Synack

TECHNICAL BRIEF

Continuing the Build vs Buy Conversation

So You Need an Agent Harness...

A GUIDE TO BUILDING AN ORCHESTRATION LAYER THAT HOLDS UP IN PRODUCTION

The model is only as good as the harness around it.

With the abundance of powerful, readily available AI models, it's no surprise security teams are starting to build their own AI pentesters to scale internal red teams.

It's relatively easy to spin up an AI agent. Point a capable model at a target, ask it to look for problems, and it will produce something that looks like a finding. However, getting that result to hold up in a real-world environment is the real challenge.

Choosing the model will get you part-way there. The bigger task is building the harness around your agents. This is the orchestration layer that decides which agent does what, in what order, and how results are checked.

Based on our own experience building Sara, the Synack Autonomous Red Agent, we've compiled this guide about what it takes to develop a successful orchestration layer. This guide outlines our top considerations for architecture, operations, and maintenance if you're planning an internal build.

THE BASICS

What Is an Orchestration Layer?

An orchestration layer, sometimes called an agent harness, is the software system that sits around a raw language model and turns it into something capable of conducting a pentest, rather than just answering a prompt. This is essential, as a frontier model on its own can reason well but struggles with long-horizon planning. The harness is what breaks a pentest into a structured pipeline that goes from reconnaissance to exploitation to validating results.

It also governs how swarms of agents divide labor and stay coordinated, deciding when to use identical worker agents for parallel coverage versus purpose-built specialist agents for specific tasks. In short, the model provides raw intelligence, but the orchestration layer is what supplies the methodology and discipline that makes that intelligence reliable enough to find and validate real vulnerabilities in production environments.

The model provides the intelligence. The harness provides the discipline that makes the agent reliable.

From Generic 'Find Vulnerabilities' to Specialized Attack Paths

A model asked to "find vulnerabilities" with no further structure has to improvise its own plan, its own priorities, and its own stopping point. That typically results in inefficient runs where the model gets way off track with outputs that are questionable. In essence, it becomes a sloppy Proof of Concept (POC) at best. On the other hand, with a well-built harness, your POC has the structure up front, so the model makes in-scope decisions within a tuned process.

With an orchestration layer wrapped around a model, you can deliver refined prompts that are specialized for a specific task, as opposed to generic instructions. The harness can break down a job into discrete steps instead of one open-ended ask. It also directs tasks to purpose-built sub-agents that each own a narrow piece of the problem so it isn't just one agent doing everything at once.

That's also why swapping in a newer or more capable model doesn't guarantee it's fit for purpose or ready for enterprise application. A stronger model dropped into a weak harness still inherits the harness' blind spots, while a well-designed harness can get strong, consistent results even from an existing stable model.

DESIGN PRINCIPLES

Top 3 Requirements for a Reliable Harness

ARCHITECTURE

01 Build a multi-stage pipeline, not a single call

A pentesting workflow should never be a single ask to a single model. It should move through distinct stages: reconnaissance to map the target, broad discovery to surface anything plausible, filtering to keep only what has real evidence, exploitation to confirm impact, chaining to find compounded severity, and independent validation before anything reaches a report.

02 Separate the judge from the worker

A model that grades its own output tends to convince itself it succeeded rather than catching its own errors. Any triage or quality check should run in a separate context that is adversarial to the findings and actively tries to disprove them, rather than trusting the same model that produced a result to also validate it.

03 Avoid a single top-level model as planner or judge

Putting one large model in charge of judging whether other agents are making progress causes it to hyperfixate on whatever most recently sounded promising and lose the bigger picture. Instead, build a distributed planning structure that assigns oversight in smaller, constrained pieces, so progress gets judged locally at each stage rather than by one model trying to hold the entire run in view.

OPERATIONS

01 Design for real scale

A working system should expect to run hundreds of agents per job, not a handful, with the count scaling automatically based on the size of the target. Even a single stage, like initial discovery, can require dozens of sub-agents running in parallel to get adequate coverage.

02 Mix agent types deliberately

Identical worker agents are useful for parallel coverage and speed across a broad attack surface. Purpose-built specialist agents are needed for tasks that require distinct expertise, such as reverse-engineering an API and writing a custom client against it, so a mature system uses both rather than relying on one agent type for everything.

03 Tightly constrain what each agent is allowed to decide

Giving an agent too many options or too much responsibility at once degrades its performance. The system should favor an assembly-line structure with branching, where each agent chooses among a small, relevant set of options for that specific target, rather than facing the full decision space at once.

MAINTENANCE

01 Don’t rely on off-the-shelf multi-agent frameworks

General-purpose orchestration frameworks tend to hold up in short, few-step benchmarks but break down once tasks require many sequential steps toward a real goal. In practice, that shows up as agents losing planning coherence over long-running tasks, duplicating work instead of coordinating hand-offs cleanly, and continuing to chase leads they should have abandoned, so teams should expect to need a custom-built planning and orchestration layer rather than adopting one off the shelf.

02 Plan for prompt-model coupling as ongoing maintenance

Hand-tuned prompts are effectively fit to the specific model they were built against. Model upgrades, including forced ones when vendors retire older versions, can silently regress performance, so teams should treat re-tuning as a continuous process rather than a one-time setup cost.

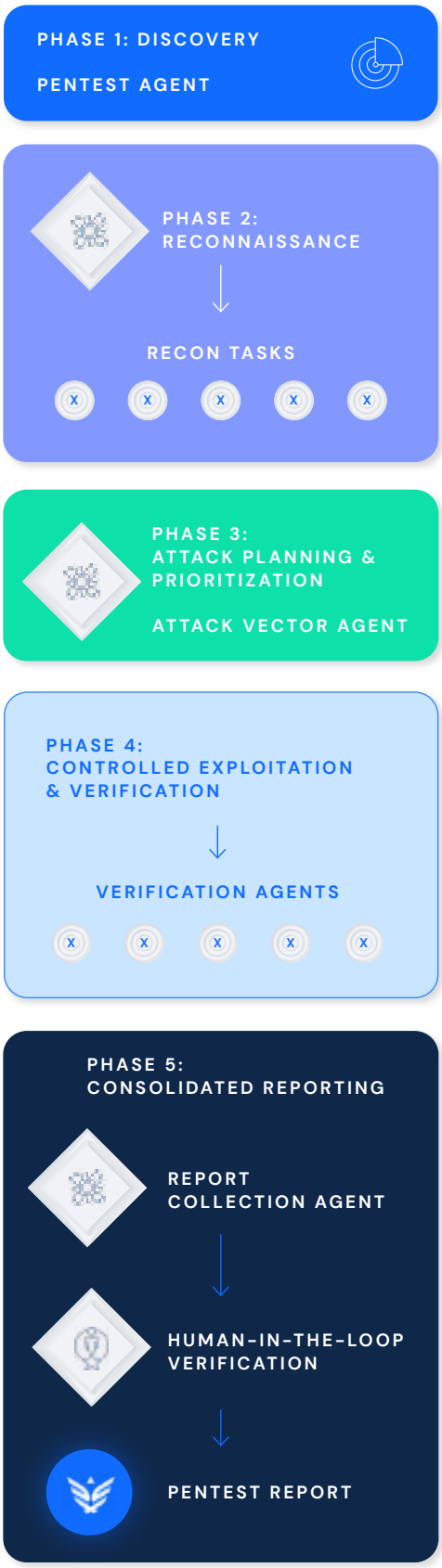
How Synack Built the Agent Harness for Sara

THE BUILD BEHIND SARA AI PENTEST AND HOW WE BROKE OUT OF THE LAB BENCHMARKS.

Admittedly, Sara's orchestration layer didn't come together on the first attempt. Initially, we tried several existing multi-agent frameworks. Each one looked promising in testing, then lost focus once a task stretched past a few steps.

That led us to build the planning and coordination layer ourselves on top of the agents. The key was sitting down with our top performing researchers to understand how they approach a target. Then we translated their thought process into a defined structure that's used to constrain the responsibilities of each agent in the system.

In practice, that structure runs through distinct stages. An early step does the groundwork: crawling the target, probing for live services, fuzzing endpoints, and understanding how user input actually behaves before anything gets flagged. A discovery step casts a wide net, surfacing anything remotely plausible. A separate, independent step then narrows that list keeping only what has real supporting evidence. From there, specialist agents take the confirmed leads and push to see how far the impact goes. The structure comes to a close with an independent validation pass before anything reaches a report.



Synack

DECISION TIME

Will You Build or Buy?

Getting the orchestration layer right takes real trial and error. It also takes ongoing investment to keep pace with new models, and constant validation to confirm performance holds up outside a lab setting. If you're in the middle of evaluating build vs buy, we'd love to compare notes.

START YOUR FREE TRIAL OF SARA AI PENTESTING

COMPARE NOTES WITH OUR TEAM

About Synack

Synack is the leader in human-led and AI-powered Penetration Testing as a Service (PTaaS), transforming offensive security to help organizations proactively reduce risk, stay compliant, and defend against evolving cyber threats. We are committed to making the world more secure by harnessing agentic AI innovations and a talented, vetted community of security researchers to deliver continuous penetration testing and autonomous vulnerability management. Founded by former NSA operatives, Synack has enabled nearly 10 million hours of expert testing to protect critical assets, from global financial systems to U.S. Defense Department networks.

To learn more about Synack, visit www.synack.com.