each finding against the live system, unvalidated model output and traditional DAST scanners alike land in the 50 to 70 percent false-positive range. In security, false positives are not merely inconvenient. They drive alert fatigue, consume remediation cycles chasing nothing, and create compliance exposure when reported findings cannot be independently verified. Every model output has to be treated as a hypothesis that is proven by execution before it is reported.

3.4 Safety and governance: bounded, auditable operation

A model given aggressive objectives will attempt aggressive techniques, because it optimizes for task completion, not infrastructure safety. Left unbounded, that means uncontrolled request rates that cause denial of service, write operations that corrupt data, and brute-force attempts that lock accounts. The model also has no concept of a test scope: given execution capability, it will follow attack paths into out-of-scope systems, production databases, and third-party services. And it produces outputs, not logs, while enterprise governance frameworks such as PCI DSS, SOC 2, ISO 27001, and DORA increasingly expect documented, auditable

evidence of testing, and DORA and PCI DSS in particular expect retained, structured records. Safety, scope, and auditability all have to be enforced at the execution layer, below the model, where they cannot be reasoned around.

THE PATTERN BEHIND ALL FOUR

None of these gaps closes with a better model. Execution, state, validation, and governance are not things a model does well or badly. They are things a model does not do at all. They are the harness's job, and the quality of the harness is what separates a demo from a system an enterprise can deploy.

Capability required for enterprise testing	Frontier model provides by default
Execute real exploits against live systems	No. Text generation only
Maintain coherent attack state across 50+ steps	No. Conversation context degrades
Validate that each finding is actually exploitable	No. Produces hypotheses, not evidence
Enforce test scope boundaries	No. Requires external enforcement
Prevent production system disruption	No. Requires platform-level safety controls
Generate timestamped, tamper-evident audit logs	No. Produces outputs, not logs
Produce consistent, reproducible results	No. Non-deterministic by nature

4 The FireCompass Harness

FireCompass has built the harness described in the previous section over several years, combining offensive security expertise with frontier model integration. This section describes how each part is engineered and how the parts interact.

4.1 Multiple models, plus purpose-built small models

No single frontier model is optimal for every task in a penetration test. Code generation benefits from strong coding models. Long multi-step reasoning benefits from extended-reasoning models. Rapid surface classification benefits from faster, lower-cost models. FireCompass runs multiple large language models and routes each task to the model that performs best for that task class, maintaining a model registry and evaluating performance continuously against internal security benchmarks. Because intelligence is separated from execution, a new frontier model can be integrated without changing a line of execution infrastructure.

Alongside those, FireCompass has built its own small language models for specific, high-volume tasks. This is partly an accuracy decision and largely an economic one. Driving every step through frontier-model tokens makes continuous testing prohibitively expensive. Purpose-built small models plus an owned orchestration engine bring the cost down far enough that always-on testing is viable, which is what makes a continuous cadence real rather than aspirational.

Task category	Model selection rationale
Initial vulnerability hypothesis generation	Best-performing security reasoning model per current benchmark, evaluated continuously across providers

Task category	Model selection rationale
Complex multi-step chain construction	Extended-reasoning model selected for chain coherence and long-horizon planning depth
Exploit code generation and payload crafting	Top-ranked coding model with security-domain prompting from the orchestration layer
Application context analysis (JS, API docs)	General model with large context window for document understanding
High-volume classification and triage	FireCompass small language models, tuned for cost and speed at scale

4.2 Specialized agents, orchestrated

Execution runs on a distributed network of specialized agents. Each agent owns one capability domain and operates as an independent unit with its own scope validation, rate limiting, and audit logging. Agents communicate through a shared state store rather than directly, so the attack state stays coherent even when an individual agent fails or is terminated. The orchestration engine spawns, monitors, and terminates agents, enforces scope before any action is dispatched, controls request velocity globally, and routes agent output back into the state store for the intelligence layer to plan against.

Agent	Execution responsibility
Surface Mapping Agent	Crawls and enumerates the attack surface: subdomains, endpoints, APIs, and authentication surfaces. Discovers shadow assets outside the initial scope definition.
Reconnaissance Agent	External intelligence gathering: dark-web credential exposure, leaked-repository scanning, DNS enumeration, and technology fingerprinting.
Vulnerability Assessment Agent	Runs targeted tests against identified endpoints. Implements OWASP Top 10: 2025 for web app targets. Routes all findings to the validation pipeline before reporting.
Authentication & Credential Agent	Manages authenticated sessions: credential injection, session-token handling, OAuth navigation, and credential-reuse testing across environment boundaries.
Business Logic Agent	Tests workflow logic: multi-step process abuse, state manipulation, authorization-boundary violations, and transaction integrity, in authenticated context.
Chain & Lateral Movement Agent	Constructs and executes multi-stage chains from validated findings, attempting credential reuse and application pivoting.
Evidence Collection Agent	Attached to every confirmation. Captures request and response pairs, generates PoC code validated against the live target, and assembles chain-of-custody documentation.

4.3 The state machine

The state machine is the structural backbone of an engagement. It maintains a persistent graph of the target environment, its assets, relationships, authentication surfaces, discovered vulnerabilities, tested paths, and accumulated credentials, that survives any individual agent or model context reset. It gives the intelligence layer full current context at each planning step, drives the orchestration layer's prioritization toward unexplored high-value paths, and forms the authoritative source of record for the audit trail. Attack paths are modeled as a

directed graph aligned to OWASP Top 10 for web application targets and presented as a framework-aligned narrative in the final report.

4.4 The validation pipeline

Every candidate finding passes through validation before it is recorded as confirmed. The pipeline runs in four stages: the intelligence layer generates a structured hypothesis; the runtime delivers the proposed exploit against the live target under full scope and safety controls, capturing the exact request and response; the response is evaluated against the expected exploitation signature, with ambiguous results escalated for re-testing; and for confirmed findings, the evidence agent assembles the record, including reproduction steps and working PoC code validated against the live target.

Findings that fail the exploitability assessment are discarded, not reported as low-confidence noise. This is the mechanism behind the below-2-percent false-positive rate. It is engineering discipline in the validation layer, not model accuracy, and it is what separates FireCompass output from the 50 to 70 percent false-positive rates of unvalidated DAST scanning.

> **WHY VALIDATION DRIVES THE FALSE POSITIVE RATE**
>
> Unvalidated model output and traditional DAST scanners produce false-positive rates of 50 to 70 percent. FireCompass reports below 2 percent because every finding is proven against the live target before it appears. The difference is attributable to the harness, not to model quality.

4.5 The safety architecture

Safety is a mandatory enforcement layer between orchestration and the execution runtime. No agent action reaches the runtime without passing through safety validation. Safety is a gateway, not a policy, which is what makes the system safe by design rather than by convention.

Component	Mechanism	What it prevents
Scope Boundary Enforcement	Asset whitelist checked against every proposed action before dispatch	Out-of-scope testing, production database access, third-party exposure
Rate Limit Controller	Token-bucket algorithm per target host with a configurable request ceiling	Unintended denial of service, account lockout from brute force
Kill Switch	Immediate termination signal propagated to all active agents	Runaway tests, unexpected system impact, operator-initiated halt
Anomaly Detector	Monitors target response patterns; flags unexpected error rates or latency spikes	Early warning of unintended disruption during execution
Credential Scope Guard	Environment tags on all credentials; prevents cross-environment reuse	UAT credentials used against production, credential leakage across programs
Safe Payload Enforcement	Read and Create operations permitted within scope; Modify, Update, Delete blocked by default and never executed autonomously	Unintended data modification, record deletion, state corruption

4.6 Enterprise governance and deployment

The governance layer has no model dependency. It is pure engineering, and it is the reason enterprises can deploy FireCompass inside their existing frameworks. RBAC scopes permissions across user roles, asset groups, programs, and data classification, with multi-tenancy isolation so findings and credentials from one

program are never accessible from another. Every platform action is recorded in an append-only audit log with a cryptographic timestamp, supporting DORA, PCI DSS 4.0, SOC 2 Type II, and ISO 27001. Reporting produces two outputs from one data model: executive reports on risk posture and business impact, and engineering reports with endpoints, HTTP evidence, reproduction steps, PoC code, and remediation guidance.

FireCompass deploys as a managed SaaS for external assets and as a customer-deployed virtual appliance for internal assets, both sharing the same intelligence layer, agent network, and governance components. Continuous, here, means always-on surface discovery plus scheduled or on-demand testing campaigns, not thousands of agents running in parallel at all times. The small-model economics of Section 4.1 are what make that cadence affordable, with delta analysis prioritizing what changed between runs rather than re-testing every surface every time.

THE FORMULA

Engine (routed frontier models) + Vehicle (the FireCompass harness)
= validated, safe, auditable autonomous pen testing

5 Validation and Results

5.1 The harness multiplier

FireCompass tested this architecture directly. Take any frontier model and use it as a plain wrapper with tools connected. Call that result X. Across FireCompass testing, the full harness produces about 3X to 10X that result depending on task class. For external reference, Mythos is about 2X its own previous model. The gain comes from the harness, routed models, purpose-built small models, orchestration, and validated execution working together, rather than from any single model being smarter. This is the basis for the claim that FireCompass already delivers Mythos-grade results on dynamic application security testing.

5.2 Benchmark validation

FireCompass agents were evaluated against recognized web application security test environments in fully autonomous mode, without manual steering or predefined exploit playbooks. The objective was end-to-end operational validation: exploit execution, evidence capture, autonomous progression, and finding confirmation against live targets. The results should be read as platform-level execution outcomes, which is to say harness outcomes, not standalone model measurements.

Benchmark environment	Observed outcome	Validation notes
XBEN Validation Suite	Complete benchmark coverage across evaluated scenarios	Autonomous execution without manual intervention
Acuart / Vulnweb Testbed	All evaluated findings validated with exploit evidence	Request and response evidence captured
DVWA (Damn Vulnerable Web App)	Full vulnerability-class coverage across configured difficulty levels	Multi-step exploit validation confirmed

5.3 Fortune 500 production deployment

A Fortune 500 technology company previously running a large consulting firm's manual pen testing program moved to FireCompass. The engagement covered 2,000+ web applications, of which only 200 were being tested annually under the previous program.

Metric	Before and after
Cost per application test	~$5,000 per test (2 consultant-days) to **under $1,000 per test**
Application coverage	200 of 2,000+ applications annually to **full portfolio, continuously**
Lead time to test initiation	2+ weeks scheduling and setup to **on-demand, zero lead time**
False positive rate	~70% from supplementary DAST scanning to **below 2%**
Attack chain discovery	Isolated findings; chains scoped out of manual work to **chained paths discovered autonomously**
Previously untested assets	No visibility to **vulnerabilities found on assets never previously tested**

6 Scope: Dynamic, Not Static

An honest boundary matters to the people who buy this. FireCompass is not a general-purpose replacement for every kind of AI-assisted security analysis, and it does not claim to be. Mythos is stronger at static analysis and code analysis, and FireCompass does not compete there.

FireCompass does dynamic testing: it exercises a running application and its APIs, orchestrating the tools and agents required to discover, exploit, validate, and chain findings against live systems. That orchestration is precisely what a model will not do on its own. A model can reason about a running application, but it will not stand up the agents, drive the tools, hold the state, and enforce the safety controls that dynamic testing requires. The harness does that. This is the Mythos-grade claim stated precisely: FireCompass delivers Mythos-grade quality on dynamic testing, and the distinction is the point, because it is what a serious buyer needs to know before choosing a tool for a given job.

	Static analysis and code analysis	Dynamic application and API testing
A frontier model alone	Strong	Reasons, but will not orchestrate the tools to execute
FireCompass harness	Not the focus	Orchestrates tools and agents to execute end to end

6.1 Beyond the core: extended modules

The same harness, its intelligence layer, agent network, and governance, is the foundation for several extended capabilities: Attack Surface Management and CTEM for continuous external discovery; Continuous Automated Red Teaming for objective-based simulation of full kill chains; PTaaS for expert-in-the-loop delivery on complex engagements; and Infrastructure Penetration Testing, which extends the agent network to internal networks using the same validation pipeline and audit infrastructure as web application testing.

7 Conclusion

Frontier models are becoming more capable, more accessible, and increasingly commoditized. That alone will not solve autonomous penetration testing, because reasoning is not execution. A model may propose an attack

path or infer a vulnerability, but it cannot assure exploit validation, operational safety, state persistence, evidence integrity, or governed operation. Hypotheses are not vulnerabilities.

The defining work of autonomous security lives in the harness around the model: orchestration, runtime control, validation pipelines, trust-boundary enforcement, auditability, and governed operation under adversarial conditions. This is the engineering problem the market is presently underestimating, and it is the reason FireCompass can deliver Mythos-grade quality on dynamic testing without a Mythos-class model. The engine is becoming commodity. The harness is the differentiator, and that transition is already underway.

8 About FireCompass

FireCompass is an Agentic AI platform for autonomous penetration testing and red teaming across Web, API and infrastructure. It discovers shadow assets and web applications, safely validates what is exploitable, and connects findings into multi-stage attack paths with near-zero false positives. Unlike traditional scanners, it discovers credential reuse, business-logic flaws, privilege escalation, and app-to-app or app-to-network lateral movement. It can operate autonomously or with expert-in-the-loop validation. FireCompass has 30+ analyst recognitions across Gartner, Forrester, IDC, and is trusted by Fortune 500 enterprises.

ANALYST RECOGNITION

Gartner · 30+ Reports, Hype Cycle 4 cycles running | Forrester · Notable Vendor

GigaOm · Radar Leader (2023) | IDC · Innovators

Evaluate FireCompass in your environment. United States · Singapore · Malaysia · Switzerland · Japan · Philippines · firecompass.com