

OPEN ACCESS

Volume: 5

Issue: 4

Month: October

Year: 2026

ISSN: 2583-7117

Published: 28.10.2026

Citation:

Vidhaan Appaji, Lokesh Upputri, Senthil Arasu T, Daksha S "Small Models, Precisely Tuned: A Two-Tier Small-Language-Model Architecture for Autonomous Penetration Testing Soft-Prompt-Tuned Strategic Reasoning Paired with a LoRA-Fine-Tuned Tool-Calling Executor, Evaluated Against VAmPI and OWASP Juice Shop" International Journal of Innovations In Science Engineering And Management, vol.5, no.4, 2026, pp. 229-243.

DOI:

10.69968/ijsem.2026v5i4229-243



This work is licensed under a Creative Commons Attribution-Share Alike 4.0 International License

Small Models, Precisely Tuned: A Two-Tier Small-Language-Model Architecture for Autonomous Penetration Testing Soft-Prompt-Tuned Strategic Reasoning Paired with a LoRA-Fine-Tuned Tool-Calling Executor, Evaluated Against VAmPI and OWASP Juice Shop

Vidhaan Appaji¹, Lokesh Upputri¹, Senthil Arasu T¹, Daksha S¹

¹Master of Science in Applied Artificial Intelligence, Shiley-Marcos School of Engineering, University of San Diego

Abstract

Large language models increasingly serve as reasoning engines inside automated penetration-testing agents, but two problems recur across the recent literature: general-purpose models used as end-to-end planners struggle to sustain coherent multi-step attack reasoning, and giving that reasoning role to a single large model is expensive per call. This project addresses both by splitting the pipeline into two small, purpose-trained language models instead of one large general-purpose one. A Strategist Qwen3-8B adapted with soft-prompt tuning on 2,311 hypothesis-generation examples spanning 26 real vulnerability categories mined from disclosed HackerOne reports - proposes which vulnerability to test next and why. An Executor Qwen3-1.7B-Base adapted with LoRA - turns that hypothesis into a concrete, correctly formatted HTTP tool call. Both models were evaluated in isolation and inside a closed, autonomous loop against two live, intentionally vulnerable targets: OWASP Juice Shop and a self-hosted VAmPI instance. The headline result is a controlled ablation: holding the Strategist and the attack loop fixed and swapping only the Executor's weights, LoRA fine-tuning takes the autonomous loop from 0 of 9 target vulnerability categories confirmed to 7 of 9. Combining that loop with a separate three-role harness run resolves 8 of VAmPI's 9 applicable categories overall. Across all 38 applicable tests pooled from both live targets, 22% produced a confirmed, evidence-backed finding, while the soft-prompt Strategist matched or beat a fully-prompted master-prompt baseline on the same base model on 11 of 18 comparable vulnerability patterns while using 59–85% fewer input tokens per call. The project is equally direct about where the approach struggles: against a frontier model with no scaffolding the Strategist scores 2.53/5 versus 4.94/5, and multi-step reasoning chains degrade to a mean score of 2.0/5 by the fourth or fifth step. The results support a narrow but load-bearing claim: fine-tuning the execution layer of a small-model agentic pipeline is, in this setting, a larger and cheaper lever on end-to-end task success than fine-tuning - or scaling to - a larger reasoning model.

Keywords: Agentic Penetration Testing, Large Language Model Fine-Tuning, LoRA-Based Vulnerability Exploitation, Autonomous Security Testing, Multi-Agent AI Architecture.

INTRODUCTION

Automated penetration testing - using software rather than a human analyst to discover exploitable weaknesses in an application - has historically split between two kinds of tools. Narrow, deterministic scanners such as Burp Suite, OWASP ZAP, and known-CVE matchers automate a single phase, such as request fuzzing, reliably and cheaply, but cannot generate or revise a hypothesis about an application's behavior. Large language models, by contrast, can reason in natural language about what an unfamiliar endpoint might be doing and propose a next test, but asking one general-purpose model to both plan and execute an engagement has been shown to produce context-management failures and inconsistent tool-calling as the engagement grows longer (Section 3).

This project targets end users who need a self-hostable, low-cost alternative to a frontier-model-driven pentesting agent - security teams evaluating internal or client applications who cannot or do not want to send attack-surface details to a third-party API, and who need a tool that can be re-run per engagement without per-call API budget concerns. Concretely,

the project builds and evaluates a two-tier pipeline: a Strategist that proposes which vulnerability to test next, and an Executor that turns that hypothesis into a real HTTP request against a live target - OWASP Juice Shop and a self-hosted VAmPI instance, two intentionally vulnerable applications purpose-built for exactly this kind of testing (Section 3). The training data for the Strategist is drawn from real, disclosed vulnerability reports on the HackerOne bug-bounty platform (Section 2), so that the model's learned reasoning is grounded in genuine, previously confirmed vulnerability patterns rather than synthetic or templated examples.

The project's central, testable hypothesis is narrow: within a small-language-model pipeline where planning and execution are split into two separate roles, fine-tuning the execution role changes what the pipeline can actually accomplish - independent of how good the planning role is. Each role is fine-tuned with the parameter-efficient technique suited to what it needs to learn: soft-prompt tuning for the Strategist, which needs to be steered toward a specific reasoning style without disturbing its broad pretrained knowledge of vulnerability classes, and low-rank adaptation (LoRA) for the Executor, which needs to learn a genuinely new, narrow skill (correctly formatted tool calls) that a small base model does not reliably exhibit out of the box. The final product, if the hypothesis holds, is not a finished commercial tool but a validated architectural pattern: evidence for whether a small, cheaply fine-tuned pipeline can do real, verifiable vulnerability-finding work without a frontier-model call anywhere in its live decision loop.

DATA SUMMARY

Two distinct datasets were built for this project, one per model role, because the Strategist and the Executor need different kinds of training signal. The Strategist's dataset supplies open-ended reasoning examples; the Executor's supplies narrow, mechanical tool-call examples.

Strategist Dataset

The Strategist's training set was built from real, publicly disclosed vulnerability reports rather than hand-written or purely synthetic examples. Starting from 10,876 disclosed HackerOne reports matching 26 target vulnerability categories, the team sampled 2,311 examples, each grounded directly in a real disclosed report rather than generated from a template, and validated every reasoning-chain step for structural correctness. Each record's key variables are a situation description (the target application type, the vulnerability category under test, and steps already attempted), a proposed next step in natural language, a resulting confidence estimate, and a continue/stop decision.

A held-out stratified split by vulnerability pattern produced 1,949 training and 362 evaluation examples across all 26 categories.

Automated data-quality checks, re-run on every dataset revision, surface the issues present in this data and how they were handled. Completeness was clean (0 missing required fields across 9,244 checked values), so no imputation was needed. Class balance across the 26 categories was good but not perfect (1.20:1 between the largest and smallest category), which was accepted rather than corrected, since a synthetic rebalancing step would have reintroduced exactly the templated, non-authentic quality the real-report sourcing was meant to avoid. The more significant issue was duplication: 112 exact duplicate (situation, completion) pairs were identified out of 2,311. Rather than silently dropping or silently keeping them, they were retained but flagged in the dataset metadata, so that any downstream evaluation could account for them; the stratified train/eval split (Section 4) was constructed to avoid placing duplicate pairs across the split boundary. Provenance was verified programmatically rather than assumed: 100% of the 2,311 examples trace to a real, disclosed HackerOne report with a working source link, and 71.4% of situations (1,651 of 2,311) are textually unique, which was used as a proxy for how much genuine input diversity the model would see beyond the categorical label. In a fully productionized pipeline, this same source-verification step would run continuously against HackerOne's public disclosure feed to keep the training distribution current as new vulnerability classes and disclosure patterns emerge.

Executor Dataset

The Executor's dataset is smaller and structurally different from the Strategist's, because the Executor needs a narrow, mechanical skill - correct HTTP method and path selection - rather than open-ended reasoning. It combines 393 base examples: 16 mined from real disclosed reports re-shaped against VAmPI's actual endpoint surface, 9 VAmPI-pattern-specific augmentations, and 368 general synthetic examples across 13 endpoint shapes, each wrapped in realistic, messy Strategist-style prose rather than a clean template, so the Executor learns to parse a hypothesis the way the Strategist actually writes one rather than an idealized instruction. Each record's key variables are the hypothesis text (category, endpoint, and the Strategist's reasoning), and the ground-truth tool call (HTTP method, path, and parameters or body). This base set was expanded through error-correction augmentation- pairing a deliberately wrong tool call with its correction - into 445 formatted training records (228 direct positive completions and 217 error-correction pairs). A stratified split across 14 categories produced 374 training and 71 evaluation records. The relationship between the

Executor's input (hypothesis text) and its target output (tool call) is direct and largely deterministic given the category and endpoint, which is exactly why LoRA - which updates

real model weights rather than only steering existing behavior - was chosen over soft-prompt tuning for this role (Section 4).

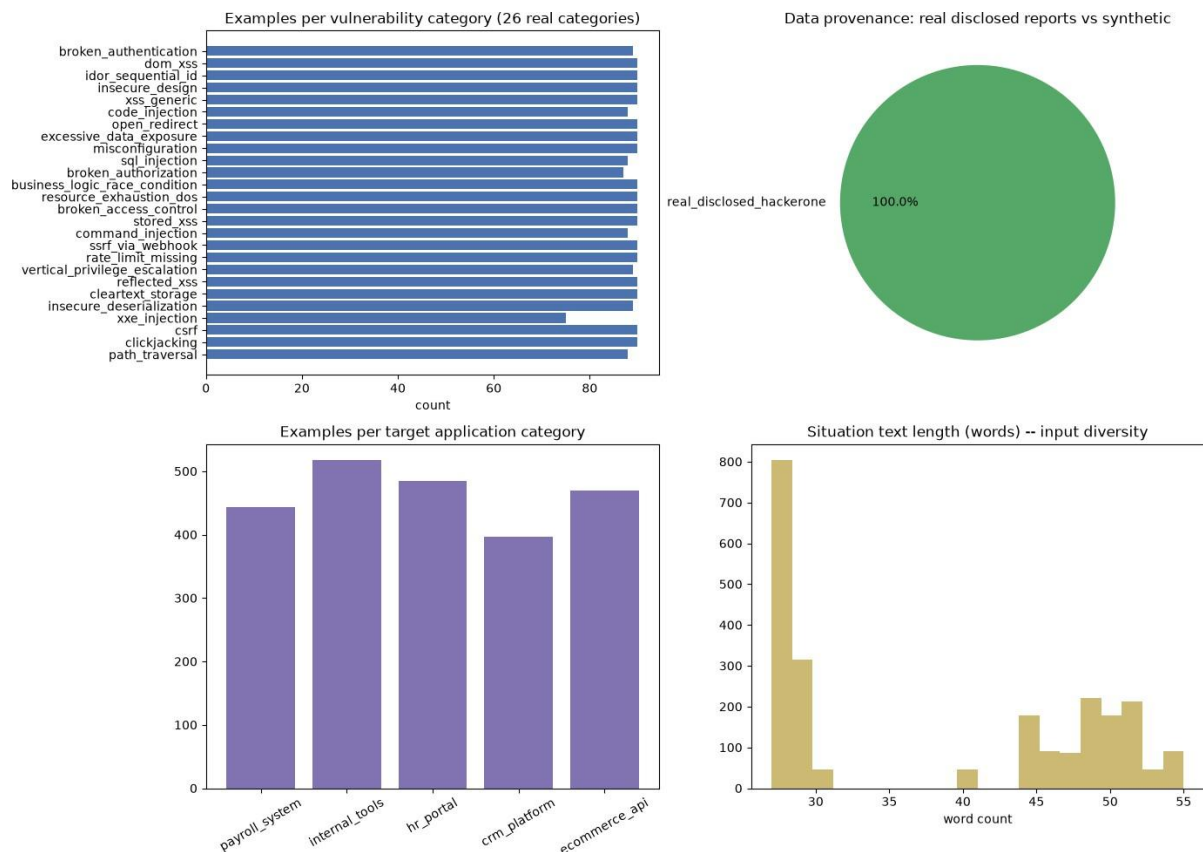


Figure 1. Strategist training-set composition: category balance across 26 real vulnerability classes (top left), source provenance (top right), distribution across five target application archetypes (bottom left), and situation-text length as a proxy for input diversity (bottom right).

LITERATURE REVIEW

Automated Penetration Testing Systems

Deng et al. (2024) introduced PentestGPT, one of the first frameworks to organize an LLM-driven penetration test around an explicit task tree rather than unstructured chat, and found that this structuring improved task completion on real-world CTF benchmarks compared with prompting a base LLM directly. A subsequent revision, PentestGPT v2, extended the task tree into an evidence-guided attack tree search combined with a Task Difficulty Assessment module, in which the same LLM performing the engagement estimates its own confidence and difficulty at each decision point to guide which branch of the attack tree to pursue next (PentestGPT v2 authors, 2026).

Shen et al. (2025) proposed PentestAgent, a multi-agent architecture that decomposes a penetration test into dedicated reconnaissance, planning, and execution agents coordinated through a shared retrieval-augmented memory, evaluated against 67 vulnerable targets spanning 32 CWE categories; separating these roles reduced the context-management failures observed in single-agent designs. Andreucci and

Castiglione (2026) introduced a related multi-agent framework, Automation-Exploit, that adds a digital-twin validation stage before executing exploit attempts against a live target, specifically to mitigate the risk of destabilizing production systems during automated testing.

A separate line of work has questioned whether specialized scaffolding of this kind is even necessary. Checkmate (2025) benchmarked a general-purpose coding agent with no pentest-specific scaffolding against PentestGPT and PentestAgent and found that it outperformed both specialized systems on the same benchmark, motivating that paper's own Planner-Executor-Perceptor architecture, in which a classical, non-learned symbolic planner governs the overall attack plan while LLM agents are restricted to execution and observation-interpretation roles. This result - a deterministic planner outperforming LLM self-assessment-driven planning - directly motivates the open question this project addresses: whether the planning role can instead be filled by a small, learned model. Commercially, static and dynamic application security testing tools have long automated narrow phases of a penetration test, such as request fuzzing or known-CVE

matching, but do not perform the open-ended hypothesis generation that the LLM-based systems above target.

Parameter-efficient Adaptation of Language Models

Lester, Al-Rfou, and Constant (2021) introduced prompt tuning, which freezes all of a pretrained language model's original parameters and learns only a small number of continuous soft-prompt embeddings; at sufficient model scale this can match full fine-tuning performance while updating several orders of magnitude fewer parameters, and it is substantially more robust to distribution shift between training and inference, since the pretrained model's general-purpose knowledge is never overwritten. Hu et al. (2021) introduced Low-Rank Adaptation (LoRA), which instead injects small trainable low-rank matrices into a model's existing weight layers, allowing the update to represent genuinely new task-specific behavior; the original paper reports comparable or better performance than full fine-tuning while training under 0.1% of total parameters. Xu et al. (2023) characterize the practical distinction directly: prompt-tuning-style methods are best suited to steering a capable base model's existing behavior toward a new format or style, while LoRA-style methods are better suited to teaching a model behavior it does not already reliably exhibit - the direct design rationale behind this project's choice of soft-prompt tuning for the Strategist and LoRA for the Executor (Section 4). Taori et al. (2023) demonstrated a widely cited application of this distinction with Alpaca, in which a larger, capable instruction-following model was used to generate a synthetic instruction-tuning dataset that was then used to fine-tune a much smaller open model, establishing the general pattern of using a capable model to generate training signal for a cheaper deployed model.

Benchmarks and Evaluation Environments

Several benchmark suites evaluate LLM-driven offensive security agents under controlled conditions. Zhu et al. (2024) introduced NYU CTF Bench, 200 real Capture-the-Flag challenges drawn from CSAW competitions between 2017 and 2023, reporting that even capable frontier models solved only a small fraction of the included challenges. Yang et al. (2023) introduced InterCode-CTF, adapting 100 PicoCTF challenges into an interactive, tool-using evaluation environment. Zhang et al. (2024) introduced Cybench, 40 professional-level CTF challenges decomposed into graded subtasks to allow partial-credit evaluation of agent progress. For vulnerabilities specific to modern web APIs rather than classic CTF-style challenges, purpose-built vulnerable applications - OWASP's crAPI and Juice Shop projects and the independently maintained VAmPI project - each intentionally implement a documented, known set of vulnerabilities, so that an agent's findings against them can

be verified against ground truth. This project uses two of these targets directly: Juice Shop as a live, publicly hosted target and VAmPI as a self-hosted target with source-verified category applicability (Section 4). Separately, the reddelexc hackerone-reports project (2026) aggregates and de-duplicates publicly disclosed vulnerability reports from HackerOne, the source used for this project's Strategist training set (Section 2).

Multi-agent and Planner/executor Architectures

Beyond the pentest-specific systems above, the broader agentic-LLM literature has separately converged on splitting a single model's reasoning into distinct roles to reduce the context-management and long-horizon coherence failures observed in single-agent designs. Yao et al. (2023) introduced ReAct, interleaving explicit reasoning traces with tool-use actions within a single agent loop, reporting fewer hallucinated actions compared with reasoning-only or acting-only baselines on multi-step tasks. Moonshot AI's (2026) technical documentation for the Kimi K2 model family reports that a dedicated post-training pipeline combining supervised fine-tuning, large-scale agentic trajectory synthesis, and reinforcement learning was necessary to maintain stable behavior across sequences of 200–300 sequential tool calls, directly motivating this project's separation of planning and execution into two distinct model instances rather than one model performing both roles under a single, longer context.

METHODOLOGY

The pipeline separates two roles that prior single-model pentest agents conflate. The Strategist observes the current engagement state - target category, endpoint, steps already tried, current confidence - and proposes a next concrete test and a rationale. The Executor receives that hypothesis and converts it into exactly one structured tool call: an HTTP GET or POST against a specific path with specific parameters or a JSON body. A lightweight Judge role, a frontier model called through an API and used only for offline evaluation, never inside the live decision loop, scores whether the Strategist's hypothesis was technically sound and whether the Executor's tool call actually operationalized it. This separation lets every failure in the pipeline be attributed to a specific role rather than to “the model” as an undifferentiated whole, which underlies both the ablation in Section 5 and the attribution analysis throughout.

Model Architecture and Training Procedure

Both adapters are parameter-efficient, but for different reasons, following the distinction established by Xu et al. (2023): the Strategist is steered toward a specific reasoning style without disturbing Qwen3-8B's broad pretrained

knowledge, using soft-prompt tuning (PromptTuningConfig, 30 virtual tokens); the Executor learns a genuinely new, narrow behavior that the base 1.7B model does not reliably exhibit zero-shot, using LoRA ($r=16$, $\alpha=32$, dropout=0.05, applied across 7 attention and MLP projection matrices). The Strategist trains 122,880 parameters, 0.0015% of its 8B base; the Executor trains 17,432,576 parameters, 1.03% of its 1.7B base. Strategist training ran on a single NVIDIA L4 (24GB) for 2.99 epochs (730 steps); Executor training ran on two Tesla T4 GPUs (15GB each) on Kaggle for 2.98–3.0 epochs (140–141 steps). Training loss fell from 3.947 to 0.340 for the Strategist and from 2.555 to 0.062 for the Executor.

The two loss curves diverge in an instructive way: the Executor's LoRA loss falls much further over far fewer steps than the Strategist's soft-prompt loss, despite the Strategist

training on a larger, harder, multi-category dataset. This is consistent with LoRA updating real attention and MLP weights, which can fit a narrow, mechanical target such as a fixed JSON tool-call shape almost exactly, while the capped 30-virtual-token capacity of soft-prompt tuning deliberately trades a lower achievable loss for lower overfitting risk on a dataset that requires open-ended reasoning rather than a fixed output shape. Neither run configured a best-checkpoint selection metric on held-out loss, so the final, highest-step checkpoint, also the one with the lowest observed training loss in both cases, was used in every downstream evaluation. On the held-out Executor evaluation set, the fine-tuned model achieves 100% HTTP-method match, 100% path match, and 64.3% fully exact match (method, path, and parameters all correct); the gap between path/method accuracy and full exact-match reflects legitimate parameter-value variation rather than formatting errors.

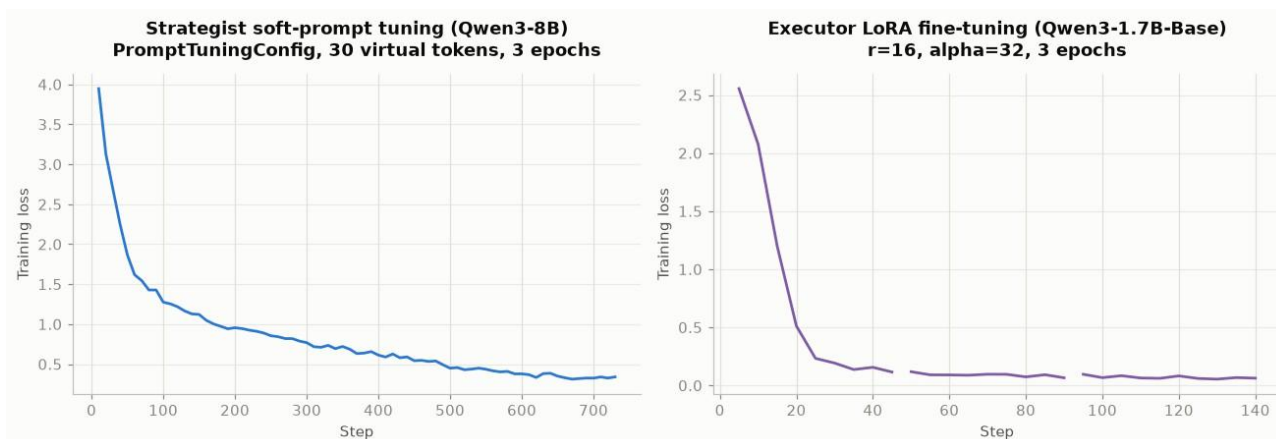


Figure 2. Real per-step training loss, read directly from each run's saved checkpoints. Left: Strategist soft-prompt tuning. Right: Executor LoRA fine-tuning.

Evaluation Harness Design

Three progressively more autonomous evaluation setups were used, each testing a different claim. The three-role harness runs, for each applicable category, a Strategist-Minimal call (soft-prompt adapter on, minimal input), a Strategist-MasterPrompt call (adapter off, full written instructions, same base model), and a separate Executor call that converts the minimal hypothesis into a real HTTP tool call fired at the live target, with a Gemini judge scoring both hypothesis quality and tool-call correspondence; this was run once against OWASP Juice Shop and once against a self-hosted VAmPI instance. The rogue-agent loop is autonomous and iterative: the Strategist formulates its own hypothesis each turn, choosing from categories still open and real known endpoints, and the Executor self-corrects up to two additional tries on failure; the loop stops when all 9 target categories are confirmed, 5 consecutive turns produce no new finding, or 25 iterations elapse. This is the project's central controlled ablation, run three times against VAmPI

varying only which model plays the Executor role. Finally, three judge-scored controlled comparisons isolate specific questions: a single-turn comparison against Gemini with no scaffolding for either side (47 situations), a comparison of the same base model instructed via a full master prompt versus the tuned soft-prompt prefix (at a 15-item pilot scale and a 362-item scale across all 26 categories), and a five-step chained-reasoning comparison judged holistically for coherence.

Every category tested against a live target was pre-classified as applicable or not-applicable ahead of time, by reading the target's actual source code for VAmPI or by direct black-box confirmation for Juice Shop, rather than being silently skipped. A status-code heuristic flags a candidate result automatically, but every flagged result was subsequently read against its actual tool call and response body and reclassified into one of four evidence-backed buckets - confirmed, plausible, false-positive artifact, or inconclusive - rather than reported at face value; this reclassification is what Section 5

reports throughout, not the raw heuristic. All offensive testing was conducted exclusively against applications explicitly designed for security-testing practice, with no engagement against production systems or third-party infrastructure outside the team's authorization, and no exploit code or trained model artifact from this project is packaged as a ready-to-run offensive tool.

RESULTS

Results below are ordered by finding strength, strongest positive evidence first and the toughest, most negative comparisons last, both across the section and within each table. Every case observed across every evaluation run is reported somewhere in this section or in Appendices A–D, not a curated subset.

Headline Result: Fine-tuning the Executor Alone Takes the Autonomous Loop From 0/9 to 7/9

The project's central, cleanest result is a controlled ablation. The same Strategist, the same VAmPI target, and the same rogue-agent loop logic were held fixed across two runs; the only variable that changed was whether the Executor carried its LoRA adapter. This is the project's headline, cleanest evidence for the small-model thesis: no architectural change, no larger model, and no extra prompting - only 393 targeted training examples and 17.4M LoRA parameters - separate total failure from majority success.

Before fine-tuning, the untuned Executor confirmed zero of nine target categories and stalled after only 5 iterations. The root cause was not a Strategist reasoning failure - it correctly and repeatedly identified `/users/v1/login` as the right endpoint for broken authentication - but a narrow, mechanical Executor error: every attempt sent an HTTP GET to an endpoint that only accepts POST, so every real response was a 404. After LoRA fine-tuning, with every other component unchanged, the same loop confirmed 7 of 9 categories and ran the full 16 iterations; the fine-tuned Executor's HTTP method and path match rates are both 100% (Section 4.1). An even earlier iteration of the pipeline, reusing the 8B Strategist itself with its adapter disabled as a stand-in Executor, also reaches 7 of 9, showing a larger, un-tuned model can substitute for fine-tuning a smaller one, but at far higher inference cost per call. The full 16-iteration log is reported in Appendix A.

The two categories the fine-tuned Executor still could not confirm, `sql_injection` and `cleartext_storage`, fail for a different reason than the untuned run: the Executor consistently targets the correct endpoint and method, but its training data taught it to construct valid request bodies, not adversarial payloads, so the status-code heuristics for those two categories never have a signal to fire on - a dataset-composition gap, not a formatting failure (Section 6).

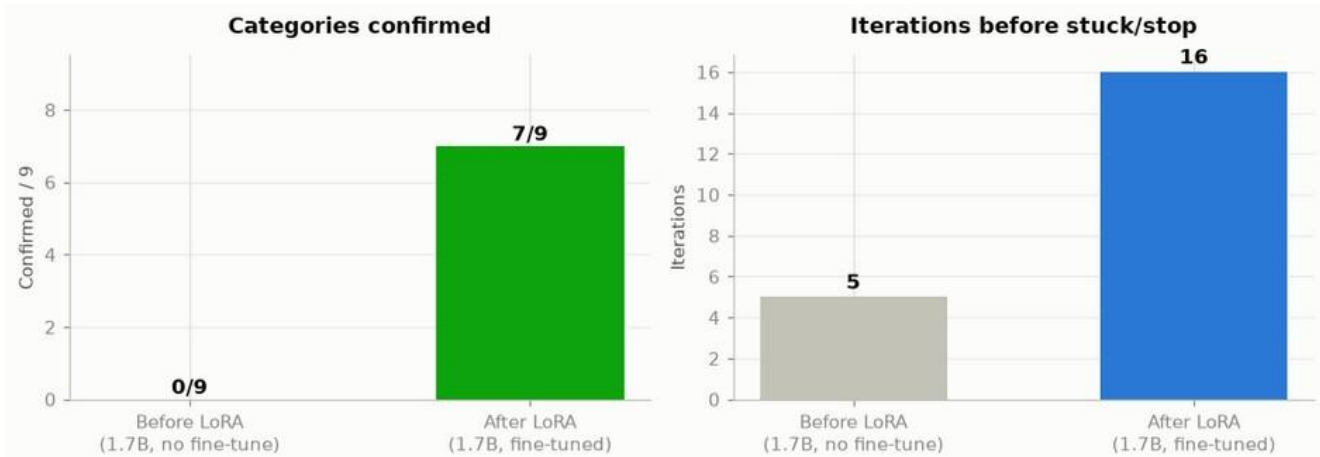


Figure 3. Left: vulnerability categories confirmed by the autonomous loop, before and after LoRA fine-tuning of the Executor alone. Right: iterations completed before the loop's stall condition was reached.

Combined Autonomous Coverage: 8 of 9 Vampi Categories Confirmed

Pooling the fine-tuned rogue-agent loop with the separate three-role harness run against the same VAmPI target resolves 8 of VAmPI's 9 applicable categories, with only `rate_limit_missing` left unresolved.

The loop performs strongly while it has real, un-tried options: across iterations 1–11, it correctly avoids every

already-attempted pair, lands a mean judge score of 3.64/5 and a 91% convincing rate, re-discovering five of VAmPI's real issues entirely autonomously. Once the open-category list is exhausted after iteration 11, every quality signal drops sharply across iterations 12–16 (mean score 2.40/5, 40% convincing), with degenerate repeated or placeholder output in four of the five remaining iterations. Iteration 11 is a distinct, more concerning failure: the model's own reasoning

correctly identifies that its candidate endpoint is on an explicit avoid-list, reasons through

an alternative - and then targets the avoid-listed endpoint anyway. Because the loop's own status-code heuristic is a

cheap first-pass signal, every category it labeled confirmed was cross-checked against the judge's independent read; only 5 of 7 held up (Appendix A), with `rate_limit_missing` and `excessive_data_exposure` both clearing the heuristic while failing the judge's read of the underlying reasoning.

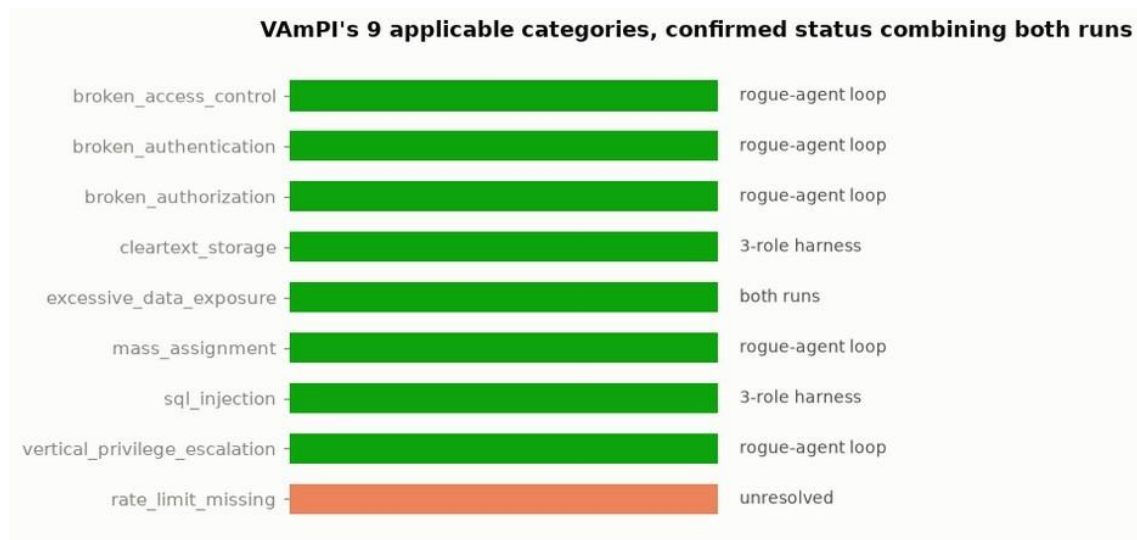


Figure 4. VAmPI's 9 applicable vulnerability categories, confirmed status once both evaluation runs are combined. Green bars are confirmed, with the responsible run labeled at right.

Cost Efficiency: 59–85% Fewer Input Tokens at Near-parity Quality

Because the tuned soft-prompt prefix is baked into the Strategist's weights rather than re-sent as a written master prompt on every call, the Strategist uses substantially fewer input tokens per call than the same base model instructed conventionally, at every scale tested: 59% fewer at the 362-item Test 2 scale (212 versus 87 mean input tokens) and 85% fewer on the VAmPI harness (119 versus 18). The corresponding quality gap is small at the scale that matters most: soft-prompt Qwen (3.23/5) trails base Qwen with the full master prompt (3.38/5) by only 0.15 points at 362 items, while the smaller 15-item pilot had overstated the gap (3.27 vs. 4.20). Critically, the token reduction is essentially constant (59–60%) regardless of whether soft-prompt wins, ties, or loses on quality for a given pattern (Table 1) - it is a mechanical saving from never re-sending the master prompt, not one that only appears in favorable cases.

Table 1. Outcome-group summary across all 362 Test 2 items.
Token reduction is constant across wins, ties, and losses.

Group	n	Base	Soft	Tok. ↓
Soft wins (6 pat.)	82	2.91	3.39	59.9%
Tie (9 pat.)	126	3.48	3.47	59.1%
Base wins (11 pat.)	154	3.53	2.95	58.6%

Where Soft-prompt Tuning Genuinely Wins

Comparing the soft-prompt Strategist against the base+master-prompt condition on the substance of the proposed test, not prose length, shows soft-prompt is a real competitor on technique. Its clearest wins at the 362-item scale are `broken_authentication` (+0.65), `csrf` (+0.57), `idor_sequential_id` (+0.57), and `xxe_injection` (+0.42); its clearest losses are `path_traversal` (−1.14), `stored_xss` (−0.93), `clickjacking` (−0.93), and `insecure_design` (−0.86). The complete 26-pattern table is reported in Appendix B.

Within just the six patterns where soft-prompt wins on aggregate, it wins or ties on 78% of individual items and loses on only 23%. Its correct rate within this set (60%) is above its own 362-item overall average (55%), while base Qwen's correct rate on these same six patterns (49%) is below its own overall average (60%) - base is specifically stumbling on these particular patterns while soft-prompt holds or improves on its usual performance. A second, hand-scored technique-level comparison at the VAmPI-harness scale (18 comparable categories, 20 instances) found soft-prompt's proposed technique outright stronger in 3 (`nosql_injection`, `cleartext_storage`, `stored_xss`), tied in 8, and weaker in 7 - matching or beating master's technique in 11 of 18 categories. The full 20-instance comparison, including the specific reason for every verdict, is reported in Appendix C. The seven losses sort into four nameable, independently fixable failure modes: fabricating a result instead of proposing a test (`sql_injection`, `misconfiguration`), an invalid

technique for the vulnerability class (vertical_privilege_escalation), no real exploit attempt at all (two broken_access_control instances), a non-actionable

request for information (rate_limit_missing), and circular reasoning that never resolves to a concrete step (csrf).

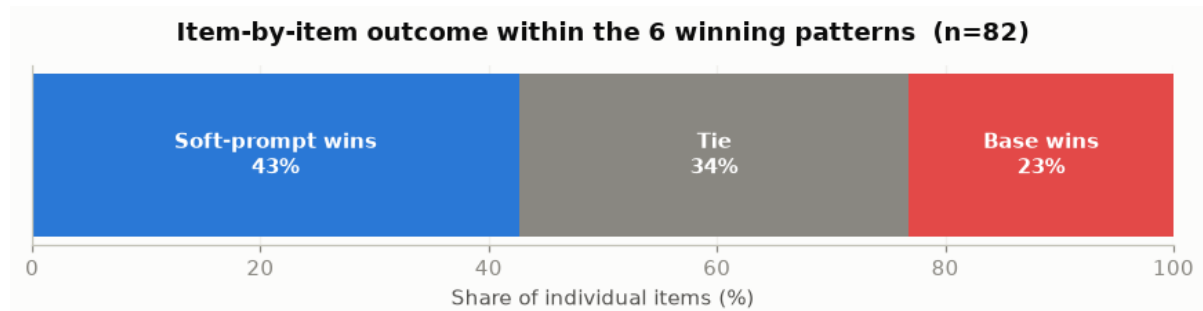


Figure 5. Item-level outcome (soft-prompt wins / ties / base wins) within the six vulnerability patterns where soft-prompt's aggregate mean score exceeds the base+master-prompt condition (n=82 items).

Live-target Validation: Vulnerabilities Found

Against the self-hosted, source-verified VAmPI target, the three-role harness confirmed 3 of 9 applicable categories with an exact 1:1 correspondence between “both roles worked” and “confirmed”: unauthenticated plaintext-password exposure via GET /users/v1/_debug (excessive_data_exposure and cleartext_storage both fire on the same endpoint), and a real SQL syntax error on GET /users/v1/{username} (sql_injection). The mass_assignment case is the clearest single quick-fix: the Strategist's hypothesis was correct and judge-approved (5/5), but the Executor's registration payload was malformed, returning a 400 rather than the 201 a correctly-shaped request would produce.

*Inconclusive due to a specific Executor tool-call bug (wrong method, unsubstituted placeholder, or malformed payload), not a target-security or Strategist-reasoning failure; see Appendix A for the Strategist's judge score on each. Against the public OWASP Juice Shop demo, the three-role harness produced three genuine, correctly attributed findings - unauthenticated IDOR via GET /rest/products/{id}/reviews, broken access control via unauthenticated GET /rest/user/whoami, and mass assignment via POST /api/Users with an injected isAdmin:true field - plus one plausible-but-unconfirmed SQL injection (a real HTTP 500 on a crafted query). Three-quarters of the applicable Juice Shop requests (15 of 20) hit HTTP 503 because the public demo instance

was down for most of this run; the complete 20-row table, including every downtime case, is reported in Appendix D. Pooling all three live-target runs, 11 distinct, corroborated vulnerabilities were confirmed plus 1 plausible finding, out of 29 total applicable tests across the two live targets - a genuine positive result, though the inconclusive bucket (55% of all 38 pooled attempts) remains larger than the confirmed bucket (22%), and is reported as such rather than minimized. The autonomous rogue-agent loop converts attempts into confirmed findings at roughly twice the rate of the single-pass harness on the same VAmPI target (67% vs. 33%).

Table 2. VAmPI three-role harness, all 9 applicable categories.

Category	Status	Outcome
excessive_data_exposure	200	Confirmed
cleartext_storage	200	Confirmed

Category	Status	Outcome
sql_injection	500	Confirmed
mass_assignment	400	Inconclusive*
vertical_priv_escal.	400	Inconclusive*
broken_authentication	404	Inconclusive*
broken_access_control	404	Inconclusive*
broken_authorization	405	Inconclusive*
rate_limit_missing	404	Heuristic artifact

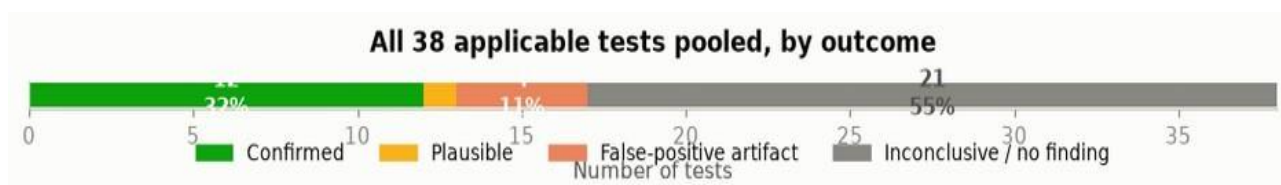


Figure 6. All 38 applicable tests, pooled across the Juice Shop harness, the VAmPI three-role harness, and the VAmPI rogue-agent loop, reclassified into four evidence-backed outcome buckets.

Honest Limitations

Two results are reported here without softening. Against a frontier model with no scaffolding, the soft-prompt Strategist is weak: Gemini scores 4.94/5 with a 98% correct rate on 47 unscaffolded situations, while the soft-prompt Strategist manages only 2.53/5 and 38%, with a substantial share of responses trailing off mid-sentence. This is expected given the parameter-count difference and is a boundary condition on applicability, not evidence against the pipeline - Sections 5.3–5.4 show the more relevant comparison, the same base model prompted two different ways, is far closer. Multi-step reasoning chains also degrade sharply with depth: a five-step chained-reasoning test scores a mean of 2.0/5, the weakest condition in the project. Three of five chains score 1/5, either repeating an already-executed action or drifting into unrelated text by steps 4–5; only two chains sustain a coherent investigation through all five steps, and one of those (path_traversal) escalates unrealistically fast from a proof-of-concept file write into simulated web-shell deployment and persistence - a scope-discipline finding worth flagging independently of its 3/5 coherence score.

Table 3. Cross-test summary, every controlled evaluation condition, ordered by mean score.

Condition	n	Score
Test 2 pilot - Base+master	15	4.20
Test 1 - Gemini	47	4.94
Test 2 scale - Base+master	362	3.38
Test 2 scale - Soft-prompt	362	3.23
Test 2 pilot - Soft-prompt	15	3.27
Test 1 - Soft-prompt	47	2.53
Test 3 - 5-step chain	5	2.00

CONCLUSION

This project set out to test a narrow, causal question: inside a two-role, small-language-model pentesting pipeline, does fine-tuning the execution role change what the pipeline can accomplish, independent of the reasoning role's own quality? The controlled ablation in Section 5.1 answers this directly - holding the 8B Strategist and the attack loop fixed, LoRA fine-tuning of a 1.7B Executor alone takes an autonomous vulnerability-finding loop from 0 of 9 confirmed categories to 7 of 9, and combining that result with a separate evaluation harness resolves 8 of 9 categories against a real, self-hosted target. The mechanism is traceable to a specific, narrow failure: an untuned model choosing a plausible-looking but methodologically wrong HTTP method, fixed by 393 training examples and 17.4M LoRA parameters - a fine-tuning run that fits on a single consumer-class GPU, at a

fraction of the cost of scaling up to, or fine-tuning, a larger reasoning model instead. Layered onto this, the soft-prompt-tuned Strategist delivers 59–85% lower per-call input-token cost than the same model fully prompted, at a quality cost that is small and, on several named vulnerability categories, absent entirely. Every model in this pipeline's live decision loop is 8B parameters or smaller, and both fine-tuning runs together train well under 20M parameters - none of this required a larger or frontier-scale model anywhere in the loop; it required specializing the small models already in hand.

The most significant unexpected result was less about accuracy and more about mechanism: the two rogue-agent runs that reached 7 of 9 confirmed categories used, respectively, a fine-tuned 1.7B Executor and an un-tuned 8B model standing in for the Executor role, converging on similar coverage through very different routes. This suggests the bottleneck this project identified, correct tool-call formatting, is fixable either by scale or by targeted fine-tuning, but fine-tuning is the far cheaper route per call. The project is equally clear about where the current approach does not yet work: an 8B model with only a learned soft-prompt is not competitive against a frontier model with no scaffolding, and reasoning quality degrades sharply as tool-using chains grow past three or four steps. If this project were to continue, the next steps would stay within the same small-model, single-GPU-fine-tuning constraint rather than defaulting to a larger model: augmenting the Executor's training data with explicit adversarial payloads for the two vulnerability categories it can locate but not exploit, adding multi-step chains to the Strategist's training data to test whether Test 3's degradation is a data gap rather than an architectural ceiling, training an explicit “no viable categories remain” exit behavior for the rogue-agent loop, and testing against a third, undocumented live target to separate genuine reasoning generalization from familiarity with VAmPI's and Juice Shop's well-known challenge structure.

ACKNOWLEDGMENTS

The team thanks Prof. Haisav Chokshi for guidance throughout the AAI-590 capstone sequence, and acknowledges the maintainers of the VAmPI and OWASP Juice Shop projects, whose purpose-built vulnerable applications made live, verifiable evaluation of this project's pipeline possible.

REFERENCES

- [1] American Psychological Association. (2025). In-text citations. <https://apastyle.apa.org/style-grammar-guidelines/citations>

- [2] Andreucci, F., & Castiglione, A. (2026). Automation-Exploit: A multi-agent framework for risk-mitigated automated exploitation using digital twins. arXiv preprint.
- [3] Checkmate authors. (2025). Checkmate: A Planner-Executor-Perceptor architecture for automated penetration testing. arXiv:2512.11143.
- [4] Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., & Rass, S. (2024). PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In Proceedings of the 33rd USENIX Security Symposium.
- [5] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. arXiv:2106.09685.
- [6] Lester, B., Al-Rfou, R., & Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 3045–3059).
- [7] Moonshot AI. (2026). Kimi K2 technical report and model documentation.
- [8] PentestGPT v2 authors. (2026). Task difficulty assessment and evidence-guided attack tree search for automated penetration testing. arXiv:2602.17622. reddelexc. (2026). hackerone-reports: A curated dataset of publicly disclosed HackerOne vulnerability reports [GitHub repository].
- [9] Shen, X., Wang, Y., Chen, Z., & Liu, J. (2025). PentestAgent: Incorporating LLM agents into automated penetration testing. In Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS).
- [10] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., & Hashimoto, T. B. (2023). Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models.
- [11] Xu, L., Xie, H., Qin, S. Z. J., Tao, X., & Wang, F. L. (2023). Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. arXiv preprint.
- [12] Yang, J., Prabhakar, A., Narasimhan, K., & Yao, S. (2023). InterCode: Standardizing and benchmarking interactive coding with execution feedback. In Advances in Neural Information Processing Systems, 36.
- [13] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In Proceedings of the 11th International Conference on Learning Representations (ICLR).
- [14] Zhang, A., Perry, N., Dutta, R., Kang, D., & Zou, J. (2024). Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. arXiv preprint.
- [15] Zhu, M., Wang, Z., Li, K., & Anastasopoulos, A. (2024). NYU CTF Bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security. In Advances in Neural Information Processing Systems, 37.

APPENDIX A: FULL ROGUE-AGENT LOOP ITERATION LOG (VAMPI)

Complete, unabridged 16-iteration log from the autonomous rogue-agent loop against VAmPI with the LoRA-fine-tuned Executor (Section 5.1–5.2), in original run order. Iterations 12–16 occurred after the open-category list was exhausted; category labels there are placeholder or off-schema. Iteration 11 is the self-contradiction case discussed in Section 5.2: the model's reasoning correctly identified its target endpoint as avoid-listed, then targeted it anyway.

Table 4: A1. Full rogue-agent loop iteration log, all 16 iterations, original run order.

Iter.	Category	Tool call → path	Status	Judge	Convincing	Matched
1	broken_authentication	POST /users/v1/login	200	4/5	Yes	Yes
2	broken_access_control	GET /users/v1/_debug	200	5/5	Yes	Yes
3	cleartext_storage	POST /users/v1/register	200	5/5	Yes	Yes
4	cleartext_storage (retry)	POST /users/v1/register	400	3/5	Yes	Yes
5	sql_injection	POST /users/v1/register	400	4/5	Yes	Yes
6	sql_injection (retry)	POST /users/v1/register	200	3/5	Yes	Yes
7	broken_authorization	GET /users/v1/_debug	200	5/5	Yes	Yes
8	mass_assignment	POST /users/v1/register	200	3/5	Yes	Yes
9	rate_limit_missing	POST /users/v1/register	200	2/5	Yes	No
10	vertical_privilege_escalation	GET /users/v1/_debug	200	5/5	Yes	Yes
11	excessive_data_exposure (avoid-list)	GET /users/v1/_debug	200	1/5	No	No

	override)					
12	(category list exhausted)	GET /books/v1/The Book Thief	401	2/5	No	Yes
13	(category list exhausted)	GET /users/v1/name1	200	4/5	Yes	Yes
14	(category list exhausted)	GET /api/v1/endpoint	404	1/5	No	No
15	(category list exhausted)	GET /users/v1/register	404	1/5	No	No
16	insecure_direct_object_reference (off-schema)	GET /users/v1/name1	200	4/5	Yes	Yes

Phase Performance and Heuristic Corroboration

Table 5: A2. Rogue-agent loop performance before versus after the open-category list ran dry.

Phase	Iterations	Mean judge score	Convincing rate	Tool-call matched rate
1 — categories still open	11	3.64 / 5	91%	82%
2 — category list exhausted	5	2.40 / 5	40%	60%

Table 6: A3. Every category the loop's own status-code heuristic marked “confirmed,” cross-checked against the judge's independent verdict on the actual reasoning and tool call.

Category	Judge score	Convincing	Tool-call matched	Judge-corroborated?
broken_access_control	5/5	Yes	Yes	Yes
broken_authorization	5/5	Yes	Yes	Yes
vertical_privilege_escalation	5/5	Yes	Yes	Yes
broken_authentication	4/5	Yes	Yes	Yes
mass_assignment	3/5	Yes	Yes	Yes
rate_limit_missing	2/5	Yes	No	No
excessive_data_exposure	1/5	No	No	No

APPENDIX B: COMPLETE PER-PATTERN COMPARISON, ALL 26 VULNERABILITY PATTERNS

Base Qwen with the full written master prompt versus soft-prompt Qwen, all 26 trained vulnerability patterns at the 362-item Test 2 scale (Section 5.4), ordered strongest soft-prompt win to weakest loss. A delta within ± 0.15 is within one-item noise at this sample size ($n \approx 14$ per pattern) and is read as a tie.

Table 7: B1. Complete per-pattern comparison, base+master-prompt versus soft-prompt Qwen, all 26 patterns.

Vulnerability pattern	n	Base mean	Soft-prompt mean	Δ (soft – base)
broken_authentication	14	3.21	3.86	+0.65
csrf	14	2.50	3.07	+0.57
idor_sequential_id	14	2.57	3.14	+0.57
xxe_injection	12	3.50	3.92	+0.42
broken_authorization	14	3.21	3.57	+0.36

International Journal of Innovations In Science Engineering And Management

code_injection	14	2.57	2.86	+0.29
misconfiguration	14	2.79	2.93	+0.14
excessive_data_exposure	14	3.36	3.43	+0.07
dom_xss	14	3.36	3.36	0.00 (tie)
business_logic_race_condition	14	3.86	3.86	0.00 (tie)
cleartext_storage	14	3.43	3.43	0.00 (tie)
ssrf_via_webhook	14	3.21	3.14	-0.07
open_redirect	14	3.43	3.36	-0.07
resource_exhaustion_dos	14	4.36	4.29	-0.07
reflected_xss	14	3.57	3.43	-0.14
broken_access_control	14	3.64	3.43	-0.21
xss_generic	14	4.07	3.86	-0.21
sql_injection	14	3.71	3.36	-0.36
rate_limit_missing	14	3.00	2.64	-0.36
vertical_privilege_escalation	14	3.43	3.00	-0.43
insecure_deserialization	14	2.43	2.00	-0.43
command_injection	14	2.93	2.36	-0.57
insecure_design	14	3.57	2.71	-0.86
clickjacking	14	3.57	2.64	-0.93
stored_xss	14	4.29	3.36	-0.93
path_traversal	14	4.21	3.07	-1.14

Appendix C: Full Vampi-harness Technique-level Comparison, All 20 Instances

Soft-prompt Qwen's proposed technique versus base Qwen with the full master prompt, judged on substance rather than judge score alone, all 20 category instances (18 distinct categories; 2 tested twice) from the VAmPI three-role harness (Section 5.4). Ordered soft-prompt wins first, then ties, then losses, then instances with no master-prompt answer to compare against.

Table 8: C1. Technique-level verdict for all 20 VAmPI-harness category instances.

Category	Endpoint	Verdict	Note
nosql_injection	/rest/products/search	Soft wins	Soft gives a literal NoSQL operator payload (\$lt) naming the exact injection point; master's approach is vaguer.
cleartext_storage	/rest/user/login	Soft wins	Master's technique (inspect server logs/database) isn't executable with this harness's external HTTP-only toolset; soft's is.
stored_xss	/rest/products/reviews	Soft wins	Both propose a script-payload review test; soft's payload choice is more evasive against basic filtering.
idor_sequential_id	/rest/products/{id}/reviews	Tie	Same technique (increment sequential ID); master adds the security-implication framing, soft is more terse.
idor_sequential_id (2nd)	/rest/products/{id}/reviews	Tie	Same technique; master checks three candidate IDs up front, soft checks them incrementally.
excessive_data_exposure	/rest/user/whoami	Tie	Same technique: inspect the whoami response for over-exposed

			fields; master lists slightly more example fields.
broken_authentication	/rest/user/login	Tie	Both propose a credential-guessing login test against a known username; comparable rigor.
reflected_xss	/rest/products/search	Tie	Same technique: inject a script payload into the search parameter and check for reflection.
mass_assignment	/api/Users	Tie	Both converge on the identical, correct exploit: POST with an unauthorized isAdmin field.
path_traversal	/ftp	Tie	Same technique and target file; soft's path uses more '../' traversal segments than strictly needed.
open_redirect	/redirect	Tie	Same technique: pass an external URL in the redirect parameter and check if the app follows it.
broken_access_control	/rest/user/whoami	Master wins	Off-target technique: soft's hypothesis argues about password-change flow rather than testing the GET /whoami call under test.
broken_access_control (2nd)	/rest/user/whoami	Master wins	No exploit attempt: soft's hypothesis only calls the endpoint to "find the user's name," never proposing an access-control test.
vertical_privilege_escalation	/api/Users	Master wins	Invalid technique: soft conflates creating a duplicate user ID with privilege escalation, which is not the same vulnerability class.
rate_limit_missing	/rest/user/login	Master wins	Non-actionable: soft's hypothesis asks to be told the rate-limit configuration rather than proposing an actual test.
sql_injection	/rest/products/search	Master wins	Fabricated result: soft asserts, without evidence, that the injection attack already succeeded, rather than proposing a test.
misconfiguration	/ftp	Master wins	Fabricated result: soft fabricates a plausible-looking directory listing rather than proposing a step to obtain one.
csrf	/rest/user/change-password	Master wins	Never resolves to a step: soft loops through the same SameSite-cookie reasoning repeatedly without committing to an action.
business_logic_race_condition	/rest/basket/1	N/A	No master-prompt answer recorded for this instance; soft's own technique (concurrent request race) is sound on inspection.
broken_authorization	/rest/basket/1	N/A	No master-prompt answer recorded for this instance.

APPENDIX D: COMPLETE OWASP JUICE SHOP HARNESS RESULTS AND ATTRIBUTION

All 20 applicable test rows from the OWASP Juice Shop three-role harness (Section 5.5), including the 15 rows affected by public-demo downtime (HTTP 503), reported here in full for transparency rather than omitted.

Table 9: D1. OWASP Juice Shop three-role harness, all 20 applicable test rows.

Category	Endpoint	Status	Judge	Class
mass_assignment	/api/Users	201	5/5	Confirmed
idor_sequential_id	/rest/products/{id}/reviews	200	5/5	Confirmed
broken_access_control	/rest/user/whoami	200	1/5*	Confirmed
sql_injection	/rest/products/search	500	3/5	Plausible
nosql_injection	/rest/products/search	503	2/5	Heuristic artifact
rate_limit_missing	/rest/user/login	503	1/5	Heuristic artifact
reflected_xss	/rest/products/search	200	5/5	Inconclusive — payload not reflected

International Journal of Innovations In Science Engineering And Management

misconfiguration	/ftp	503	4/5	Inconclusive — target down
excessive_data_exposure	/rest/user/whoami	503	5/5	Inconclusive — target down
broken_authentication	/rest/user/login	503	4/5	Inconclusive — target down
cleartext_storage	/rest/user/login	503	3/5	Inconclusive — target down
idor_sequential_id (retest)	/rest/products/{id}/reviews	503	4/5	Inconclusive — target down
broken_access_control (retest)	/rest/user/whoami	503	5/5	Inconclusive — target down
vertical_privilege_escalation	/api/Users	503	1/5	Inconclusive — target down
stored_xss	/rest/products/reviews	503	2/5	Inconclusive — target down
path_traversal	/ftp	503	5/5	Inconclusive — target down
open_redirect	/redirect	503	4/5	Inconclusive — target down
business_logic_race_condition	/rest/basket/1	503	1/5	Inconclusive — target down
broken_authorization	/rest/basket/1	503	1/5	Inconclusive — target down
csrf	/rest/user/change-password	503	2/5	Inconclusive — target down

Attribution: Strategist Vs. Executor, Both Harnesses

Because every harness scores the Strategist's hypothesis and the Executor's tool call independently, every non-success case can be attributed to a specific role rather than to "the model" as an undifferentiated whole (Section 6, Discussion).

Table 10: D2. Attribution of every non-"both worked" case to a specific role, pooled across the Juice Shop and VAmPI three-role harnesses.

Harness	Attribution	n	Mean score	Categories
Juice Shop	Both roles worked	12	4.33	idor_sequential_id (×2), broken_access_control, excessive_data_exposure, broken_authentication, reflected_xss, misconfiguration, mass_assignment, path_traversal, open_redirect, cleartext_storage, sql_injection
VAmPI	Both roles worked	3	5.00	excessive_data_exposure, cleartext_storage, sql_injection
Juice Shop	Executor failed alone	2	2.00	nosql_injection, stored_xss
VAmPI	Executor failed alone	1	4.00	mass_assignment
Juice Shop	Strategist failed alone	3	1.33	broken_access_control (2nd instance), vertical_privilege_escalation, csrf
VAmPI	Strategist failed alone	0	—	(none)
Juice Shop	Both roles failed	3	1.00	rate_limit_missing, business_logic_race_condition, broken_authorization
VAmPI	Both roles failed	5	—	broken_authentication, broken_access_control, broken_authorization, vertical_privilege_escalation, rate_limit_missing

Test 3: Complete Five-chain Results

Table 11: D3. Complete Test 3 results, all five 5-step chains, strongest first.

Chain	Vulnerability pattern	Score	Correct?	What happened
4	stored_xss	4/5	Yes	Coherent, on-hypothesis investigation sustained through all 5 steps.
3	path_traversal	3/5	Yes	Stays on-topic throughout, but escalates unrealistically fast from a single successful file write to uploading a web shell, obtaining a reverse shell, and establishing persistence — well past hypothesis confirmation into unauthorized post-exploitation, which should have stopped at proof-of-concept.
0	business_logic_race_condition	1/5	No	Stalls after step 2: steps 3–5 re-propose the same already-executed action with no new evidence or progression.
1	open_redirect	1/5	No	Steps 1–3 stay loosely on-hypothesis; step 4 degenerates into unrelated buzzword text with no real test action, and step 5 continues the derailment with an irrelevant, disconnected finding.
2	csrf	1/5	No	Never actually tests the stated CSRF hypothesis; drifts through unrelated topics by steps 3–5, padded with fabricated regulatory citations.