

RangeFactory: Scalable Construction of Multi-Hop Cyber Ranges

Hanlin Jiang¹, Puyi Wang¹, Jiandong Jin¹, Shaofei Li¹, Zhan Shen¹, Pengli Wang¹, Ziming Wang¹, Yifeng Cai¹, Ning Jia², Yuxin Ren², Peng Jiang³, Yao Guo¹, Ding Li^{1*}

¹Key Laboratory of High Confidence Software Technologies, Peking University, Ministry of Education

²Huawei Technologies Co., Ltd.

³Southeast University

Abstract

Real-world cyberattacks often require sustained progress across multiple hosts and network segments, making multi-hop cyber ranges essential infrastructure for studying and improving LLM agents' ability to sustain complete attack chains. Prior work has scaled isolated vulnerability tasks and constructed multi-host scenarios from manually specified vulnerability semantics. However, they are still unable to automatically orchestrate the growing supply of vulnerability environments into end-to-end validated multi-hop ranges. To this end, we present **RangeFactory**, an automated cyber-range orchestration framework that constructs multi-hop cyber ranges at scale from isolated vulnerability environments. RangeFactory formulates range construction as dependency resolution: it extracts dependency information from agents' actual attacks against real vulnerabilities, resolves known dependencies through template-guided orchestration, and uses end-to-end attack execution to validate runtime dependencies that emerge after composition.

Using RangeFactory, we construct **RangeBench** with 1,148 validated range instances spanning 287 distinct attack chains and evaluate frontier attack agents across attack depth, network scale, and task information. Among runs that compromise the entry vulnerability, 24.5–47.0% still fail to complete the remaining attack path, revealing a substantial sustained-compromise gap between establishing an initial foothold and completing a multi-hop attack. RangeFactory further produces a corpus of 5,541 outcome-annotated multi-hop attack trajectories, providing execution data for attack-process analysis and future agent training.

1 Introduction

Large language model agents are evolving from executing isolated instructions to autonomously completing long-horizon tasks in complex environments. This trend is extending agent autonomy to cybersecurity, where real-world intrusions often require sustained progress across multiple network positions (Deng et al. 2024; Fang et al. 2024). A cyber range is an interactive network environment for executing and validating cyberattacks; a multi-hop cyber range further requires an attacker to use footholds established by earlier compromises to reach targets in distinct network segments. Such environments therefore provide the infrastructure for

systematically studying whether attack agents can sustain multi-hop compromise (Folkerts et al. 2026).

Prior work has developed two classes of resources for evaluating attack agents. One line scales the conversion of real vulnerabilities into executable single-vulnerability tasks, but each task ends at an isolated exploit and cannot evaluate sustained compromise across network positions (Luo et al. 2026). Another line uses multi-host cyber ranges to evaluate end-to-end attacks, but existing benchmarks depend on a small collection of expert-designed, fixed scenarios (Liu et al. 2026). Automated scenario-generation systems can expand environment deployment, yet still require experts to specify cross-vulnerability dependencies in advance (Skandylas and Asplund 2026). As a result, existing works cannot automatically set up the environments that support complete attack chains. Therefore, this paper addresses how to automatically transform the growing supply of single-vulnerability environments into multi-hop cyber ranges with executable end-to-end attack paths.

The central challenge in this transformation is automatically making independently exploitable vulnerabilities execute consecutively within one attack. This requires resolving both capability and runtime dependencies. A capability dependency determines whether adjacent attacks can connect; for example, file-read access cannot support a subsequent exploit that requires command execution. A runtime dependency determines whether the concrete attack procedure can operate from its new foothold; for example, a network policy may block an exploit's callback (Nguyen et al. 2025). Directly composing vulnerability environments and executing every combination creates a rapidly growing candidate space. Dependency-guided orchestration can eliminate known dependency conflicts before deployment, while conditions introduced by joint deployment still require complete attack execution. Scalable construction therefore first composes candidates whose dependencies can connect and then executes each end to end to confirm that the path remains feasible.

Based on this observation, we present **RangeFactory**, a multi-agent system that automatically transforms isolated vulnerability environments into validated multi-hop cyber ranges. RangeFactory views range construction as a dependency-resolution process: it characterizes dependencies through actual attacks, resolves them through scenario orchestration, and validates the resulting runtime conditions

*Corresponding author.

through end-to-end execution. Atomization agents attack each CVE environment and probe its post-exploitation capabilities, producing a CVE Capability Atom with dependency information and a reusable attack procedure. Orchestration agents generate attack-path templates and bind compatible Atoms, after which Validation agents execute the complete attack against hidden objectives. Diagnostic feedback supports attack retries and subsequent path construction. Every retained range thereby contains a path successfully exercised during construction.

Using 239 CVE Capability Atoms and 20 enterprise-style network configurations, RangeFactory produces 1,840 candidate range instances. End-to-end validation retains 1,148 instances spanning 287 distinct ordered attack chains, for a validation rate of 62.4%. We organize them into **RangeBench** and evaluate frontier attack agents across attack depth, network scale, and task information. The results reveal a substantial sustained-compromise gap: success on an entry vulnerability does not reliably translate into completing the attack when the same path is extended across additional network layers. Beyond evaluation, RangeFactory constructs a corpus of 5,541 outcome-annotated multi-hop interaction trajectories from the same ranges. Validated demonstrations and autonomous outcomes share verifier-derived milestone annotations that may support future attack-agent training.

Our contributions are summarized as follows:

- **RangeFactory.** We introduce an automated multi-agent framework that derives vulnerability dependencies through actual attacks and combines orchestration with end-to-end validation to construct multi-hop cyber ranges at scale.
- **RangeBench and Empirical Findings.** We construct RangeBench with 1,148 validated range instances spanning 287 distinct attack chains and systematically study how attack depth, network scale, and task information affect frontier attack agents, revealing a substantial gap between early compromise and sustained multi-hop compromise.
- **Outcome-Annotated Attack Trajectories.** We collect a corpus of 5,541 multi-hop trajectories pairing validated demonstrations with autonomous outcomes on the same ranges, with verifier-derived milestone annotations that may support future attack-agent training.

2 Related Work

Executable Cybersecurity Tasks and Agent Data. Recent work has transformed real vulnerabilities into executable tasks, grounding cyber-agent evaluation in actual interaction (Zhu et al. 2025; Wang et al. 2026). Automated vulnerability reproduction further converts sparse CVE meta-data into complete single-vulnerability task packages (Ullah et al. 2026; Luo et al. 2026). Complementary benchmark efforts package professional CTF and penetration-testing challenges as reproducible interactive environments (Shao et al. 2024; Zhang et al. 2025). Beyond benchmark construction, cybersecurity-agent data have been scaled from CTF artifacts: runtime-free methods synthesize trajectories from writeups, while execution-grounded platforms collect

verified rollouts from containerized challenges (Zhuo et al. 2025a,b). These resources remain centered on individual vulnerabilities or challenges. Multi-host cyber-range benchmarks evaluate how agents expand an initial foothold across a network, but rely on a small, fixed collection of ranges (Liu et al. 2026, 2025). RangeFactory addresses this environmental bottleneck and collects outcome-annotated multi-hop interactions as a downstream artifact of range construction and evaluation.

Automated Cyber-Range Construction. A separate line of work automates cyber-range construction. Attack-graph methods derive multi-host attack chains from formal system models (Ou et al. 2005). Cyber-range generators instantiate expert-authored scenario models (Pham et al. 2016; Schreuders et al. 2017), while model-driven approaches support automated deployment validation (Costa, Russo, and Armando 2022; Russo, Costa, and Armando 2020). LLM-based systems further translate natural-language range requirements into platform-specific configurations and deployment artifacts (Lupinacci et al. 2025; Rizos et al. 2025), while constraint-based synthesis uses model finding and capability semantics to generate enterprise scenarios at scale (Skandylas and Asplund 2026). These approaches automate cyber-range construction once authored scenario descriptions or predefined composition semantics are available, but adding a new vulnerability still requires its attack dependencies to be specified in advance. RangeFactory obtains these dependencies from successful attacks against existing vulnerability environments, uses them to orchestrate candidate chains, and then executes each deployed chain end to end to detect unresolved runtime incompatibilities introduced by composition.

3 System Design

3.1 Insight and Overview

Deploying independently exploitable CVE environments together does not yield a valid multi-hop attack chain unless both capability and runtime dependencies hold. A capability dependency concerns what an attack requires and produces: a preceding compromise must provide the capability needed by the next exploit. File-read access, for example, cannot support an exploit that requires command execution. A runtime dependency concerns whether the concrete attack procedure can operate from its new execution position. A blocked reverse connection or a missing tool on the compromised host can prevent exploitation even when the capability interface matches. Either violation breaks the consecutive execution of otherwise exploitable CVEs.

RangeFactory views multi-hop range construction as dependency resolution: actual attacks characterize dependencies, orchestration resolves them, and end-to-end execution confirms candidate-chain executability. Single-vulnerability attacks reveal the capabilities required and produced and record the runtime requirements observed during exploitation. Orchestration connects consecutive capabilities and uses the network template to constrain known runtime dependencies. Because the new footholds and joint deployment can introduce conditions that were absent from the single-vulnerability environment, RangeFactory then executes the

Field	Semantics
<i>Capability Interface</i>	
<code>exploit_access</code>	Structured preconditions: attack vector, privilege, and service protocol/port.
<code>capability_grants</code>	Verified post capability from a fixed ontology, with principal and evidence.
<i>Execution Artifacts</i>	
<code>exploit_guide</code>	Successful procedure with tool, material, authentication, and callback requirements.
<code>poc_materials</code>	Exploit artifacts referenced by <code>exploit_guide</code> .

Table 1: Description of CVE Capability Atom fields.

complete attack to expose remaining conflicts. This division prunes clearly infeasible compositions before deployment while retaining only chains successfully exercised after composition.

Figure 1 presents RangeFactory’s multi-agent workflow. Atomization produces CVE Capability Atoms, Orchestration binds them into candidate ranges, and Validation exercises the complete attack before acceptance. Diagnostic feedback is routed to attack execution for bounded retry or to chain construction as persistent incompatibility evidence.

3.2 CVE Capability Atom

RangeFactory represents each successfully attacked vulnerability environment as a CVE Capability Atom, which binds the deployable environment to dependency information and execution artifacts. Table 1 summarizes its four fields. `exploit_access` records the structured conditions under which the exploit was exercised. `capability_grants` binds each verified post-compromise capability to the compromised principal and its evidence; only verified grants participate in orchestration. A grant’s host scope is assigned when the Atom is bound to a template position. `exploit_guide` preserves the validated procedure and observed runtime requirements, while referenced PoC files are packaged as `poc_materials`. The interface supports dependency matching, and the execution artifacts allow the vulnerability to be exercised again in a composed range.

The capability ontology represents command execution `execute_command`, file access `read_file` and `write_file`, network position `network_vantage`, credential discovery `read_credential`, and authentication `authenticate`. For example, when `exploit_access` requires an internal service, the preceding compromise must provide `network_vantage`. A TCP callback requirement is instead recorded in `exploit_guide` for runtime compatibility checking. Capability matching uses only grants confirmed by probes, avoiding chains supported by an Agent’s unverified inference. Runtime dependencies vary with the composed environment and cannot be completely exposed by a single-vulnerability attack, so the Atom records observed requirements and leaves the remainder to end-to-end validation.

3.3 Atomization

Exploiter. RangeFactory deploys a reproducible vulnerability environment, configures a private objective aligned with the vulnerability impact, and provides the Exploiter with its reproduction context and available exploit materials. The Exploiter adapts the attack according to environment feedback, while an independent verifier confirms success against the private objective. After a successful attack, it distills the procedure, exploit conditions, and observed runtime dependencies into `exploit_guide`. This establishes for every Atom an execution-confirmed attack for subsequent range construction.

Explorer. Reaching the vulnerability objective confirms its expected security impact; orchestration also needs to know which subsequent actions that impact enables. The Explorer therefore reuses the Exploiter’s environment and established foothold, selecting capability probes according to the available access mechanism and probe feedback. For example, it directs the compromised host to a designated endpoint to confirm `network_vantage`. Only capabilities confirmed by probes and their corresponding principals enter `capability_grants`. This process converts heterogeneous vulnerability impacts into a uniform interface for subsequent orchestration.

3.4 Orchestrator

Generator. The Generator transforms public architecture references into parameterized enterprise-style templates. Each template defines the network structure and communication policies and marks an attack-chain skeleton from an external entry point to the final objective. It reserves one CVE slot in every traversed layer and independently parameterizes background business assets to control the search space. For example, a three-tier template can traverse a DMZ, an internal application network, and a data network. The template fixes chain structure and network-dependent constraints while keeping Atom selection open.

Composer. The Composer takes a scenario template and a library of verified Atoms as input and constructs candidate bindings in attack-chain order. For each CVE slot, it retrieves Atoms whose service requirements are compatible with the template position and checks whether the capabilities accumulated at preceding positions satisfy the next Atom’s `exploit_access`. After each binding, the Composer associates the Atom’s `capability_grants` with the current network position and updates the capability set through fixed closure rules. For example, verified command execution can support file access and network vantage under the same principal. A partial chain is pruned as soon as a dependency is violated, after which the Composer backtracks to another Atom.

Beyond capability dependencies, the Composer combines the network template with `exploit_guide` to check runtime requirements that can be determined before deployment, such as the attack’s execution position and material delivery. The concrete exploit procedure and runtime dependencies that emerge only in the composed environment are left to

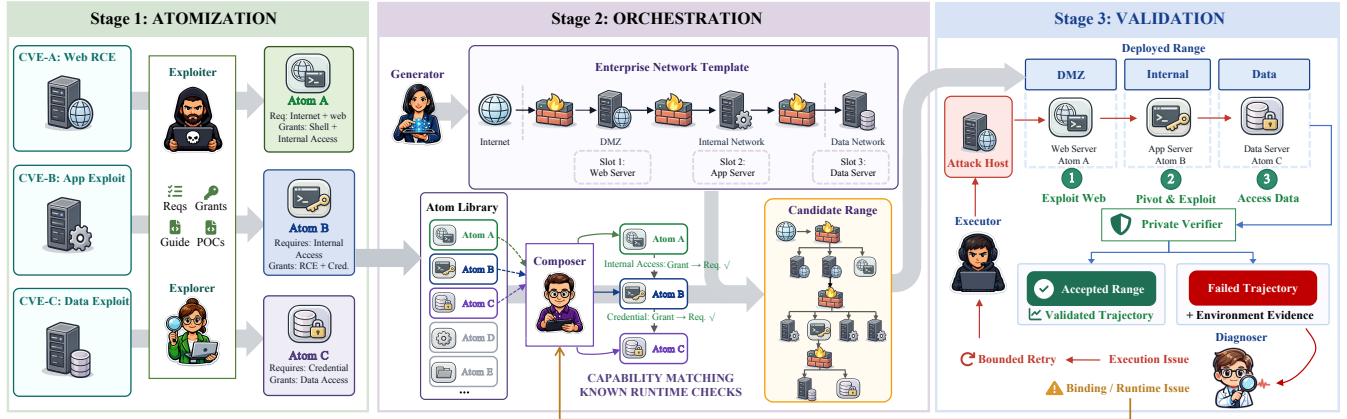


Figure 1: Overview of RangeFactory’s multi-agent workflow for constructing validated multi-hop cyber ranges.

end-to-end execution. Once all slots are bound, the Composer forms an attack graph whose nodes are Atom-bound template positions and whose edges are dependency-satisfying attack transitions. When the candidate set is large, it prioritizes chains that expand coverage of Atom and asset configurations. The output records Atom bindings, capability transitions, and deployment parameters for range instantiation and validation.

3.5 Validation

Executor. After deployment, RangeFactory checks that all nodes and services operate under the network configuration defined by the template. The Executor accesses each Atom’s execution artifacts, organizes the per-hop exploits into an end-to-end attack, and adapts them as the foothold and runtime environment change. A verifier independently checks the ordered private objectives hidden from the Executor. Only a candidate that completes the end-to-end attack is accepted; the complete trajectory and verification result are retained, while failed executions are passed to the Diagnoser.

Diagnoser. The Diagnoser locates the failed step from the trajectory and environment evidence and distinguishes attack execution from chain composition failures. An execution failure is returned to the Executor for a bounded retry that preserves the current context. An unsatisfied runtime dependency instead filters the candidate and produces an incompatibility record associated with the binding conditions. The Composer retrieves these records during later orchestration, allowing evidence from validated failures to improve subsequent candidate generation.

4 Experiment

4.1 Scalable and Reliable Construction of Multi-Hop Ranges

Setup. To evaluate whether RangeFactory can construct multi-hop cyber ranges at scale, we automatically screen and deduplicate vulnerability resources from Vulhub and CVE-Factory (Vulhub 2026; Luo et al. 2026). Eligible resources must provide a deployable network service and allow the

Stage	Transformation
Atomization	257 candidates → 239 Atoms (93.0% successful)
Orchestration	23,789 prop. → 1,840 cand. (92.3% filtered)
Validation	1,840 cand. → 1,148 ranges (62.4% validated)

Table 2: Stage-wise construction and validation results. Prop. and cand. denote proposals and candidates.

vulnerability impact to be verified through network interaction. This process yields 257 Atomization candidates, of which 239 are successfully converted into CVE Capability Atoms (93.0%); the remaining failures arise from missing build artifacts or unstable reproduction. These Atoms cover 122 products and CVEs disclosed between 2010 and 2026. Given public architecture references and the requested scale, the Generator produces five three-tier enterprise-style templates without manual CVE binding or chain construction. Each places one vulnerable node per tier, yielding an entry compromise and two pivots. Although the representation accepts variable-length attack chains, three tiers provide matched one-, two-, and three-hop measurements within the same complete range. Four background-asset scales produce 20 configurations spanning small to medium-sized networks. All construction agents use DeepSeek-V4-Pro; validation receives the Atom execution artifacts, 300 turns, and a one-hour timeout. Source selection and the choice of architecture references are one-time dataset preparation. RangeFactory instantiates private objectives and capability probes from reusable adapters, then performs Atomization, CVE binding, deployment, and end-to-end validation without per-range expert annotation or chain design.

Construction Scale and Coverage. Given 239 CVE Capability Atoms and 20 scenario configurations, the Composer examines 23,789 CVE binding proposals. It rejects 21,949 proposals whose capability or runtime dependencies cannot be satisfied and produces 1,840 candidate range instances, pruning 92.3% before deployment. The candidates form 460 matched groups that instantiate the same template and Atom

chain at four network scales. End-to-end validation retains 287 groups, each associated with a distinct ordered Atom chain, yielding 1,148 range instances that cover 213 Atoms, 113 products, and all 20 configurations. Each chain is instantiated in four matched network contexts whose background assets differ in scale and composition. The resulting benchmark therefore separates attack-chain diversity from variation in the surrounding network.

Execution Grounding. RangeFactory deploys and validates the 1,840 candidate range instances, of which 1,148 complete an end-to-end attack and pass private-objective verification, for an overall validation success rate of 62.4%. Each candidate directly reuses the vulnerability environments and execution artifacts packaged in its Atoms; instantiating it and completing the first validation pass takes 16.2 minutes on average. Because Atomization is performed once for each vulnerability environment, its outputs can be reused across many candidate ranges without repeating the single-vulnerability attack.

Construction Quality. Scale alone does not establish construction quality. We therefore randomly sample 150 accepted ranges for an independent audit that checks whether the injected vulnerability preserves its original security impact and whether the Atom’s access and capability fields agree with the observed attack states. Among the audited ranges, 146 (97.3%) preserve the source vulnerability behavior and 142 (94.7%) agree with the Atom annotations. Separately, every retained range has an attack witness that completes its ordered private objectives; this constructive evidence does not assert the absence of other valid attack chains. The audit supports preservation of vulnerability behavior and dependency semantics.

4.2 RangeBench: Evaluating Multi-Hop Attack Agents

Benchmark Design. We organize 1,148 end-to-end validated range instances—287 distinct ordered attack chains, each instantiated in four matched network contexts—into RangeBench to study sustained execution, exploration, and prior knowledge. A private objective on each vulnerable node defines matched one-, two-, and three-hop success within the same rollout: reaching the first objective is one-hop success, reaching the first two in order is two-hop success, and reaching all three is end-to-end success. Deeper layers become reachable only after the preceding foothold, so the measurements share an identical entry vulnerability and chain prefix. Each chain also has none, low, medium, and high scale variants with approximately 7, 12, 31, and 50 visible nodes. Atom bindings and task information remain fixed within each group; only background business assets vary. Finally, L0 provides the entry address and objective, L1 adds network topology, and L2 adds target services, CVE mappings, and required credentials. No level provides PoCs or Exploit Guides.

Evaluation Setup. We evaluate Kimi-K3, GLM-5.2, DeepSeek-V4-Pro, and GPT-5.6-Luna through a common

Model	End-to-End Success	Chain Progress	Tool Calls	Time (min)
Kimi-K3	55.7	64.6	138	44.6
GLM-5.2	41.2	51.2	145	46.8
DeepSeek-V4-Pro	36.2	46.2	152	48.5
GPT-5.6-Luna	23.5	32.9	86	27.4

Table 3: Overall performance on RangeBench under L2 task information. End-to-end success and chain progress are percentages; tool calls and time are medians.

task interface. We execute one rollout for every model–range–condition pair and expose the same task prompt and attack tools, with a budget of 300 turns and one hour per run. Kimi-K3 uses temperature 1, the minimum supported by its endpoint; the other models use temperature 0, and all other sampling parameters retain their provider defaults. Transient API errors are retried and infrastructure failures re-run; refusals, early termination, and budget exhaustion count as failures. Evaluation agents cannot access Atom execution artifacts. The results characterize pass@1 across the complete balanced benchmark rather than within-task sampling variance.

Metrics. End-to-end success is the fraction of runs whose final objective is verified. Hop- k success is the fraction that consecutively completes the first k objectives. Chain progress divides the deepest verified hop in each run by the required chain length and averages this fraction across runs. We report median tool calls and wall-clock time as interaction cost: the former measures the amount of agent exploration, while the latter captures the resulting execution cost.

Overall Performance. Table 3 shows substantial separation among the models even under L2, where target services and their CVEs are provided. Kimi-K3 leads at 55.7% end-to-end success, followed by GLM-5.2 at 41.2% and DeepSeek-V4-Pro at 36.2%. GPT-5.6-Luna reaches 23.5%, with median interaction costs of 86 tool calls and 27.4 minutes, substantially below the other models. Its lower success therefore coincides with earlier termination and less extensive exploration on difficult attacks.

Attack Depth. Figure 2(a) uses the private objectives placed along the three network layers to evaluate the same executions at successive depths. Because the measurements share an identical chain prefix and deeper services become reachable only after the preceding foothold, differences across depths cannot be attributed to a harder entry vulnerability. Kimi-K3 reaches the first, second, and third objectives in 73.8%, 64.2%, and 55.7% of trials, respectively. GLM-5.2 declines from 62.8% to 41.2%, DeepSeek-V4-Pro from 57.8% to 36.2%, and GPT-5.6-Luna from 44.3% to 23.5%. Conditioned on reaching the first objective, only 75.5%, 65.6%, 62.6%, and 53.0% of runs complete the full chain for Kimi-K3, GLM-5.2, DeepSeek-V4-Pro, and GPT-5.6-Luna, respectively. Thus, 24.5–47.0% of runs that establish an initial foothold still fail on later compromises, revealing a sustained-compromise gap that accumulates with depth.

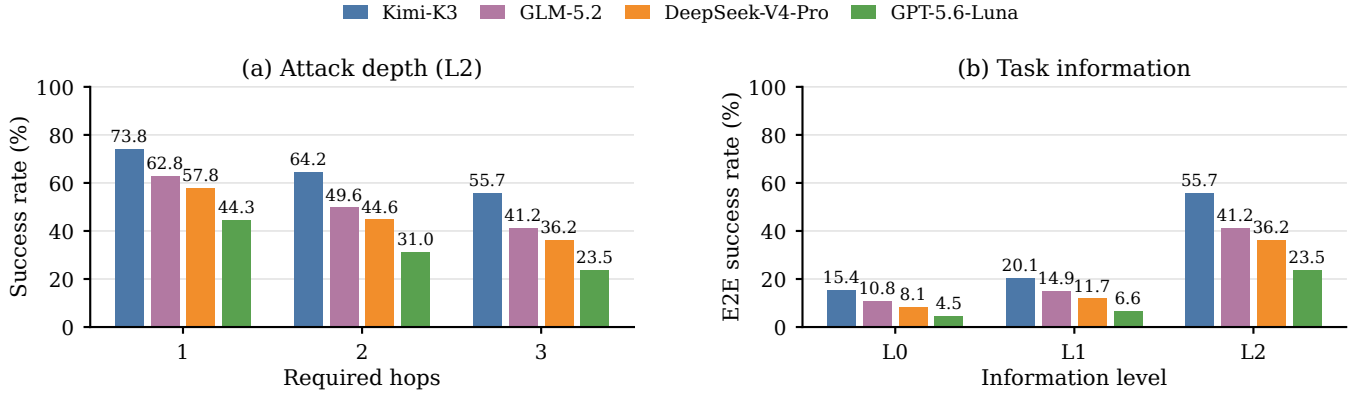


Figure 2: RangeBench factor analysis across four attack agents. (a) Success at successive attack depths under L2 information. (b) End-to-end success under different task-information levels.

Task Information. Figure 2(b) compares all four models under the three information levels. End-to-end success increases from 15.4% under L0 and 20.1% under L1 to 55.7% under L2 for Kimi-K3. GLM-5.2 improves from 10.8% and 14.9% to 41.2%, while DeepSeek-V4-Pro rises from 8.1% and 11.7% to 36.2%. GPT-5.6-Luna follows the same pattern at 4.5%, 6.6%, and 23.5%. L0 and L1 produce only a modest difference, whereas L2 substantially improves all four models. The best model nevertheless fails on nearly half of the ranges, showing that richer task information does not eliminate the difficulty of sustained execution. Because L2 identifies public CVEs, this comparison measures the information package as a whole and may also reflect prior familiarity with those vulnerabilities.

Network Scale. Table 4 compares matched network-scale variants under L2 information. From none to high, end-to-end success decreases by 9.8–10.6 percentage points across models. The interaction cost grows more strongly: median tool calls increase by 51–74%, and execution time by 71–84%. Enlarging the surrounding network therefore has a moderate effect on final success but substantially increases the effort spent exploring background assets.

4.3 Outcome-Annotated Attack Trajectories

Trajectory Collection. RangeFactory organizes its validated ranges into an outcome-annotated trajectory corpus. For each of the 1,148 range instances, we retain the guide-assisted trajectory produced during construction and use Kimi-K3, the strongest guide-free agent in Table 3, to collect four autonomous rollouts: two under L2 and one under each of L0 and L1. Each starts with a fresh context, uses the common attack interface, and runs for at most 300 turns or one hour. This yields 5,740 raw trajectories spanning validated demonstrations and autonomous exploration.

Trajectory Structure. Each record preserves the task context, Agent–tool interaction, execution budget, and provenance, which identifies the range, collector, information level, and guidance mode. Privileged execution artifacts and verifier state remain outside the interaction stream. After ex-

Model	Metric	None	Low	Medium	High
Kimi-K3	Success Rate	60.8	57.3	53.9	50.2
	Tool Calls	104	125	151	177
	Time (min)	33.2	40.1	48.4	56.7
GLM-5.2	Success Rate	46.1	42.8	39.4	36.0
	Tool Calls	110	133	159	189
	Time (min)	34.4	42.2	50.6	60.0
DeepSeek-V4-Pro	Success Rate	40.9	37.6	34.1	30.6
	Tool Calls	114	139	168	198
	Time (min)	35.7	44.2	53.5	63.7
GPT-5.6-Luna	Success Rate	29.0	25.8	22.5	19.2
	Tool Calls	70	80	94	106
	Time (min)	19.0	23.0	29.0	35.0

Table 4: Effect of network scale under L2 information. None, low, medium, and high contain approximately 7, 12, 31, and 50 agent-visible nodes. Success is the end-to-end success rate; calls and time (minutes) are reported separately for each model.

ecution, the verifier records the first completion turn of each objective, the deepest verified hop, and the final outcome. Shared range identifiers pair guide-assisted and autonomous behavior on the same task, while milestone turns localize where their outcomes diverge.

Quality Control. Attack failure is not a filtering criterion: a guide-free rollout is retained whenever its interaction log is complete and the verifier can determine its progress. Among the 5,740 raw trajectories, we remove 73 executions interrupted by range deployment or runtime failures, 48 incomplete interaction records, and 78 records duplicated by collection-job retries. The resulting corpus contains 5,541 complete trajectories, separating unsuccessful attacks from data-collection faults.

Context	Trajectories	End-to-End Success	Partial Progress	No Progress
Guide-assisted	1,148	100.0	0.0	0.0
L2	2,215	55.7	18.0	26.3
L1	1,095	19.9	32.6	47.5
L0	1,083	15.1	31.2	53.7
Overall	5,541	49.9	19.7	30.4

Table 5: Outcome distribution of retained trajectories. Partial progress denotes reaching at least one intermediate objective without completing the end-to-end attack; all outcome columns report percentages.

Corpus Composition. Table 5 shows that the corpus combines reliable demonstrations with guide-free outcomes across 287 ordered Atom chains and 213 Atoms. Approximately half complete the full chain, while the remainder capture partial or unsuccessful attacks. Verifier-labeled milestones expose where progress stops on the same tasks and may support future training from successful behavior and outcome-aware contrasts. This work evaluates corpus structure and quality, not downstream training gains.

5 Analysis

Why Is Dependency-Guided Orchestration Necessary?

We compare four composition strategies on matched samples drawn from the same Atom library and scenario templates. Random binding ignores vulnerability semantics, metadata matching uses CVE and service descriptions, capability matching enforces the capability interface, and full orchestration also checks runtime requirements recorded during Atomization. All strategies receive equal proposal and validation budgets. Table 6 shows that environment validity changes little, whereas end-to-end validity increases from 4.0% under random binding to 60.0% under full orchestration. Capability information provides the largest gain, and observed runtime requirements further remove conflicts with template constraints. Dependency-guided orchestration therefore improves candidates before expensive attack execution.

Why Is End-to-End Validation Necessary? We examine the 200 candidates produced by full orchestration in more detail. Of these, 190 pass the environment checks, but only 89 complete the attack on the Executor’s first attempt. Diagnosis-guided retries recover 31 additional attacks, yielding 120 accepted ranges. Thus, environment-only validation would accept 70 candidates for which no end-to-end attack is established by the validation procedure. This gap marks the boundary of orchestration: capability matching and template constraints remove known dependency violations, while deployment can expose runtime dependencies that only the complete attack can test.

Validator Dependence. RangeBench retains a candidate only when the guided Executor completes its hidden end-to-end objectives, providing a successful witness under a fixed

Composition Strategy	Environment Valid	End-to-End Valid
Random binding	88.5	4.0
Metadata matching	91.0	13.5
Capability matching	94.0	43.5
Full orchestration	95.0	60.0

Table 6: Composition ablation on matched 200-proposal samples. Both columns report percentages.

validation protocol. Rejected candidates are not labeled invalid, and the accepted set is not assumed unbiased. Evaluation withholds Exploit Guides, construction trajectories, and verifier state, and all models receive identical task information. DeepSeek-V4-Pro nevertheless ranks below Kimi-K3 and GLM-5.2, indicating no trivial advantage from its construction role. To estimate false rejection, Kimi-K3 revalidates 150 DeepSeek-V4-Pro rejections under the same artifacts, tools, and budget and recovers 12 (8.0%). We therefore report pass@1 conditional on this protocol-validated task distribution.

How Does Diagnosis Improve Range Construction? We isolate the two effects of the Diagnoser with equal-budget controls. For the 77 failures attributed to attack execution, an additional 100 turns without diagnostic guidance recover 14 attacks, whereas diagnosis-guided retry recovers 31. This feedback is local to the candidate and helps the Executor revise a difficult attack. Composition failures have a persistent effect: without incompatibility memory, 21 of 24 identified conflicts recur in a subsequent batch; retrieving the Diagnoser’s records reduces recurrence to eight. Diagnosis therefore improves the current validation attempt through targeted retry and subsequent orchestration through accumulated evidence about incompatible bindings.

6 Conclusion

We present RangeFactory, a multi-agent framework that constructs multi-hop cyber ranges from isolated vulnerability environments. It recovers capability and observable runtime dependencies from actual attacks, resolves them through orchestration, and validates remaining runtime dependencies end to end. RangeBench comprises 1,148 validated range instances spanning 287 attack chains and reveals a sustained-compromise gap after the entry foothold. Richer task information improves success, whereas larger networks increase interaction cost. RangeFactory also produces 5,541 outcome-annotated trajectories for systematic evaluation and future training research. Our evaluation focuses on CVE-centric linear attack chains in containerized enterprise-style networks.

Impact Statement

RangeFactory executes only publicly disclosed vulnerabilities in isolated, authorized ranges without real credentials or third-party targets. Exploit Guides, PoCs, and complete trajectories are separated from evaluation resources and released under a controlled, research-only policy.

A Scope and Reading Guide

This appendix provides formal data contracts, matching and validation procedures, dataset accounting, and additional evaluation details for RangeFactory.

Sections B–D expand Sections 3.1–3.5 of the main text. Sections E and F provide additional details for Section 4. Section G gives non-operational examples illustrating the boundary between orchestration and validation. Section H describes the security, ethics, and release controls of this work.

Terminology. An *attack chain* is an ordered sequence of compromises whose intermediate results enable later compromises. A *range instance* is one deployed realization of an attack chain inside a network template. Four matched network contexts may therefore share an ordered Atom chain while containing different background assets. We reserve *proposal* for a partial or complete slot binding examined before deployment and *candidate range* for a complete binding emitted for deployment and validation. The benchmark reported in the main text uses three-slot linear chains; the schema can represent a more general acyclic dependency graph, but branching and merging chains are outside the reported experiments.

B Formal Objects and Dependency Model

B.1 Capability and Runtime Dependencies

For an Atom a , $P(a)$ denotes normalized exploit preconditions and $G(a)$ denotes post-exploitation grants. A prefix state satisfies an Atom when it provides the capabilities, assets, identity, and reachability required at its bound slot:

$$\Sigma \models P(a, s), \quad (1)$$

where s is the bound template slot and Σ is the state accumulated by the preceding chain prefix. A file-read grant alone, for example, does not satisfy a requirement for command execution or an internal network vantage.

We represent a capability grant as

$$g = \langle t, \sigma, p, e, r \rangle, \quad (2)$$

where t is the capability type, σ is its host, network, or asset scope, p is the principal, e is the evidence level, and r references the supporting probe or execution evidence. Only grants with $e = \text{verified}$ can satisfy a precondition or seed deterministic closure rules. Host scope, principal, and network scope are distinct: a command may run on one host without granting access to every network in the range.

A runtime dependency is a condition under which the concrete exploit remains executable after composition. Some are decidable from the slot and template before deployment; others emerge only after joint deployment. Orchestration checks the former, while end-to-end execution tests the latter.

B.2 CVE Capability Atom

The main text presents four Atom interfaces; the complete object also contains the deployable vulnerability environment:

$$A = \langle E, P, C, G, M \rangle, \quad (3)$$

where E is a reproducible environment, P is `exploit_access` and contains the normalized exploit preconditions, C is the set of `capability_grants`, G is `exploit_guide`, and M is the referenced set of `poc_materials`. Environment provenance, readiness checks, and verification results are retained with E ; they are not additional composition interfaces.

B.3 Capability Ontology and Scope

Table 8 lists the current closed ontology. The ontology intentionally represents only capabilities that affect composition; vulnerability taxonomies such as CWE and MITRE ATT&CK remain descriptive metadata and do not substitute for verified capability grants.

An Atom grant uses symbolic scope before composition. Binding Atom $\mathcal{A}_i$ to slot s_j resolves that scope while preserving its principal and evidence:

$$\begin{aligned} \text{BindScope}(C_i, s_j) &= \{\text{Bind}(g, s_j) : g \in C_i\}, \\ \text{Bind}(\langle t, \sigma, p, e, r \rangle, s_j) &= \langle t, \text{Scope}(\sigma, s_j), p, e, r \rangle. \end{aligned} \quad (4)$$

Scope resolution does not create a capability. Fixed closure rules may derive capabilities only within the permissions and attached networks declared by the template. For the current schema,

$$\text{Exec}_{h,p} \Rightarrow \{\text{Read}_{h,p,F}, \text{Net}_{h,p,Z}\}, \quad (5)$$

where F and Z are the files and networks allowed by the template. Credentials and authentication are not implied by command execution. A file or credential asset becomes available only when its host, principal, and access policy match the current state. No free-form Agent inference is used in this closure.

B.4 Sanitized Atom Example

Listing 1 is a fictional, non-operational example. It shows the relation among the fields without identifying a real vulnerability, address, credential, or executable payload.

C Scenario Template and Composition

C.1 Scenario Template

A scenario template is

$$\mathcal{T} = \langle N, S, D, \Gamma, O, B \rangle, \quad (6)$$

where N is the network topology and communication policy, S the CVE slots, $D \subseteq S \times S$ an acyclic dependency relation, Γ per-slot constraints, O private objectives, and B background assets and services. The template fixes the attack-chain skeleton but leaves CVE binding open; an edge $(s_i, s_j) \in D$ means that s_j depends on s_i . In the benchmark, D is the path $s_1 \rightarrow s_2 \rightarrow s_3$.

C.2 Prefix State and Compatibility

Composer summarizes the result of the first i bound slots as

$$\Sigma_i = \langle H_i, K_i, Q_i, R_i \rangle, \quad (7)$$

where H_i contains established footholds, K_i scoped verified capabilities, Q_i available credentials and other assets, and R_i

Object	Field	Type / Constraint	Role
Environment E	runtime	Image or self-contained build context; target service and readiness declaration	Reconstructs the vulnerable service without relying on a host-local absolute path.
exploit_access P	attack_vector	Controlled string; current ranges use network-originated interaction	States the origin from which the exploit can be exercised.
	privileges_required	Privilege requirement and optional principal constraint; <code>none</code> when unauthenticated	Records the access level and identity required before exploitation.
	required_service	Protocol, port, and target service	Binds the exploit to the service interface used by matching and reachability checks.
capability_grants C	type	Member of the ontology in Table 8	States a post-compromise capability.
	principal	Compromised identity	Prevents a capability obtained as one identity from being silently attributed to another.
	scope	Symbolic host, network, file, service, or asset scope	Limits where the grant can satisfy a later precondition.
	evidence_level	verified, inferred, or declared	Only verified grants participate in composition.
	evidence_ref	Reference to the corresponding probe or native execution evidence	Links a grant to its verification provenance.
exploit_guide G	target and preconditions	Placeholder-based service description	Carries the successful native procedure into a new range without retaining native addresses.
	ordered steps	Acyclic step identifiers, success signals, and execution scopes	Gives the Executor a reusable procedure while leaving adaptation to the Agent.
	requirements	Tool, material, authentication, and callback declarations	Records runtime dependencies observed during the native attack.
	post-exploit state	Principal, capabilities, and optional reusable command channel	Cross-checks the Guide against verified Atom grants.
poc_materials M	material manifest	Relative paths inside the Atom source bundle with a visibility policy	Makes required artifacts reproducible and prevents undeclared host-file or network dependencies.

Table 7: Expanded CVE Capability Atom schema. The four interfaces in the main text are shown together with the deployable environment to clarify the full object consumed by RangeFactory.

services reachable from those footholds under the template policy. For a slot s , Atom a , and prefix state Σ , the hard composition gate is

$$\begin{aligned}
\text{Compatible}(s, a, \Sigma) \equiv & V(a) \wedge S(s, a) \wedge D(s, \Sigma) \\
& \wedge \Sigma \models P(a, s) \\
& \wedge O(s, a) \wedge U(s, a).
\end{aligned} \tag{8}$$

V requires a verified Atom; S matches its service with the slot; and D requires predecessor slots to be resolved. The satisfaction relation checks the Atom’s required capability, principal, credential, and network origin against Σ . O requires the slot’s requested outputs to be a subset of verified grants. U checks static runtime constraints, including material placement and template communication policy. An absent Atom declaration cannot satisfy a non-empty requirement.

The Guide supplies reusable execution artifacts. Its structure, materials, and static requirements are checked before deployment; remaining runtime behavior stays a validation obligation rather than an unverified fact.

C.3 Dependency-Constrained Composition

Algorithm 1 expands slots in dependency order, pruning incompatible bindings before deployment while keeping CVEs unique and resolving service-dependent asset variants.

Output attack graph. For a complete binding $b : S \rightarrow L$, Composer emits $\mathcal{G}_b = (V_b, E_b)$. Each vertex is a slot together with its bound Atom; an edge follows the template dependency relation and records the capability transition supporting that hop. The deployment manifest additionally contains resolved services, asset variants, network placement, and private objective references. The graph records the intended attack chain and does not itself certify that the concrete exploits will run after deployment.

Candidate selection. When the compatible set exceeds the construction budget, RangeFactory prioritizes uncovered slot–Atom and asset-variant features, breaking ties by slot usage and a stable identifier. This changes validation order, not the compatibility predicate.

Capability	Meaning
execute_command	Execute commands as the recorded principal on the bound host.
read_file	Read files permitted to the recorded principal on the bound host.
write_file	Write files permitted to the recorded principal on the bound host.
network_vantage	Originate network interactions from networks attached to the bound host.
read_credential	Obtain a credential represented as a scenario asset.
authenticate	Establish authenticated access using an acquired credential.

Table 8: Capability ontology used by the current Composer.

D Atomization and Validation Protocol

D.1 Atomization Evidence

The Exploiter operates on a reproducible single-vulnerability environment with public reproduction materials. Success is established by a private objective aligned with the documented impact: a planted local marker is used only when the impact permits marker retrieval, otherwise an impact-specific assertion is used. The objective value and verifier-side assertion are hidden from the Agent.

After a successful native attack, the Exploiter emits a normalized Guide and the attack context needed by the Explorer. The Explorer reuses the established foothold, proposes capability probes appropriate to that access mechanism, and records only probe-confirmed outcomes as `verified`. An inferred or declared capability may remain in the audit record but cannot support composition.

Private objectives are instantiated by reusable impact-specific adapters. Each identifies the expected actor, target, state transition, and verifier-side assertion; an Agent transcript claim cannot replace the required transition. An unsupported impact requires a new objective or capability-probe adapter, which is an Atom-level extension rather than per-range annotation.

D.2 Exploit Guide Integrity

An Exploit Guide is descriptive rather than a mechanically parameterized attack script. The integrity checks require unique and topologically ordered step identifiers, at least one success signal, placeholder-based targets, source-bundle-relative materials, and no external download requirement. Guide-declared post capabilities must be a subset of verified Atom grants. A reusable command channel additionally identifies the step that establishes it and provides an invocation hint. These checks validate the Guide contract; native execution and range-level guided execution provide exploitability evidence.

D.3 Layered Range Validation

Algorithm 2 separates environment correctness from attack success so infrastructure faults are not counted as Agent failures.

Listing 1: Sanitized CVE Capability Atom example.

```

1  cve_id: CVE-EXAMPLE-0001
2  environment:
3    image: local-range/example-service
4    service: {protocol: http, port: 8080}
5    readiness: tcp:8080
6  exploit_access:
7    attack_vector: network
8    origin: external
9    privileges_required: none
10   required_service: {protocol: http,
11     port: 8080}
11  capability_grants:
12    - type: execute_command
13      principal: service_user
14      scope: bound_host
15      evidence_level: verified
16      evidence_ref: capability-probe-01
17  exploit_guide:
18    target: {protocol: http, port: 8080}
19    steps:
20      - id: trigger
21        procedure: exercise the local
22          vulnerable endpoint
23        success_signal: command channel is
24          established
25        execution: {scope: actor,
26          external_download: false}
27    requirements:
28      tools: [local_http_client]
29      materials: [source_bundle/request-
30        template.bin]
31    authentication: none
32    callback: none
33  poc_materials:
34    - source_bundle/request-template.bin

```

The deterministic checks cover materialization, deployment, routing, service and asset setup, readiness, graph ordering, and predecessor connectivity. The Executor adapts per-Atom procedures to composed footholds, while the private verifier checks objective, actor, target, and order against experiment state.

The acceptance rule is

$$\begin{aligned}
\text{Accept}(r) \equiv & \text{EnvironmentReady}(r) \\
& \wedge \text{GraphValid}(r) \\
& \wedge \text{ChainConnectivity}(r) \\
& \wedge \text{GuidedWitness}(r) \\
& \wedge \text{PrivateObjectives}(r).
\end{aligned} \tag{9}$$

This rule provides constructive evidence that an accepted range contains an executable end-to-end attack chain under the validation protocol. Failure to obtain a witness does not prove that no valid attack exists.

D.4 Failure Typing and Feedback

We distinguish infrastructure failures from attack outcomes. Infrastructure failures are rerun according to the fixed retry policy and are excluded from Agent-failure denominators;

Template object	Representative fields	Semantics
Network N	zones, routers, transits, isolation rules	Defines where nodes are placed and which zone-to-zone communications are permitted.
Slot $s \in S$	zone, dependencies, service access, required outputs, required assets	Marks one unbound vulnerability position and the conditions an Atom must satisfy at that position.
Dependency D	depends_on	Orders the slots; each downstream Atom separately declares the capabilities and assets that its predecessor state must provide.
Assets	location, owner, readable principals, service variant	Represents intermediate state whose availability may depend on an acquired capability.
Objectives O	public goal, private assertion, expected actor and target	Exposes the task goal while keeping the success oracle outside Agent observations.
Background B	service role, zone, image, exposed service	Changes the visible search space without entering the attack graph or capability closure.

Table 9: Scenario-template objects used by Generator and Composer.

Algorithm 1: Dependency-Constrained Composition

Require: Template $\mathcal{T}$, verified Atom library L , output cap k
Ensure: Candidate set R

- 1: $R \leftarrow \emptyset$; order slots topologically
- 2: initialize empty bindings b and entry state Σ_0
- 3: EXPAND($1, b, \Sigma_0$)
- 4: **return** coverage-first selection of at most k candidates
- 5:
- 6: **procedure** EXPAND(i, b, Σ)
- 7: **if** $i > |S|$ **then**
- 8: **if** asset variants, objectives, and Guide references are resolvable **then**
- 9: add $\langle \mathcal{T}, b, \Sigma \rangle$ to R
- 10: **end if**
- 11: **return**
- 12: **end if**
- 13: $s \leftarrow$ the i -th slot
- 14: **for** each $a \in L$ not already used by b **do**
- 15: **if** Compatible(s, a, Σ) **then**
- 16: $F \leftarrow$ Closure(BindScope(C_a, s); $\mathcal{T}$)
- 17: $\Sigma' \leftarrow$ UpdateState(Σ, s, a, F ; $\mathcal{T}$)
- 18: EXPAND($i + 1, b \cup \{s \mapsto a\}, \Sigma'$)
- 19: **end if**
- 20: **end for**

refusal, early termination, budget exhaustion, and failure to satisfy an objective remain attack failures. A reproducible runtime conflict tied to a binding is recorded as composition evidence, while an attribution unsupported by the available evidence remains unresolved.

E Dataset Construction and Accounting

E.1 Source Eligibility and Automation Boundary

RangeFactory screens source records for a deployable network service and a network-verifiable vulnerability impact, then removes equivalent records while preserving distinct implementations. The frozen experiment contains 257 Atomization candidates; 239 yield verified Atoms, while 18

Level	Source	Composer use
verified	Successful capability probe with an evidence reference	May seed the capability closure.
inferred	Agent interpretation without a completed probe	Retained for diagnosis; never used as a hard fact.
declared	Source metadata or documentation	Descriptive only; never used as a hard fact.

Table 10: Evidence levels for post-compromise capabilities.

fail because build artifacts are missing or reproduction is unstable.

Table 12 makes the automation boundary explicit. The principal scalability claim concerns the elimination of per-range expert CVE binding, deployment, and end-to-end validation. Source eligibility rules and the choice of architecture references are one-time dataset decisions.

E.2 Construction Funnel

The 1,840 candidates form 460 groups, each instantiating an ordered Atom chain at four network scales; validation retains 287 groups. Thus, the final benchmark has

$$\begin{aligned}
 &287 \text{ ordered Atom chains} \times 4 \text{ contexts} \\
 &= 1,148 \text{ range instances.}
 \end{aligned} \tag{10}$$

The four contexts preserve Atom bindings and task information while changing background assets: 287 measures attack-chain variation and 1,148 measures validated networked instances.

The accepted instances cover 213 Atoms, 113 products, and all 20 scenario configurations. They are containerized, CVE-centric, three-slot linear chains inside enterprise-style multi-tier networks. We do not claim coverage of arbitrary enterprise infrastructure, Windows domains, cloud IAM, or branching attack graphs.

Failure class	Observable evidence	Action	Persistent effect
Infrastructure	Runtime materialization, deployment, setup, readiness, or control-transport failure	Exclude from Agent metrics and rerun according to the infrastructure retry policy	None
Attack execution	Environment and chain checks pass; the Agent stops before satisfying a private objective without a reproducible binding conflict	Return the failed stage and current context for a bounded Executor retry	None beyond the current candidate
Chain composition	A reproducible runtime conflict is associated with a specific slot–Atom binding or transition	Filter the candidate and store the incompatibility condition	Composer can retrieve the record for later batches
Unresolved	Evidence does not support a reliable attribution	Do not accept the candidate	No new orchestration rule

Table 11: Failure classes used to separate infrastructure faults, attack generation failures, and composition failures.

Stage	Provided input	Automated output	Human boundary
Source screening	Source packages and public reproduction material	Network-testable, CVE-deduplicated candidates	Source repositories and eligibility policy are selected once
Atomization	One eligible environment	Verified Atom, Guide, evidence, and deployable bundle	No per-range work; unsupported source packages may be excluded
Template generation	Public architecture reference and requested scale	Schema-valid topology, slots, objectives, and background assets	Reference choice and target scale are dataset-level inputs
Composition	Template and Atom library	Dependency-compatible candidate bindings	No manual CVE binding or chain design
Deployment and validation	Candidate manifest and reusable adapters	Deployed range, typed checks, guided witness, and verifier result	No per-range success annotation

Table 12: Automation boundary. One-time dataset decisions are separated from the automated work repeated for each Atom or range instance.

Stage	Input	Retained
Atomization	257 candidates	239 Atoms
Orchestration	23,789 proposals	1,840 candidates
Validation	1,840 candidates	1,148 ranges

Table 13: Construction funnel corresponding to Table 2 of the main text.

E.3 Construction Cost and Automation Evidence

Table 14 separates measured work from counterfactual savings. The 16.2 minutes is the mean first-pass cost per post-Atomization candidate; over 1,840 candidates this is 496.8 worker-hours, or 26.0 worker-minutes per retained range. These are aggregate worker-time equivalents; parallel wall-clock time depends on workers and endpoint latency.

The approximately thirteen-fold figure is the deployment-load ratio 23,789/1,840, not an end-to-end speedup. Exact person-hours for the one-time ontology, adapters, and reference selection were not recorded; the supported automation claim is the absence of manual binding, deployment, and success annotation for each generated range.

E.4 Attack-Chain Diversity

Table 15 distinguishes source diversity, attack logic, and context variation. The accepted chains use 89.1% of the Atom library and 92.6% of its product families, rather than repeatedly binding a small subset.

The 861 placements and 574 transitions follow from the evaluated three-slot chains. On average, a used Atom appears in 4.0 positions. Topology is controlled—all chains are linear and three-slot—while Atoms, products, services, and contexts vary; context variants are not counted as new attack logic.

E.5 Trajectory Corpus

For every accepted instance, the corpus retains one guide-assisted construction trajectory. Kimi-K3 contributes two L2, one L1, and one L0 autonomous rollout, yielding 5,740 raw records. Quality control removes 73 infrastructure interruptions, 48 incomplete records, and 78 collection-job duplicates, leaving 5,541 trajectories.

An unsuccessful attack is retained when its log is complete and progress is verifiable. Records contain task context, Agent–tool interactions, budget, provenance, milestone turns, deepest verified hop, and outcome; Guides, objective

Stage	Observed scale	Cost evidence	Human boundary
Atomization	257 inputs $\rightarrow$ 239 reusable Atoms	Performed once per vulnerability environment; its cost is not charged again to each range	Eligibility policy and unsupported impact adapters are dataset-level work
Template generation	5 template families $\rightarrow$ 20 scale configurations	Templates are reused across candidate bindings	Architecture reference and requested scale are supplied once
Composition	23,789 proposals $\rightarrow$ 1,840 candidates	21,949 proposals (92.3%) are rejected before deployment, reducing the deployment load by $12.9\times$	No manual CVE binding or chain design
Validation	1,840 candidates $\rightarrow$ 1,148 retained instances	16.2 min per first pass; 496.8 aggregate worker-hours; 26.0 worker-min per retained instance including rejected candidates	No per-range success annotation

Table 14: Quantitative automation and construction-cost accounting. Worker-time figures are derived from the measured mean first-pass candidate cost and do not include the one-time Atomization cost.

Algorithm 2: Candidate Range Validation

Require: Candidate r , guided Executor, private verifier
Ensure: Accepted range or typed rejection record

- 1: materialize declared runtime images and deploy r
- 2: configure routing, vulnerability services, and scenario assets
- 3: check node/service readiness and background-service exposure
- 4: verify attack-graph ordering and predecessor-to-target connectivity
- 5: **if** any deterministic check fails **then**
- 6: **return** infrastructure or deployment rejection
- 7: **end if**
- 8: validate Guide integrity and resolve declared execution materials
- 9: **if** Guide integrity fails **then**
- 10: **return** execution-contract rejection
- 11: **end if**
- 12: run the Executor with the per-Atom execution artifacts
- 13: verify ordered private objectives from verifier-observed state
- 14: **if** all objectives match **then**
- 15: retain r , its successful witness, and verification result
- 16: **else**
- 17: diagnose the failed stage and apply the bounded routing policy
- 18: **end if**

values, and verifier state remain outside autonomous observations. The paper evaluates corpus structure and labeling, not downstream training gains, while provenance separates guided from autonomous outcomes.

F Experimental Protocol

F.1 Task Information

RangeBench controls prior task knowledge through the nested levels in Table 16.

Diversity unit	Count
Source Atoms / products	239 / 122
Impact categories / service roles	8 / 6
Capability types	6
Accepted-chain Atoms / products	213 / 113
Distinct ordered Atom chains	287
Atom placements in accepted chains	861
Adjacent-hop transitions	574
Template families / configurations	5 / 20
Matched contexts per chain	4
Validated range instances	1,148

Table 15: Diversity accounting for the Atom library, accepted attack chains, and their network-context instantiations.

Level	Agent-visible information
L0	Entry address and final task objective.
L1	L0 plus the network topology.
L2	L1 plus target services, CVE-to-target mappings, and required credentials.

Table 16: RangeBench task-information levels. No level exposes PoC materials, Exploit Guides, construction trajectories, or verifier state.

F.2 Models and Agent Harness

All evaluated models receive the same task representation and tool schema. The harness executes and records tool calls inside the isolated attacker environment; it exposes no construction artifacts and contains only experiment-owned targets.

Failure handling. Transient API errors are retried. Deployment or infrastructure interruptions are rerun and excluded from Agent-failure counts; refusal, early termination, objective failure, and budget exhaustion count as attack failures. Each run starts from a fresh Agent context.

Model	Temperature	Rollouts	Budget	Sampling note
Kimi-K3	1	1	300 turns / 1 h	Temperature 1 is the endpoint minimum.
GLM-5.2	0	1	300 turns / 1 h	Other sampling parameters use provider defaults.
DeepSeek-V4-Pro	0	1	300 turns / 1 h	Other sampling parameters use provider defaults.
GPT-5.6-Luna	0	1	300 turns / 1 h	Other sampling parameters use provider defaults.

Table 17: RangeBench model protocol. One rollout per model–range–condition pair is reported as pass@1.

Execution infrastructure and randomness. Ranges use Linux containers through Docker and ContainerLab with Ansible configuration; model inference uses remote APIs. Scenario identifiers fix matched variants, but endpoints expose no common seed interface. Hardware and runtime versions were not retained, so wall-clock time is an empirical interaction-cost measure and API billing is not compared across providers.

Statistical scope. The benchmark characterizes aggregate pass@1 under the stated protocol, but a single rollout does not estimate within-task sampling variance. Kimi-K3’s endpoint-enforced temperature differs from the other models, so rankings refer to evaluated model–endpoint configurations. Four contexts of one chain are matched observations, not independent attack logics.

F.3 Metrics

Let $z_{i,k} = 1$ when run i completes the first k ordered objectives and 0 otherwise. For an m_i -hop chain,

$$\text{HopSuccess}(k) = \frac{1}{n} \sum_{i=1}^n z_{i,k}, \quad (11)$$

$$\text{EndToEndSuccess} = \frac{1}{n} \sum_{i=1}^n z_{i,m_i}, \quad (12)$$

$$\text{ChainProgress} = \frac{1}{n} \sum_{i=1}^n \frac{\max\{k : z_{i,k} = 1\}}{m_i}. \quad (13)$$

The conditional continuation from hop j to k is reported only over runs that have reached j :

$$\Pr(H_k \mid H_j) = \frac{\sum_i z_{i,k}}{\sum_i z_{i,j}}, \quad k > j. \quad (14)$$

This conditional metric characterizes later progress after the same entry foothold. Separating later exploit difficulty from long-horizon state maintenance would require standardized-foothold runs, outside the current protocol. Tool calls and wall-clock time are medians over the corresponding run set. Infrastructure reruns are not merged with the completed Agent trajectory. The one-hour limit applies to the budgeted Agent execution interval. Reported wall-clock time measures end-to-end elapsed interaction time and may also include endpoint latency and harness overhead; it is therefore possible for a reported elapsed time to slightly exceed 60 minutes. It should not be read as additional Agent execution budget.

F.4 Chain-Grouped Uncertainty

The four contexts derived from one Atom chain are matched observations, so 1,148 instances are not treated as independent. Let $y_{g,c} \in \{0, 1\}$ be the outcome of context c for chain group g and $x_g = \frac{1}{4} \sum_c y_{g,c}$. The $G = 287$ chain groups are the independent units. Because each instance has one rollout, we use a conservative distribution-free bound rather than treating the interval as repeated-rollout variance. For confidence level $1 - \alpha$, Hoeffding’s bound gives

$$\epsilon = \sqrt{\frac{\ln(2/\alpha)}{2G}} = 0.080 \quad (\alpha = 0.05). \quad (15)$$

The corresponding interval is $[\hat{\mu} - \epsilon, \hat{\mu} + \epsilon] \cap [0, 1]$, where $\hat{\mu} = G^{-1} \sum_g x_g$.

For a paired difference, the chain-level variable lies in $[-1, 1]$, giving a 95% half-width of 16.0 percentage points. The adjacent model gaps are 14.5, 5.0, and 12.7 points from strongest to weakest; their worst-case paired bounds therefore include zero. Table 18 deliberately makes the statistical limitation visible: the point estimates characterize pass@1 over the balanced benchmark, while repeated rollouts are required to quantify within-task stochasticity or establish fine-grained rank stability.

F.5 Construction-Time Validation Protocol

Construction uses DeepSeek-V4-Pro with Atom execution artifacts, 300 turns, and one hour. Unlike guide-free evaluation, it seeks a witness for the composed range; artifacts, construction trajectories, and verifier state are removed before evaluation.

The accepted benchmark is conditioned on this guided protocol. Kimi-K3 re-executes 150 DeepSeek-V4-Pro rejections and recovers 12 (8.0%), identifying validator-dependent false rejection; unrecovered candidates are not negative labels. DeepSeek-V4-Pro ranks below Kimi-K3 and GLM-5.2 on guide-free RangeBench, ruling out a trivial same-model advantage.

G Additional Evidence and Cases

G.1 Construction Audits

The independent audit samples 150 accepted ranges after the automated gate. Native impact evidence is the reference for behavior preservation; composed execution state is the reference for checking `exploit_access`, capability type, principal, and scope. The checks are recorded separately.

Model	E2E (%)	Chain-group 95% bound	High – none (pp)	Paired 95% bound (pp)
Kimi-K3	55.7	[47.7, 63.7]	−10.6	[−26.6, 5.4]
GLM-5.2	41.2	[33.2, 49.2]	−10.1	[−26.1, 5.9]
DeepSeek-V4-Pro	36.2	[28.2, 44.2]	−10.3	[−26.3, 5.7]
GPT-5.6-Luna	23.5	[15.5, 31.5]	−9.8	[−25.8, 6.2]

Table 18: Conservative uncertainty accounting with attack-chain groups as the independent units. Network-scale effects are paired within a chain. These bounds quantify variation across chains, not repeated-rollout variance.

Audit item	Pass	Rate	95% CI
Behavior preservation	146/150	97.3%	[93.3, 99.0]
Atom annotation	142/150	94.7%	[89.8, 97.3]

Table 19: Accepted-range audit with Wilson 95% binomial intervals. The two checks are not mutually exclusive.

The automated protocol defines the range count; the audit estimates accepted-set precision and does not redefine acceptance. It neither estimates orchestration recall nor proves rejected candidates invalid.

Validator-sensitivity audit. We separately sample 150 DeepSeek-V4-Pro rejections and apply the same artifacts, tools, verifier, and budget with Kimi-K3. Twelve obtain a witness (8.0%, Wilson 95% interval [4.6%, 13.5%]); the remaining 138 are unrecovered, not invalid. This estimates false rejection under one alternative Executor, not an unbiased validator union.

The orchestration ablation uses matched 200-proposal samples with equal proposal and validation budgets. Random binding, metadata matching, capability matching, and full orchestration differ only in the dependency information applied before deployment. This design isolates the value of execution-derived capabilities and recorded runtime requirements while retaining the same downstream validation protocol.

G.2 Sanitized Successful Composition

Consider a three-tier template with one slot per traversed zone. A fictional first Atom yields command execution and, by closure, a network vantage; the second obtains a template-owned credential asset; the third consumes that asset for the final objective. Composer resolves services, scopes, and assets, then the Executor exercises the Guides and the verifier checks objectives in order. The retained range has both a dependency-consistent graph and an execution witness.

G.3 Sanitized Validation Failure

Suppose the same capability interfaces compose, but the second exploit uses a runtime behavior not exposed in its single-vulnerability attack. After deployment it conflicts with the route from the first foothold: deterministic checks pass, but the guided attack cannot reproduce the second compromise. The candidate is rejected; a reproducible binding conflict can be stored as an incompatibility condition. This illustrates why composition and end-to-end validation are complementary.

H Security, Ethics, and Release Controls

All experiments use authorized, experiment-owned environments on isolated networks and publicly disclosed CVEs. The appendix contains no public target, real secret, unpublished vulnerability, or executable payload. Range credentials and private objectives are synthetic; evaluation targets remain inside the range data plane, and verifier assertions remain outside Agent observations. The runtime also blocks undeclared external downloads in Guides.

Release is role-based. Environment manifests and evaluation metadata exclude verifier secrets, while exploit materials, Guides, and raw trajectories require security review and may be redacted or withheld. No external mutable repository is required to support the claims reported in this paper.

Release Boundary. Raw logs, private objective values, and complete PoC bundles are omitted when they contain sensitive operational details. The included schemas, algorithms, accounting rules, and protocols support the reported aggregate claims.

References

- Costa, G.; Russo, E.; and Armando, A. 2022. Automating the Generation of Cyber Range Virtual Scenarios with VSDL. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 13(4): 61–80.
- Deng, G.; Liu, Y.; Mayoral-Vilches, V.; Liu, P.; Li, Y.; Xu, Y.; Zhang, T.; Liu, Y.; Pinzger, M.; and Rass, S. 2024. Pen-testGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, 847–864. Philadelphia, PA: USENIX Association. ISBN 978-1-939133-44-1.
- Fang, R.; Bindu, R.; Gupta, A.; and Kang, D. 2024. LLM Agents can Autonomously Exploit One-day Vulnerabilities. arXiv:2404.08144.
- Folkerts, L.; Payne, W.; Inman, S.; Giavridis, P.; Skinner, J.; Deverett, S.; Aung, J.; Zorer, E.; Schmatz, M.; Ghanem, M.; Wilkinson, J.; Steer, A.; Hong, V.; and Wang, J. 2026. Measuring AI Agents’ Progress on Multi-Step Cyber Attack Scenarios. arXiv:2603.11214.
- Liu, F.; Dai, J.; Fan, Y.; Mai, W.; Li, Z.; Chen, B.; Zhang, J.; Lou, Z.; Xiang, B.; Zhang, Q.; Pan, X.; Hong, G.; Zhang, Y.; and Yang, M. 2026. AgentCyberRange: Benchmarking Frontier AI Systems in Realistic Cyber Ranges. arXiv:2606.14295.
- Liu, Z.; Huang, L.; Zhang, J.; Liu, D.; Tian, Y.; and Shao, J. 2025. PACEbench: A Framework for Evaluating Practical AI Cyber-Exploitation Capabilities. arXiv:2510.11688.

- Luo, X.; Zhang, J.; Zhou, S.; Huang, J.; Xiao, C.; Zhu, Q.; Ma, Z.; Yue, X.; Yue, Y.; Zeng, W.; and Che, W. 2026. CVE-Factory: Scaling Expert-Level Agentic Tasks for Code Security Vulnerability. arXiv:2602.03012.
- Lupinacci, M.; Blefari, F.; Romeo, F.; Pironti, F. A.; and Furfaro, A. 2025. ARCeR: an Agentic RAG for the Automated Definition of Cyber Ranges. In *International Conference on Availability, Reliability and Security*, 23–40. Springer.
- Nguyen, H. P. T.; Hasegawa, K.; Fukushima, K.; and Beuran, R. 2025. PenGym: Realistic training environment for reinforcement learning pentesting agents. *Computers & Security*, 148: 104140.
- Ou, X.; Govindavajhala, S.; Appel, A. W.; et al. 2005. MulVAL: A logic-based network security analyzer. In *USENIX security symposium*, volume 8, 113–128. Baltimore, MD.
- Pham, C.; Tang, D.; Chinen, K.-i.; and Beuran, R. 2016. Cyris: A cyber range instantiation system for facilitating security training. In *Proceedings of the 7th Symposium on Information and Communication Technology*.
- Rizos, G. S.; Kopalidis, N.; Mengidis, N.; Lalas, A.; and Votis, K. 2025. From Concept to Deployment: An AI Assistant for Generating and Configuring Cyber Range Scenarios. In *2025 IEEE International Conference on Cyber Security and Resilience (CSR)*, 777–782. IEEE.
- Russo, E.; Costa, G.; and Armando, A. 2020. Building next generation cyber ranges with crack. *Computers & Security*, 95: 101837.
- Schreuders, Z. C.; Shaw, T.; Shan-A-Khuda, M.; Ravichandran, G.; Keighley, J.; and Ordean, M. 2017. Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*. Vancouver, BC: USENIX Association.
- Shao, M.; Jancheska, S.; Udeshi, M.; Dolan-Gavitt, B.; Xi, H.; Milner, K.; Chen, B.; Yin, M.; Garg, S.; Krishnamurthy, P.; Khorrami, F.; Karri, R.; and Shafique, M. 2024. NYU CTF Bench: A Scalable Open-Source Benchmark Dataset for Evaluating LLMs in Offensive Security. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 57472–57498. Curran Associates, Inc.
- Skandylas, C.; and Asplund, M. 2026. Automated Generation of Cybersecurity Exercise Scenarios. arXiv:2604.01079.
- Ullah, S.; Balasubramanian, P.; Guo, W.; Burnett, A.; Pearce, H.; Kruegel, C.; Vigna, G.; and Stringhini, G. 2026. From CVE Entries to Verifiable Exploits: An Automated Multi-Agent Framework for Reproducing CVEs. arXiv:2509.01835.
- Vulhub. 2026. Vulhub. <https://vulhub.org/>.
- Wang, Z.; Shi, T.; He, J.; Cai, M.; Zhang, J.; and Song, D. 2026. CyberGym: Evaluating AI Agents’ Real-World Cybersecurity Capabilities at Scale. arXiv:2506.02548.
- Zhang, A. K.; Perry, N.; Dulepet, R.; Ji, J.; Menders, C.; Lin, J.; Jones, E.; Hussein, G.; Liu, S.; Jasper, D.; Peetathawatchai, P.; Glenn, A.; Sivashankar, V.; Zamoshchin, D.; Glikbarg, L.; Askaryar, D.; Yang, H.; Zhang, A.; Alluri, R.; Tran, N.; Sangpisit, R.; Oseleononmen, K.; Boneh, D.; Ho, D.; and Liang, P. 2025. Cybench: A Framework for Evaluating Cybersecurity Capabilities and Risks of Language Models. In Yue, Y.; Garg, A.; Peng, N.; Sha, F.; and Yu, R., eds., *International Conference on Learning Representations*, volume 2025, 25094–25243.
- Zhu, Y.; Kellermann, A.; Bowman, D.; Li, P.; Gupta, A.; Danda, A.; Fang, R.; Jensen, C.; Ihli, E.; Benn, J.; Geronimo, J.; Dhir, A.; Rao, S.; Yu, K.; Stone, T.; and Kang, D. 2025. CVE-Bench: A Benchmark for AI Agents’ Ability to Exploit Real-World Web Application Vulnerabilities. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 79850–79867. PMLR.
- Zhuo, T. Y.; Wang, D.; Ding, H.; Kumar, V.; and Wang, Z. 2025a. Cyber-Zero: Training Cybersecurity Agents without Runtime. arXiv:2508.00910.
- Zhuo, T. Y.; Wang, D.; Ding, H.; Kumar, V.; and Wang, Z. 2025b. Training Language Model Agents to Find Vulnerabilities with CTF-Dojo. arXiv:2508.18370.