SysRepair-Bench : A Benchmark for AI Agents' Ability to Remediate Real-World System Vulnerabilities

Anonymous Author(s)
Affiliation
Address
email

Abstract

Existing agent benchmarks for computer security predominantly evaluate offensive capability or source-level program repair, but do not capture the real world task of system remediation that system administrators or incident responders provide such as remediating misconfigurations, vulnerable packages, and unsafe permissions while preserving service availability. We introduce SysRepair-Bench , a reproducible benchmark comprising 313 Dockerized and virtual machine scenarios across six suites. Each container scenario is scored on two mandatory objectives **(i)** the vulnerability is no longer exploitable, and **(ii)** all previously operational services remain functional. **(iii)** An additional objective, enforced in 39 scenarios where direct source patching is forbidden, evaluates the agent's ability to implement compensating controls. This requires the agent to reason about alternative mitigations, such as network isolation or privilege restriction, to neutralize the exploit path. We contribute two new artifacts: Meta4, a 137-scenario suite covering modern Common Vulnerabilities and Exposures including the Copy Fail, Log4Shell family, Spring4Shell, PwnKit, Dirty Pipe, GameOver(lay), regreSSHion, Leaky Vessels, the XZ backdoor, container-runtime escapes, and cloud-on-localhost misconfigurations and Hivestorm, an unguided, system-wide remediation partial-credit track with per-build identity randomization. We report baselines with MiniMax-M2.7, Qwen3.5-35B-A3B, and Qwen3.5-9B paired with ReAct agent solvers in two variants: black-box (no prior vulnerability context) and report-informed (provided with CVE details). The strongest configuration remediates approximately **60%** of all 313 scenarios end-to-end at $\mathrm{success@5}$. We observe that failures are concentrated where the agent fails to discover the threat, applies an ineffective fix, breaks protected service availability during remediation, or encounters compensating-control requirements. We release the scenarios, scan reports, grading harness, and evaluation code under an open license at `https://anonymous.4open.science/r/sysrepair-bench-425F/`

1 Introduction

Modern computing infrastructure accumulates security debt continuously through vulnerable packages, unsafe permissions, exposed services, insecure defaults, and configuration drift. Paying down this debt is the daily work of system administrators and incident responders. They tighten a configuration directive, apply a patch, scope a bind address, remove a backdoor package, or revoke an unsafe privilege. This work is performed on live systems, must preserve service availability, and is graded by the absence of an exploit rather than by a passing unit test.

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

Prior agent benchmarks address adjacent capabilities. Program-repair benchmarks such as SWE-bench [6] evaluate source-level edits against a project's test suite, and offensive security benchmarks such as Cybench [26], NYU CTF Bench [18], and CVE-Bench [28] evaluate exploitation capability against vulnerable targets. To our knowledge, none evaluates the defensive task that organizations now deploy agents to perform: identify a threat, reason about a running service, select a system-administration primitive, execute it on a live host, and verify that the vulnerability is closed without disrupting the service it protects.

We introduce SysRepair-Bench , a benchmark that evaluates a distinct capability regime: **infrastructure remediation under operational constraints**. The benchmark jointly exercises four reasoning skills that together define this regime: *threat discovery* from a scan report or running service, *configuration reasoning* across heterogeneous service stacks, *dependency management* for vendored and source-built components, and *compensating-control selection* when direct patching is forbidden. Its central methodological commitment is that **a remediation which closes a vulnerability by breaking the protected service is scored as a failure, not a partial success**. This dual constraint mirrors the operational reality of incident response, and it is what most clearly distinguishes SysRepair-Bench from offensive benchmarks, which reward only the exploit, and from program-repair benchmarks, which require no live service.

Scenarios are drawn from six suites that together span two decades of the operational threat landscape. We deliberately diversify the underlying vulnerability classes, service families, and required remediation primitives so that no single repair template (package upgrade, service restart, configuration edit, firewall rule, account change) can dominate the benchmark. This forces agents to select the appropriate primitive per scenario rather than overfit to one operational habit. Two of the six suites also address known weaknesses of prior corpora: `Meta4` contributes 137 modern Common Vulnerabilities and Exposures scenarios that close the temporal gap left by aging corpora, and `Hivestorm` introduces weighted partial credit over per-build randomized artifact identities to mitigate memorization and benchmark contamination effects.

The six suites are:

1. **CCDC**: derived from published blue-team materials on the Collegiate Cyber Defense Competition [10].
2. **Metasploitable 2**: grounded in an OpenVAS scan of the canonical image [14].
3. **VulnHub**: per-virtual-machine rebuilds from fourteen community images [21].
4. **Metasploitable 3**: ports of the Rapid7 upstream covering Drupalgeddon, ProFTPD mod_copy, Docker group escalation, Struts, Jenkins, GlassFish, Tomcat, ElasticSearch, IIS WebDAV, and Server Message Block [15].
5. **Meta4**: modern Common Vulnerabilities and Exposures including the Copy Fail, Log4Shell family, Spring4Shell, PwnKit, Dirty Pipe, GameOver(lay), regreSSHion, Leaky Vessels, the XZ backdoor, intentionally vulnerable web-API targets, and cloud-on-localhost misconfigurations (LocalStack, MinIO, ArgoCD, k3s).
6. **Hivestorm**: an unstructured, full-system track whose planted artifacts (backdoor account, trojan path) are randomized per build and scored with a weighted grader rather than pass or fail [5].

Our contributions are:

1. **297** binary-scored scenarios across six suites with uniform format and harness.
2. A dual-objective scoring protocol extended by compensating-control adequacy.
3. **Meta4**, closing the temporal gap left by aging corpora.
4. **Hivestorm**, with per-build identity randomization.
5. Baseline results for MiniMax-M2.7, Qwen3.5-35B-A3B, and Qwen3.5-9B agents in two variants: black-box (no prior vulnerability context) and report-informed (provided with CVE details).

2 Related work

Program repair. SWE-bench [6] and the broader automated program repair literature [7, 23] evaluate source-level edits against test suites. SysRepair-Bench differs along three axes: the action space is system administration rather than source editing, the artifact under test is a running service rather than a repository, and success is defined by the absence of an exploit plus preservation of service behavior.

Offensive security agent benchmarks. Cybench [26], NYU CTF [18], InterCode [24], and CVE-Bench [28] release problem sets that measure attack capability. CVE-Bench, the closest prior work, collects 40 critical web-application Common Vulnerabilities and Exposures and reports that the strongest team of agents exploits up to 13 percent in the one-day setting. SysRepair-Bench is the defensive counterpart: each of our scenarios is the mirror image of an exploitation target, and our evaluation rewards closing the vulnerability rather than opening it.

Cyber defense competitions. The Collegiate Cyber Defense Competition [11] and Hivestorm [4] are long-running collegiate exercises in which student teams harden intentionally vulnerable systems under red-team pressure. The hardening toolkits and Hivestorm scoring engine directly informed our `ccdc/` and `hivestorm/` suites.

Vulnerable VM corpora. Metasploitable 2 [16], Metasploitable 3 [17], and VulnHub [21] are the canonical intentionally vulnerable images. We decompose them into per-vulnerability scenarios with uniform scoring and extend them with Metasploitable 4 (Meta4).

Partial-credit evaluation and contamination. Binary scoring is standard in program-repair and agent-tool benchmarks, but discards information: an agent that fixes four of five faults scores identically to one that fixes none. Partial-credit scoring has a long history in educational assessment [1] and is increasingly applied to agent evaluation. Randomization of problem identities has been proposed in the contamination-detection literature [8, 12]. Hivestorm combines both: weighted partial credit over randomized identities.

Agent scaffolds. Our harness instantiates ReAct [25], a basic tool-use baseline, Reflexion [19], Plan-and-Solve [22], and Language Agent Tree Search [27], all off-the-shelf. We utilize ReAct as our primary experimental scaffold because its iterative cycle of reasoning and action closely mirrors how a human competitor or penetration tester naturally explores and interacts with a live system. This aligns with contemporary offensive security benchmarks like Cybench [26] and CVE-Bench [28], which rely on ReAct-style agent loops equipped with terminal execution and web search tools to evaluate autonomous exploitation capabilities. In SysRepair-Bench , we extend this established paradigm to the defensive task of system remediation.

3 The SysRepair-Bench benchmark

3.1 Design principles

SysRepair-Bench is designed around six principles. **(1)** *Grounded and representative*: every scenario traces to a documented public artifact (a CCDC hardening script, an OpenVAS scan, a VulnHub write-up, a CVE advisory, or a Hivestorm challenge). Vulnerability selection is anchored to authoritative industry frameworks: classes drawn from the OWASP Top 10 [13] and the MITRE CWE Top 25 [9] for systemic coverage, and high-impact modern CVEs (Copy Fail, Log4Shell, Dirty Pipe) selected from the CISA Known Exploited Vulnerabilities catalog [2] so that the benchmark targets actively exploited threats rather than theoretical flaws. **(2)** *Reproducible*: each scenario builds deterministically from a `Dockerfile` or Vagrantfile. **(3)** *Operationally scoped action space*: fixes are expressible as system-administration primitives (edits, package operations, permission changes, service and firewall management); source code modification is deliberately excluded. **(4)** *Regression as a first-class failure*: a remediation that eliminates the vulnerability by disabling or breaking the protected service fails the scenario outright. Unlike offensive evaluation, where success is typically defined by exploit execution alone, remediation requires simultaneously restoring security while preserving operational functionality. **(5)** *Strategy diversity*: scenarios are deliberately distributed across

Table 1: Suite composition and distributions across the 313 scenarios. Hivestorm is additionally distributed across six operating systems (Debian, Ubuntu, CentOS, Windows Server Core, FreeBSD, Active Directory Domain Controller).

Suite			CVSS v3.1		Remediation category	
Suite	#	Era	Severity	#	Category	#
`ccdc`	50	2015-22	Critical 9.0-10.0	94	Config. Hardening	113
`meta2`	40	2008-12	High 7.0-8.9	106	Dependency & Pkg. Mgmt.	71
`vulnhub`	30	2012-22	Medium 4.0-6.9	44	Access Control	64
`meta3/ubuntu`	19	2014-20	Low 0.1-3.9	1	Compensating Controls	39
`meta3/windows`	21	2016-20	Unscored	68	Network Security	10
`meta4`	137	2018-26			Free-roam (Hivestorm)	16
`hivestorm`	16	HS20-23				
Total	**313**		**Total**	**313**	**Total**	**313**

remediation primitives, including configuration edits, package operations, account and permission changes, firewall rules, service management, and compensating controls, so that no single repair template (for example, `apt upgrade` or `systemctl restart`) can dominate the benchmark. This requires the agents to determine which primitive a scenario requires, rather than apply a previously memorized solution. **(6)** *Complementary partial credit*: a single binary score cannot cleanly express multi-fault scenarios, the Hivestorm track adds a weighted partial-credit dimension over per-build randomized artifact identities, which jointly target the partial-credit signal lost in binary scoring and the memorization shortcut available when artifact names are stable.

3.2 Scenario format and categories

Binary-suite scenarios are directories with three files: `Dockerfile` (builds the vulnerable container), `threat.md` (severity, CVSS v3.1 score, CVE identifier, affected service, and canonical remediation steps), and `verify.sh` (exit 0 if the vulnerability is closed and the service still works, exit 1 otherwise). `verify.sh` is split into a proof-of-concept block, a regression block, and, for compensating-control scenarios, a third attack-surface block. Hivestorm scenarios emit a stream of weighted checks instead of a binary exit code; planted artifacts are randomized per build so that a memorized identity confers no advantage.

Every scenario is labeled with one of five operational remediation categories: *Access Control* (authentication, authorization, privileges, ownership); *Configuration Hardening* (insecure defaults, missing directives, misconfigured parameters); *Dependency and Package Management* (outdated or compromised packages, unnecessary high-risk daemons); *Network Security and Firewall Policy* (exposed ports, missing rules, unrestricted listeners); and *Compensating Controls*, in which direct remediation is forbidden by scenario constraints (legacy dependency, end-of-life software, no maintenance window) and the agent must instead apply network or application-layer mitigations. Scoring adds a third mandatory objective for this last category.

3.3 Composition and distributions

Table 1 summarizes the benchmark across three axes. Two features are worth highlighting. First, Critical and High scenarios together constitute 200 of 313 (64%), reflecting operational triage priorities. Second, Compensating Controls (39, 12%) is intentionally larger than Network Security (10, 3%) because it is the harder and more under-studied evaluation target. A full service-and-application-type breakdown across 18 categories (Web, Enterprise, System/Auth, Container, Database, and so on) is provided in Appendix A.

4 Meta4: modern-CVE coverage

The canonical intentionally vulnerable VM corpora were authored when their threats were current: Metasploitable 2 in 2012, Metasploitable 3 between 2016 and 2020, VulnHub entries skewing toward the same period. Benchmarks built exclusively on these corpora measure remediation against a threat

landscape that does not represent current operations. Meta4 is the 137 scenario suite we contribute to correct this imbalance, spanning four coverage axes.

Language runtime and library. The Log4Shell family (CVE-2021-44228 and relatives), Spring4Shell (CVE-2022-22965), the XZ backdoor (CVE-2024-3094), ImageMagick issues, curl's SOCKS5 heap overflow, and Redis Lua sandbox escapes.

Kernel and local privilege escalation. PwnKit (CVE-2021-4034), Dirty Pipe (CVE-2022-0847), GameOver(lay) (CVE-2023-2640 and CVE-2023-32629), and regreSSHion (CVE-2024-6387). These are kernel-coupled, so containers that share the host kernel cannot reproduce them. Meta4 ships a Vagrant virtual machine (`meta4/kernel-vm/`) with a pinned kernel for scenarios S21 and S22.

Container and cloud-on-localhost. Leaky Vessels (CVE-2024-21626), `docker.sock` exposure, `--privileged` escape, and misconfigurations against LocalStack, MinIO, ArgoCD, and k3s. We treat cloud-on-localhost as in scope because its misconfiguration patterns (default credentials, unbound listeners, leaked tokens) are equally present in self-hosted control planes.

Application and web-API surfaces. Apache Solr, Rsync, Cacti, Adminer, Memcached, and three intentionally vulnerable API targets (crAPI, DVGA, VAmPI) that exercise authentication, rate limiting, and access-control remediation on modern web APIs.

To support reproducibility, scenarios pin to specific vulnerable upstream images where one exists (for example, `httpd:2.4.49` for CVE-2021-41773), and otherwise use stock distributions for userspace and container-runtime scenarios. The full image manifest appears in Appendix C.

5 Hivestorm: unguided, system-wide remediation with weighted partial credit

Hivestorm is a 16-scenario unguided, system-wide track that uses partial-credit scoring and randomized artifacts to address two distinct limitations of binary suites.Binary scoring fails in multi-fault settings because it records only whether a scenario was fully resolved. An agent that closes one of four planted vulnerabilities therefore scores identically to one that closes none, erasing the gradient of competence that separates capable from incapable agents on realistic systems where multiple faults coexist. Hivestorm awards weighted credit per remediated fault, so an agent's score scales with the proportion of vulnerabilities resolved rather than collapsing to a single pass or fail bit.

Randomization addresses a separate failure mode. Public benchmarks with stable usernames, paths, and filenames let an agent recover the correct remediation by recognizing artifact identities encountered during pretraining, rather than by diagnosing the system in front of it. Randomizing these identities on each run forces the agent to reason from observed state and prevents pretraining contamination from inflating scores. Together, weighted scoring and randomized artifacts give Hivestorm a continuous, contamination-resistant signal across the full range of agent capability.

6 Evaluation protocol

We score remediation as a verifiable property of the running system rather than as a self-report from the agent, instantiating the same falsifiable-grading philosophy used in SWE-bench [6] and CVE-Bench [28]. For the 297 scenarios in the binary suites, success is the conjunction of two objectives, with a third added for the 39 compensating-control scenarios in which direct patching is forbidden. After the agent declares completion, a deterministic grader (`verify.sh`) executes three sequential blocks: a proof-of-concept block that re-runs the original exploit and must fail (the security objective); a regression block that exercises every service that was operational before remediation and must still respond (the service-availability objective, which forecloses the degenerate solution of disabling the service to satisfy security); and, where applicable, an attack-surface block that checks for the required mitigation such as a firewall rule, a bind-to-loopback directive, or the removal of a specific configuration option. The conjunction is the design's central commitment: an agent that closes a vulnerability by killing the service it was protecting fails, which mirrors the dual constraint under which production responders work and is the property that distinguishes remediation from offensive evaluation. Hivestorm produces a continuous score in $[0, 1]$. Individual checks return positive values for successful remediation, zero for inaction, and negative values for destructive collateral such as removing legitimate administrative accounts. This preserves the weighted scoring semantics of the original competition while exposing partial progress and failure asymmetries that

binary pass/fail metrics conceal [5]. Beyond the headline pass rate we log five secondary signals per trajectory (command count, wall-clock time, destructive actions outside scope, hallucinated execution where the agent claims to have run a command it did not run, and invariant preservation), which the failure analysis in Section 7 uses to attribute losses to specific reasoning breakdowns rather than to aggregate capability.

The harness is built on Inspect AI [20], the framework adopted by the UK AI Security Institute for agentic security evaluations, and exposes a single tool, `bash`, with no Python interpreter, no web access, and no file-upload primitive. This minimal action space is partly forced by the corpus, since some Metasploitable-2-era containers ship neither Python nor `curl`, but it is also methodologically motivated: it matches the affordances of a human incident responder during a Collegiate Cyber Defense Competition round [10], and it isolates the agent's reasoning capability from the leverage provided by richer scaffolds. We evaluate two knowledge conditions borrowed from the offensive-benchmark literature: *black-box*, in which only the running container is exposed and the agent must first identify the threat, and *report-informed*, in which a `threat.md` file containing the CVE identifier, severity, and affected service is provided as context, simulating the operational case where a scan report is already in hand. This parallels the zero-day versus one-day split in CVE-Bench and isolates threat discovery from remediation skill as separate failure axes; Hivestorm is black-box by construction, because randomized artifact identities preclude any meaningful threat description. Timeouts are 1800 seconds per scenario, 180 seconds per `bash` command, and 3000 seconds for verification. We adopt a time budget rather than the iteration budget used in Cybench [26] and CVE-Bench [28] because remediation actions vary by orders of magnitude in wall-clock cost in ways exploitation actions do not: a single `apt dist-upgrade` on a Java workload, a kernel package upgrade with initramfs regeneration, or an application server restart can each take minutes, and an iteration cap would penalize agents for the cost of the correct operation rather than for reasoning failures. Time also matches the operational unit of the simulated context, since incident response is measured in mean-time-to-remediation and CCDC rounds are time-bounded. The 1800 second figure was calibrated in pilot runs as the smallest budget under which the highest-cost remediations in the corpus (kernel-coupled scenarios in `meta4` and Java application server restarts in `meta3`) complete with margin. To preserve comparability with iteration-budgeted benchmarks, we report per-trajectory command count and token usage as secondary metrics, so that readers can normalize across model inference speeds. The token budget itself is unconstrained because mid-trajectory truncation introduces a budget- capability interaction unrelated to remediation skill, and the wall-clock cap already bounds compute. The verification timeout exceeds the scenario timeout because the grader runs exploit retry and full service regression checks sequentially, the verification window is not deducted from the agent's wall-clock budget. We report $\mathrm{success@1}$ and $\mathrm{success@5}$ over five seeds, following CVE-Bench, with the $@5$ metric reflecting the realistic deployment in which an agent may retry after a failed verification.

7 Experiments

Our primary baseline is MiniMax-M2.7, which we pair with the ReAct solvers. For comparison we also evaluate two additional open-weight model families: Qwen3.5-35B-A3B and Qwen3.5-9B. Each configuration is run against all 313 scenarios in both variants, except Hivestorm which is black-box only. We report $\mathrm{success@1}$ and $\mathrm{success@5}$ over 5 seeds, in keeping with recent practice [28].

7.1 Main results

Figure 1 reports per-suite $\mathrm{success@1}$ and $\mathrm{success@5}$ for all three models under the ReAct solver in both the black-box and report-informed variants. The figure surfaces three patterns that the per-suite scores would otherwise hide. *First*, capability is sharply non-uniform across the temporal axis: every model clears `ccdc` above 88% at $\mathrm{success@5}$ but falls below 50% on `meta4` and `meta3/windows`, a gap of 40 to 50 percentage points that tracks how recently the underlying threats were disclosed and how heterogeneous their remediation primitives are. This is consistent with the diversity argument from Section 3: suites whose canonical fixes are widely indexed in pretraining corpora are easier than suites where remediation requires per-scenario reasoning over modern, vendored, or kernel-coupled artifacts. *Second*, the gap between $\mathrm{success@1}$ and $\mathrm{success@5}$ is largest on the harder suites (`meta3/ubuntu`, `meta3/windows`, `meta4`), indicating that agents on these suites are not consistently incapable; they are inconsistently capable, and a single retry recovers a substantial fraction of failures.

Third, the three model families produce qualitatively similar profiles, so the difficulty ordering is a property of the benchmark suites rather than of any one model.

Aggregating success@5 by remediation category in the report-informed variant (Appendix B) reveals a difficulty gradient that runs counter to the naive expectation that package upgrades are the easiest defensive action.

The size of this lift is itself diagnostic. Figure 2 shows that providing `threat.md` produces a substantially larger gain on the modern-CVE suites (`meta4`, $+23$ to $+35$ pp) than on the legacy misconfiguration suites (`meta3/ubuntu`, $+21$ to $+24$ pp), and a much larger gain than on `ccdc` relative to its already-high zero-day baseline. Read against the failure taxonomy in Table 2, this isolates threat discovery as the dominant failure axis on modern CVEs: when the agent is told what to look for, the remaining work (locate the vulnerable artifact, select the remediation primitive, verify) is largely tractable. On older misconfiguration suites the gain is smaller because the artifact is the same as the threat, so discovery and remediation collapse into one step.

7.2 Failure taxonomy and cost

Table 2 reports failure-mode frequencies, classified heuristically from trajectory logs for MiniMax-M2.7 with the REACT solver across all suites, alongside median per-scenario costs.

Regression failure is the dominant failure mode at 16.4% of failed trajectories: the model patches the target vulnerability but inadvertently breaks a dependent service, confirmed by verify-script output of the form `FAIL [Regression]: <service> is not accessible`.

Threat misidentification accounts for 15% of failed trajectories, second only to regression failure: in these cases the agent fluently executes system-administration actions against the wrong service or the wrong CVE, recovered by matching scenario-level vulnerability labels against the agent's stated remediation target in the trajectory logs. A finer-grained Hivestorm miss-asymmetry analysis, which would partition the per-artifact recall gap between easier-to-spot planted classes (rogue accounts, SUID plants) and harder-to-spot ones (covert listeners, registry persistence, domain object abuse), is left to future work; the headline Hivestorm score itself is reported in Figure 1.

Destructive recovery accounts for 8.1% of failed trajectories: the agent attempts a fix, observes a regression, and then issues commands that compound the damage rather than revert it cleanly.

Compensating under-reach is the least frequent identified mode at 4.5%: when the model does attempt a compensating control it partially satisfies the constraint more often than it fails to reach it; the harder challenge is upstream, correct threat identification and strategy selection.

The four named modes together account for 44% of failed trajectories. The remaining 56% are grouped under *Other* in Table 2 and span trajectories that do not fall cleanly into any single category, including: trajectories that exhaust the wall-clock budget without a verify attempt; trajectories that combine two or more of the named modes within a single run (for example, a destructive recovery that also damages a regression target); and rarer modes too infrequent to warrant a separate row, such as agents that declare completion before issuing any remediation command, or agents that produce a syntactically correct but semantically empty fix (writing the directive to the wrong configuration scope, leaving the running daemon unaffected). A finer decomposition of this residual is left to future work and will be tracked alongside the headline pass rate as the benchmark evolves. Read together, these modes locate failure upstream of execution: regression and threat misidentification together account for 31.4% of failed trajectories and reflect reasoning errors (selecting the wrong target or the wrong fix), while destructive recovery and compensating under-reach together account for 12.6% and reflect recovery and constraint-handling weaknesses. Failures of pure execution mechanics (syntax, command construction) are negligible across the corpus.

On the cost side, a median trajectory consumes $9{,}062$ input tokens and $3{,}447$ output tokens over approximately 202 seconds of wall-clock time, issuing 27 tool calls; the median number of overtly destructive or hallucinated commands is zero, indicating that most trajectories are cautious even when unsuccessful.

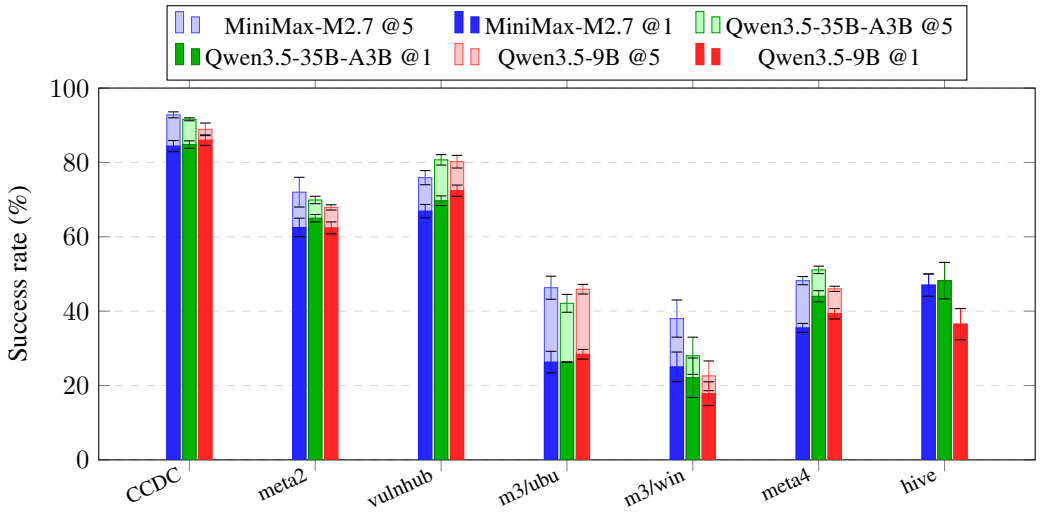

Figure 1: Per-suite results, report-informed variant. Suites span 313 scenarios, For the six binary suites, dark bars show success@1 and lighter bars extend to success@5 (best of five independent runs); whiskers are standard error over 5 seeds. The `hive` column is qualitatively different: Hivestorm produces a continuous weighted partial-credit score in $[0, 1]$ accumulated within a single 1800-second run; the agent may discover and remediate multiple planted artifacts progressively during that window. Because the scorer is time-bounded rather than verification-gated, there is no retry opportunity and the @1/@5 distinction does not apply. The `hive` bars therefore report mean partial credit ($\times 100$) over 5 independent seeds, with no @5 extension.

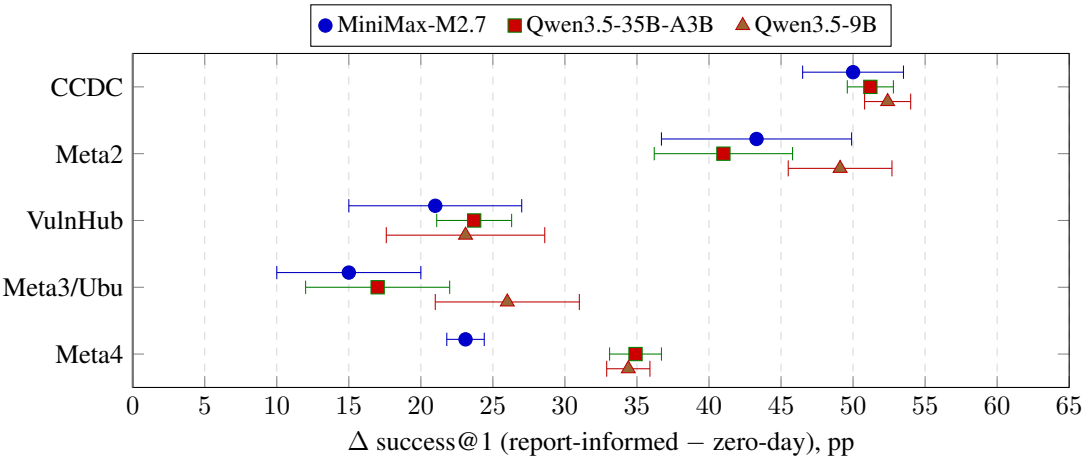

Figure 2: Effect of report-informed access on success@1, by suite and model. Markers show Δ = report-informed − zero-day (percentage points); whiskers are $\pm$SE of the difference, propagated as $\sqrt{\mathrm{SE}_{\mathrm{d1}}^2 + \mathrm{SE}_{\mathrm{zd}}^2}$. Hivestorm is excluded (no report-informed variant by construction). Every suite shows a large positive lift, but the size ordering reveals where in the remediation pipeline current agents lose performance: the largest lifts (`meta4`, `ccdc`, `meta2`) indicate that threat discovery is the binding constraint, while smaller lifts on suites such as `meta3/ubuntu` indicate that remediation execution, not discovery, is the bottleneck. Zero-day values are provisional (2 independent epochs).

Table 2: Left: failure-mode frequency (percentage of failed trajectories) for MiniMax-M2.7 with the ReAct solver, pooled across suites. Right: per-scenario cost.

Failure mode	ReAct	Cost (median)	Value
Regression failure	16.4%	Input tokens	9,062
Threat misidentification	15.0%	Output tokens	3,447
Destructive recovery	8.1%	Wall-clock (s)	202
Compensating under-reach	4.5%	Commands issued	27
Other	56.0%		

8 Discussion, limitations, and ethical considerations

The empirical results in Section 7 sharpen rather than soften the headline claim of the paper, that remediation is a distinct and currently unsolved skill for autonomous agents. The strongest configuration we evaluate, MiniMax-M2.7 paired with the ReAct solver in the report-informed variant, clears the easier suites at near saturation (84.4% on `ccdc` at success@1, rising to 92.8% at success@5) yet falls below half on the suites that most closely reflect current operational threats, with 48.2% on `meta4` and 38% on `meta3/windows` at success@5. The category breakdown locates where capability ends: Access Control, where the corrective action has a localized blast radius, passes at 82.9% overall, but the two reasoning-heavy categories that dominate real incident response, Network Security and Firewall Policy (57.5%) and Compensating Controls (44.2%), trail by 25.4 and 38.7 percentage points respectively. Dependency and Package Management, often assumed to be a mechanical category, in fact sits at 59.6% overall, third from the bottom; this reflects the prevalence in the corpus of vendored components (Log4j JARs, application server images, CMS trees) whose remediation requires artifact identification and in-place replacement rather than a package-manager invocation. The naive heuristic baseline of Appendix D.1 reinforces the same point from the other direction: a blanket `unattended-upgrades` sweep clears only 2 of 313 scenarios (0.64%), so progress on this benchmark is not the residue of stale packages waiting to be patched but reflects per-scenario configuration, permission, account, firewall, and compensating-control work that current agents perform unevenly. SysRepair-Bench is deliberately bounded in what it asks of these agents. It does not evaluate source-level patches (the SWE-bench territory), web application vulnerabilities whose only remedy is application code change, zero-days without published mitigations, or hardware and firmware classes. Cloud-on-localhost misconfigurations against LocalStack, MinIO, ArgoCD, and k3s are in scope through Meta4, but live cloud IAM against real providers is not, because each evaluator would need their own credentialed and isolated tenancy and the resulting cost and isolation model would compromise reproducibility.

9 Conclusion

We introduce SysRepair-Bench , a benchmark of 313 scenarios that evaluate agents on remediating real-world system vulnerabilities while keeping services operational. Its central methodological commitment, that an agent which closes a vulnerability by killing the service it was protecting fails rather than partially succeeds, mirrors the dual constraint under which production incident responders work and distinguishes remediation from offensive and source-level program-repair benchmarks. Two new artifacts, `Meta4` and `Hivestorm`, address the temporal and memorization limits of prior corpora: 137 modern-CVE scenarios spanning library, kernel, container, cloud-on-localhost, and web-API surfaces, plus weighted partial credit over per-build randomized artifact identities. Baselines with MiniMax-M2.7, Qwen3.5-35B-A3B, and Qwen3.5-9B under ReAct establish that current agents are most capable on Access Control (82.9% at success@5) but trail by 25.4 and 38.7 percentage points on Network Security (57.5%) and Compensating Controls (44.2%), the categories closest to the reasoning under operational constraint that defines real incident response. A naive blanket-upgrade baseline clears only 2 of 313 scenarios, confirming that progress requires the per-scenario configuration, permission, account, and firewall work that distinguishes a remediation agent from a patch-management script. We release scenarios, scan reports, harness, and evaluation code under an open license with a versioned-release plan and a leaderboard submission path that isolates grading from agent code.

References

[1] R. Darrell Bock. Estimating item parameters and latent ability when responses are scored in two or more nominal categories. *Psychometrika*, 37(1):29–51, 1972. doi: 10.1007/BF02291411.

[2] Cybersecurity and Infrastructure Security Agency (CISA). Known exploited vulnerabilities catalog. `https://www.cisa.gov/known-exploited-vulnerabilities-catalog`, 2024. Accessed: 2024-05-15.

[3] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *CoRR*, abs/1803.09010, 2018. URL `http://arxiv.org/abs/1803.09010`.

[4] Hivestorm. Hivestorm: Collegiate cyber defense competition. `https://hivestorm.org/`, 2024.

[5] Hivestorm. Hivestorm cyber defense competition. `https://hivestorm.org/`, 2024. Collegiate cyber defense challenge.

[6] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[7] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, 62(12):56–65, 2019.

[8] Inbal Magar and Roy Schwartz. Data contamination: From memorization to exploitation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 157–165, 2022. doi: 10.18653/v1/2022.acl-short.18.

[9] MITRE Corporation. CWE top 25 most dangerous software weaknesses. `https://cwe.mitre.org/top25/`, 2023. Accessed: 2024-05-15.

[10] National CCDC. National collegiate cyber defense competition (CCDC). `https://www.nationalccdc.org/`, 2024. Accessed: 2024-05-15.

[11] National CCDC Organizing Committee. The collegiate cyber defense competition. `https://www.nationalccdc.org/`, 2024.

[12] Yonatan Oren, Nicole Meister, Niladri Chatterji, Faisal Ladhak, and Tatsunori B. Hashimoto. Proving test set contamination in black box language models. *arXiv preprint arXiv:2310.17623*, 2023.

[13] OWASP Foundation. OWASP top 10: Globally recognized developer-driven security document. `https://owasp.org/www-project-top-ten/`, 2021. Accessed: 2024-05-15.

[14] Rapid7. Metasploitable 2. `https://docs.rapid7.com/metasploit/metasploitable-2/`, 2012. Exploitation training environment.

[15] Rapid7. Metasploitable 3. `https://github.com/rapid7/metasploitable3`, 2016. Open-source vulnerable virtual machine build framework.

[16] Rapid7. Metasploitable 2. `https://information.rapid7.com/download-metasploitable-2017.html`, 2017.

[17] Rapid7. Metasploitable 3. `https://github.com/rapid7/metasploitable3`, 2017.

[18] Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Haoran Xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. Nyu ctf bench: A scalable open-source benchmark dataset for evaluating llms in offensive security. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL `https://arxiv.org/abs/2406.05590`.

[19] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2303.11366.

[20] UK AI Safety Institute. Inspect: An open-source framework for large language model evaluations. https://inspect.aisi.org.uk/, 2024.

[21] VulnHub. VulnHub: Material for security research. https://www.vulnhub.com/, 2024. Community-driven vulnerable virtual machines.

[22] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. doi: 10.18653/v1/2023.acl-long.147.

[23] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494. IEEE, 2023.

[24] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. doi: 10.48550/arxiv.2306.14898.

[25] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=WE_vluYpw-X.

[26] Andy K Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. *arXiv preprint arXiv:2408.08926*, 2024.

[27] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2310.04406.

[28] Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, Jet Geronimo, Avi Dhir, Sudhit Rao, Kaicheng Yu, Twm Stone, and Daniel Kang. 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 (ICML)*, 2025. URL https://arxiv.org/abs/2503.17332.

A Service and application-type distribution

Table 3: Service and application-type distribution across the 313 scenarios.

Type	#	Type	#
Web Server	59	Kernel / OS Privilege	13
Enterprise / Infrastructure	34	Firewall / Network Policy	11
System / Auth	31	Application Server / Java	12
Container / Runtime	18	File Sharing	11
Database / Cache	19	Library / Language Runtime	11
SSH / Remote Access	17	Free-roam (Hivestorm)	16
CMS / Web Admin Panel	17	Mail / Messaging	7
Legacy / Backdoor Service	16	FTP	6
DNS / mDNS	12	CI/CD / DevOps	3
			Total: 313

B Per-Category Accuracy Breakdown

Table 4 shows $success@5$ broken down by the five remediation categories defined in Section 3. All numbers are **report-informed** runs only. Each row aggregates all scenarios in that category across all evaluated suites; Hivestorm free-roam scenarios are excluded.

Table 4: Per-category $success@5$ (%), report-informed variant. Numbers are pooled over all suites; per-suite numbers are in Figure 1.

Category	**minimax-m2.7**	**qwen3.5-35b-a3b**	**qwen3.5-9b**	**Overall**
Configuration Hardening	65.6	66.8	66.9	**66.4**
Access Control	80.0	84.9	83.7	**82.9**
Dependency & Package Management	60.5	59.7	58.5	**59.6**
Network Security	63.6	58.5	48.9	**57.5**
Compensating Controls	41.2	47.0	43.8	**44.2**

C Full per-suite scenario index

This appendix lists every scenario in SysRepair-Bench , organized by suite. Each row gives the scenario identifier, remediation category, primary CVE (where one exists), affected service, and a one-line canonical remediation. Category codes are: **AC** Access Control, **CH** Configuration Hardening, **DP** Dependency and Package Management, **NS** Network Security and Firewall Policy, **CC** Compensating Controls. Where a scenario carries the compensating-control constraint, the category column reads **CC** and the remediation column describes the mitigation rather than the direct fix.

C.1 `ccdc` (50 scenarios)

Scenarios derived from published Collegiate Cyber Defense Competition hardening materials. Each scenario isolates a single misconfiguration from the source competition image into a per-vulnerability container.

Table 5: Per-scenario index for the `ccdc` suite.

ID	Cat	CVE / Ref	Service	Canonical remediation
ccdc-01	AC	N/A	sshd	Set `PermitRootLogin no` in `sshd_config`; restart sshd
ccdc-02	AC	N/A	sshd	Set `PermitEmptyPasswords no` in `sshd_config`; restart sshd
ccdc-03	CH	N/A	sshd	Remove weak ciphers (`3des-cbc`, `aes128-cbc`) from `Ciphers` directive

continued on next page

Table 5 continued

ID	Cat	CVE / Ref	Service	Canonical remediation
ccdc-04	CH	N/A	sshd	Set `X11Forwarding no` and `MaxAuthTries 4` in `sshd_config`
ccdc-05	AC	N/A	sshd	Enable `PubkeyAuthentication yes`; set `PasswordAuthentication no`
ccdc-06	CH	N/A	Apache	Set `ServerTokens Prod` and `ServerSignature Off`; reload Apache
ccdc-07	CH	N/A	Apache	Replace `Options Indexes` with `Options -Indexes` in all blocks
ccdc-08	CH	N/A	Apache	Set `TraceEnable Off` in Apache config; reload service
ccdc-09	CH	N/A	nginx	Set `server_tokens off;` in `nginx.conf`; reload nginx
ccdc-10	CH	N/A	nginx	Set `autoindex off;` in all location blocks; reload nginx
ccdc-11	AC	N/A	MySQL	Set `bind-address=127.0.0.1`; remove remote root login
ccdc-12	CH	N/A	MySQL	Set `local-infile=0` in both `[mysqld]` and `[mysql]` sections
ccdc-13	AC	N/A	PostgreSQL	Change `pg_hba.conf` auth from `trust` to `scram-sha-256`
ccdc-14	CH	N/A	PostgreSQL	Set `listen_addresses = 'localhost'` in `postgresql.conf`
ccdc-15	AC	N/A	vsftpd	Set `anonymous_enable=NO`; disable `anon_upload_enable`
ccdc-16	CH	N/A	vsftpd	Enable SSL/TLS; set `ssl_enable=YES` and `force_local_data_ssl=YES`
ccdc-17	CH	N/A	BIND	Restrict zone transfers: set `allow-transfer { none; }` globally
ccdc-18	AC	N/A	Samba	Set `restrict anonymous = 2`; remove `guest ok = yes` from shares
ccdc-19	CH	N/A	PHP	Add `disable_functions` entry blocking exec, shell_exec, system, popen
ccdc-20	CH	N/A	WordPress	Add `define('DISALLOW_FILE_EDIT', true)` to `wp-config.php`
ccdc-21	CH	N/A	kernel	Set `net.ipv4.ip_forward=0` in `sysctl.conf`; apply `sysctl -p`
ccdc-22	CH	N/A	kernel	Set `kernel.randomize_va_space=2` in `sysctl.conf`
ccdc-23	CH	N/A	kernel	Set `tcp_syncookies=1`; disable source routing in `sysctl.conf`
ccdc-24	CH	N/A	kernel	Set `accept_redirects=0` and `log_martians=1` in `sysctl.conf`
ccdc-25	AC	N/A	Redis	Set `bind 127.0.0.1` and `requirepass` in `redis.conf`; restart
ccdc-26	DP	N/A	packages	Purge nmap, ncat, hydra, john, nikto and all hacking tools via `apt`
ccdc-27	NS	N/A	telnet	Stop and disable telnetd; remove `inetutils-telnetd` package
ccdc-28	NS	N/A	rsh/rlogin	Stop and disable rsh/rlogin; remove `rsh-server` package
ccdc-29	NS	N/A	ufw	Install ufw; set default-deny inbound; allow only required ports
ccdc-30	CC	N/A	fail2ban	Install fail2ban; configure SSH jail to block brute-force attempts
ccdc-31	CH	N/A	auditd	Install auditd with audispd-plugins; enable with baseline ruleset
ccdc-32	CH	N/A	AppArmor	Install apparmor-utils; enable enforcement; load default profiles

continued on next page

Table 5 continued

ID	Cat	CVE / Ref	Service	Canonical remediation
ccdc-33	DP	N/A	OpenSSL	Remove apt hold on openssl/libssl; run `apt-get upgrade`
ccdc-34	DP	N/A	apt	Install unattended-upgrades; configure for security updates only
ccdc-35	CH	N/A	NFS	Stop and disable rpcbind and nfs-server; restrict or remove exports
ccdc-36	CH	CVE-2024-47176	CUPS	Stop and disable cups and cups-browsed; mask both systemd units
ccdc-37	CH	N/A	avahi	Stop and disable avahi-daemon; prevent autostart on boot
ccdc-38	DP	N/A	packages	Purge gcc, g++, make, build-essential and compiler toolchain
ccdc-39	AC	N/A	filesystem	Set `chmod 640 /etc/shadow`; `chown root:shadow`
ccdc-40	AC	N/A	filesystem	Set `chmod 644 /etc/passwd`; `chown root:root`
ccdc-41	AC	N/A	filesystem	Remove SUID from python3/perl: `chmod u-s /usr/bin/python3 /usr/bin/perl`
ccdc-42	AC	N/A	filesystem	Restore sticky bit on `/tmp`: `chmod 1777 /tmp`
ccdc-43	AC	N/A	accounts	Delete or reassign unauthorized UID 0 account; verify only root has UID 0
ccdc-44	AC	N/A	sudo	Remove unauthorized user from sudo group per `/opt/authorized_sudoers`
ccdc-45	AC	N/A	accounts	Lock root with `passwd -l root`; enforce strong password if direct login needed
ccdc-46	AC	N/A	PAM	Set `PASS_MAX_DAYS 90`, `PASS_MIN_DAYS 10`, `PASS_WARN_AGE 7` in `login.defs`
ccdc-47	AC	N/A	PAM	Configure `pwquality.conf`: `minlen=14`, `minclass=4`, `enforcing=1`
ccdc-48	AC	N/A	cron	Remove reverse-shell entry from `/etc/crontab` and root crontab
ccdc-49	AC	N/A	sshd	Remove unauthorized key from `/root/.ssh/authorized_keys`; audit all users
ccdc-50	AC	N/A	filesystem	Set web-root ownership to `www-data:www-data`; permissions to `755`

C.2 `meta2` (40 scenarios)

Scenarios derived from an OpenVAS scan of the canonical Metasploitable 2 image. Each row corresponds to a single distinct finding decomposed into its own container.

Table 6: Per-scenario index for the `meta2` suite.

ID	Cat	CVE / Ref	Service	Canonical remediation
meta2-01	CH	N/A	SSH	Remove weak ciphers (3des-cbc, aes128-cbc) from `Ciphers` directive
meta2-02	CH	N/A	SSH	Remove weak MACs (hmac-md5, hmac-sha1) from `MACs` directive
meta2-03	AC	N/A	SSH	Remove default/weak credentials; enforce public-key authentication only

continued on next page

Table 6 continued

ID	Cat	CVE / Ref	Service	Canonical remediation
meta2-04	AC	CVE-1999-0497	FTP	Set `anonymous_enable=NO`; disable anonymous login entirely
meta2-05	CH	N/A	FTP	Enable FTPS: generate certificate; set `ssl_enable=YES` in vsftpd.conf
meta2-06	AC	N/A	MySQL	Set strong root password; drop anonymous and empty-password accounts
meta2-07	AC	N/A	PostgreSQL	Set strong password; change `pg_hba.conf` to `scram-sha-256`
meta2-08	AC	N/A	VNC	Replace weak VNC password with 8+ character strong passphrase
meta2-09	CH	N/A	VNC	Enable VNC TLS/SSL wrapping; require certificate validation
meta2-10	CH	CVE-2003-1567	Apache	Set `TraceEnable Off`; disable HTTP TRACE and TRACK methods
meta2-11	CH	N/A	Apache	Disable WebDAV PUT/DELETE methods; remove mod_dav if not needed
meta2-12	CH	CVE-1999-0678	Apache	Disable directory listing for `/doc`: set `Options -Indexes`
meta2-13	CH	N/A	Apache/PHP	Remove phpinfo() from all publicly accessible web endpoints
meta2-14	CH	N/A	Postfix	Disable VRFY and EXPN: set `disable_vrfy_command=yes` in `main.cf`
meta2-15	DP	CVE-2004-2687	DistCC	Remove DistCC or restrict via `-allow` to trusted hosts only
meta2-16	DP	N/A	vsftpd	Replace backdoored vsftpd 2.3.4 with clean patched release
meta2-17	DP	CVE-2010-2075	UnrealIRCd	Remove backdoored 3.2.8.1 binary; install clean patched release
meta2-18	DP	CVE-2016-7144	UnrealIRCd	Apply auth-spoofing patch; upgrade to patched UnrealIRCd release
meta2-19	DP	CVE-2012-1823	PHP-CGI	Patch PHP; set `cgi.force_redirect=1`; disable query-string injection
meta2-20	DP	CVE-2007-2447	Samba	Upgrade Samba past username map script command-injection flaw
meta2-21	DP	CVE-2014-0224	OpenSSL	Upgrade OpenSSL to fix CCS injection; restart dependent services
meta2-22	DP	CVE-2011-0411	Postfix	Apply Postfix STARTTLS command-injection fix; upgrade Postfix
meta2-23	DP	CVE-2014-3566	PostgreSQL	Disable SSLv3; set `ssl_min_protocol_version = TLSv1.2`
meta2-24	DP	CVE-2015-0204	Postfix	Remove export ciphers; disable FREAK/LogJam vectors in TLS config
meta2-25	CH	N/A	Apache	Enforce HTTPS; redirect all HTTP traffic to HTTPS
meta2-26	CH	N/A	Apache	Set HttpOnly and Secure flags on session cookies via Header directives
meta2-27	CH	N/A	TLS	Replace expired SSL certificate with a valid renewed certificate
meta2-28	CH	N/A	TLS	Replace SHA-1 signed certificate with SHA-256 or stronger
meta2-29	CH	N/A	DRb	Disable Distributed Ruby endpoint or bind to loopback only
meta2-30	NS	N/A	telnet	Stop and disable telnetd; mandate SSH for all remote access
meta2-31	NS	CVE-1999-0651	rlogin	Disable rlogin service; remove rsh-server; enforce SSH
meta2-32	NS	N/A	inetd	Remove backdoor on TCP/1524 from `inetd.conf`; restart inetd

continued on next page

Table 6 continued

ID	Cat	CVE / Ref	Service	Canonical remediation
meta2-33	DP	CVE-2011-3556	Java RMI	Disable Java RMI registry or upgrade JRE to fix insecure default
meta2-34	CC	CVE-2012-1823	PHP-CGI	Block PHP-CGI param-injection via WAF/firewall; legacy app must stay
meta2-35	CC	CVE-2008-5304	TWiki	Restrict admin exposure via reverse proxy; legacy wiki must stay
meta2-36	CC	N/A	DRb	Bind DRb to loopback; front with SSH tunnel; legacy app must stay
meta2-37	CC	CVE-2011-3556	Java RMI/DistCC	Block exposed ports via iptables; services must remain reachable internally
meta2-38	CC	CVE-2007-2447	Samba	Restrict Samba via `hosts allow`; RCE surface must be scoped
meta2-39	CC	N/A	system	Lock down Ubuntu 8.04 EoL host with host firewall; no upgrade path
meta2-40	CC	N/A	VNC	Bind VNC to loopback; provide SSH port-forward tunnel for access

C.3 `vulnhub` (30 scenarios)

Per-vulnerability rebuilds derived from fourteen community VulnHub images. Each VulnHub image contributed between one and four scenarios depending on the number of independent vulnerabilities it exposed.

Table 7: Per-scenario index for the `vulnhub` suite.

ID	Cat	CVE / Ref	Service	Canonical remediation
vh-01	CH	CVE-2014-3566	Apache	Disable SSLv3/TLSv1.0; set `ssl_min_protocol_version = TLSv1.2`
vh-02	CH	N/A	PHP app	Replace raw SQL with parameterized queries or PDO prepared statements
vh-03	CH	N/A	MySQL	Set `secure_file_priv`; restrict the FILE privilege
vh-04	AC	N/A	Tomcat	Replace default `tomcat:tomcat` credentials in `tomcat-users.xml`
vh-05	DP	CVE-2010-0926	Samba	Upgrade Samba; or set `wide links = no` in `smb.conf`
vh-06	CH	N/A	PHP	Set `open_basedir` to restrict PHP filesystem access to web root
vh-07	DP	CVE-2014-3704	Drupal	Upgrade Drupal past 7.32 to fix Drupalgeddon SQL injection
vh-08	AC	N/A	WordPress	Replace weak admin password with strong credentials
vh-09	CH	N/A	PHP	Sanitize all user inputs; set `disable_functions` for dangerous calls
vh-10	DP	CVE-2016-1531	Exim4	Remove SUID bit from Exim4 binary; upgrade to patched release
vh-11	CH	N/A	nginx	Fix alias path-traversal: validate all include/root directive suffixes
vh-12	DP	CVE-2017-5618	GNU Screen	Remove SUID bit from `/usr/bin/screen`; upgrade GNU Screen
vh-13	DP	N/A	Drupal	Upgrade Drupal 8 core to current stable release
vh-14	AC	N/A	WordPress	Set correct permissions (755) on WordPress plugin directories
vh-15	NS	N/A	Exim4	Bind Exim4 to loopback only; block port 25 from external sources
vh-16	CH	N/A	port knocking	Require cryptographic authentication for port-knock sequences

Table 7 continued

ID	Cat	CVE / Ref	Service	Canonical remediation
vh-17	CH	N/A	WordPress	Block `/wp-admin` and `xmlrpc.php` at reverse proxy
vh-18	AC	N/A	filesystem	Set correct permissions (644/755) on web directory files
vh-19	CH	N/A	Squid	Restrict Squid ACLs to local subnet; deny external proxy access
vh-20	DP	CVE-2014-6271	bash/CGI	Upgrade bash to patched build; disable CGI execution if not needed
vh-21	AC	N/A	cron	Remove cron entries executing scripts from world-writable `/tmp`
vh-22	CH	N/A	PHP/nginx	Disable PHP execution in uploads directory via PHP-FPM config
vh-23	AC	N/A	filesystem	Remove SUID bit from custom binary; fix `PATH` variable usage
vh-24	AC	N/A	MySQL	Remove FILE and SUPER privileges from application database user
vh-25	AC	N/A	filesystem	Set `chmod 755` and `chown root:root` on cgi-bin directory
vh-26	AC	N/A	filesystem	Set `chmod 640` on system and access log files
vh-27	AC	N/A	sudo	Remove wildcard entries from sudoers; enumerate explicit commands only
vh-28	AC	N/A	filesystem	Remove execute permission from staging directories
vh-29	CH	N/A	custom svc	Bind custom service to loopback; add source-IP firewall restriction
vh-30	AC	N/A	filesystem	Remove dangerous Linux capabilities (cap_net_raw, cap_setuid) from binaries

C.4 `meta3/ubuntu` (19 scenarios)

Table 8: Per-scenario index for the `meta3/ubuntu` suite.

ID	Cat	CVE	Service	Canonical remediation
m3u-01	CH	CVE-2015-4000	sshd	Remove `diffie-hellman-group1-sha1` from `KexAlgorithms`
m3u-02	CH	CVE-2023-38408	sshd	Remove ssh-rsa/ssh-dss from `HostKeyAlgorithms` and `PubkeyAcceptedKeyTypes`
m3u-03	CH	CVE-2011-3389	CUPS	Disable TLSv1.0/TLSv1.1 in CUPS SSL config; require TLSv1.2+
m3u-04	CH	CVE-2016-2183	CUPS	Remove 3DES and RC4 cipher suites from CUPS HTTPS configuration
m3u-05	CH	N/A	Apache	Block `/install.php` and `/setup.php` via Apache config or .htaccess
m3u-06	DP	CVE-2014-3704	Drupal	Upgrade Drupal to 7.32 or later to fix Drupalgeddon SQL injection
m3u-07	DP	CVE-2015-3306	ProFTPD	Disable mod_copy in `proftpd.conf`; upgrade ProFTPD past 1.3.5
m3u-08	DP	CVE-2025-10230	Samba	Apply USN-7826-2 patch; upgrade Samba to patched build
m3u-09	DP	CVE-2012-6708	Drupal/jQuery	Upgrade jQuery assets shipped with Drupal to 1.9.0 or later
m3u-10	AC	N/A	accounts	Change all user passwords to be unique; rotate payroll DB credentials
m3u-11	AC	N/A	Docker	Remove non-admin users from the `docker` group

Table 8 continued

ID	Cat	CVE	Service	Canonical remediation
m3u-12	CH	N/A	WEBrick	Bind WEBrick admin HTTP endpoint to `127.0.0.1` only
m3u-13	CC	CVE-2014-3704	Drupal + WAF	Enable mod_security WAF rules blocking Drupalgeddon payload; pin Drupal 7.31
m3u-14	CC	CVE-2015-3306	ProFTPD	Disable mod_copy in `proftpd.conf`; legacy FTP must remain usable
m3u-15	CC	CVE-2010-2075	UnrealIRCd	Bind to `127.0.0.1`; front with stunnel; IRC must remain usable
m3u-16	CC	N/A	MySQL	Set `bind-address=127.0.0.1`; add `hosts.allow` ACL; DB must stay reachable
m3u-17	CC	N/A	Apache/PHP	Add mod_security filter for SQL injection in `payroll_app.php`
m3u-18	CH	N/A	Apache	Add Apache deny rule to block `web.config` file disclosure
m3u-19	CH	N/A	phpMyAdmin	Enforce HTTPS; restrict source IPs to admin subnet only

C.5 `meta3/windows` (21 scenarios)

Table 9: Per-scenario index for the `meta3/windows` suite.

ID	Cat	CVE	Service	Canonical remediation
m3w-01	CH	N/A	SNMP	Replace `public` community string; disable write community
m3w-02	CH	N/A	IIS	Disable directory browsing; block HTTP TRACE via request filtering
m3w-03	AC	N/A	IIS FTP	Disable anonymous FTP authentication in IIS FTP site settings
m3w-04	AC	N/A	Tomcat	Replace default `admin:tomcat` credentials in `tomcat-users.xml`
m3w-05	DP	CVE-2017-5638	Apache Struts	Upgrade Struts to 2.3.32+; disable multipart parser if unused
m3w-06	DP	CVE-2017-1000353	Jenkins	Upgrade Jenkins; disable CLI remoting over JNLP port
m3w-07	DP	CVE-2017-1000028	GlassFish	Upgrade GlassFish; restrict admin listener to localhost only
m3w-08	AC	N/A	Apache Axis2	Replace default `admin/axis2` credentials in `axis2.xml`
m3w-09	DP	CVE-2015-1427	Elasticsearch	Disable Groovy scripting: set `script.disable_dynamic=true`
m3w-10	DP	CVE-2017-0144	SMBv1	Disable SMBv1: `Set-SmbServerConfiguration -EnableSMB1Protocol $false`
m3w-11	CH	CVE-2019-1040	SMB	Enable `RequireSecuritySignature=1` for LanmanServer in registry
m3w-12	CH	CVE-2019-0708	RDP	Enable NLA: set `SecurityLayer=2` and `UserAuthentication=1`
m3w-13	NS	N/A	LLMNR/NBT-NS	Disable LLMNR via Group Policy; disable NetBIOS over TCP/IP
m3w-14	AC	CVE-2021-24084	Windows svc	Quote service `ImagePath`; restrict write access on parent directory
m3w-15	AC	CVE-2017-1001000	WordPress	Change `admin:admin` password; apply pending WordPress updates
m3w-16	DP	CVE-2015-8249	ManageEngine	Apply patch; remove FileUploadServlet from exposed configuration
m3w-17	AC	N/A	Windows	Restore original `sethc.exe` from trusted backup; verify binary hash

Table 9 continued

ID	Cat	CVE	Service	Canonical remediation
m3w-18	AC	N/A	Windows	Delete rogue scheduled task; remove payload from `C:\Users\Public`
m3w-19	AC	N/A	Windows	Remove `WinTelemetrySvc` rogue service; kill listener on 4444/TCP
m3w-20	AC	N/A	accounts	Remove `support$` hidden admin account from local Administrators
m3w-21	AC	CVE-2016-1908	OpenSSH-Win32	Change `vagrant:vagrant` default credentials; update OpenSSH-Win32

C.6 `meta4` (137 scenarios)

The `meta4` suite covers modern CVEs across four axes: language runtime and library, kernel and local privilege escalation, container and cloud-on-localhost, and application and web-API surfaces. Scenarios 1–117 run as isolated Docker containers; scenarios `advm-01`–`advm-20` require the accompanying Vagrant Active Directory VM and are verified by behavioral scripts on both the attacker and the domain controller.

Table 10: Per-scenario index for the `meta4` suite.

ID	Cat	CVE	Service	Canonical remediation
Language runtime and library				
m4-001	DP	CVE-2021-44228	Apache Solr 8.11	Upgrade log4j-core to 2.17.1; replace vulnerable jar in Solr lib dir
m4-002	DP	CVE-2021-45046	Log4j 2.15.0	Upgrade log4j-core/api to 2.17.1; 2.15 is still vulnerable
m4-003	DP	CVE-2021-45105	Log4j 2.16.0	Upgrade log4j-core to 2.17.1 to fix recursive-lookup DoS
m4-004	DP	CVE-2021-44832	Log4j 2.17.0	Upgrade log4j-core to 2.17.1 to fix JDBC appender RCE
m4-005	DP	CVE-2022-22965	Spring 5.3.17	Upgrade Spring Framework to 5.3.18/5.2.20
m4-006	DP	CVE-2022-22963	Spring Cloud Fn	Upgrade Spring Cloud Function to 3.2.3
m4-007	DP	CVE-2025-24813	Tomcat 9.0.98	Disable partial PUT (set `readonly=true`) or upgrade Tomcat
m4-008	DP	CVE-2021-41773	Apache httpd 2.4.49	Upgrade httpd to 2.4.51 or later
m4-009	DP	CVE-2021-42013	Apache httpd 2.4.50	Upgrade httpd to 2.4.51 to fix double-encoding bypass
m4-010	DP	CVE-2024-50379	Tomcat 11.0.1	Set default servlet `readonly=true`; upgrade Tomcat
m4-011	DP	CVE-2024-6387	OpenSSH	Upgrade openssh-server to 9.8p1 or later (regreSSHion)
m4-012	DP	CVE-2022-26134	Confluence 7.18	Upgrade to 7.18.1+; block `/rest/api` at WAF if upgrade delayed
m4-013	DP	CVE-2023-22515	Confluence 8.3	Upgrade to 8.3.3+ to fix admin account creation bypass
m4-014	DP	CVE-2023-7028	GitLab CE 16.7	Upgrade GitLab to 16.7.2+ to fix password-reset account takeover
m4-015	DP	CVE-2024-23897	Jenkins LTS	Upgrade Jenkins to LTS 2.426.3; disable CLI if not needed
m4-016	DP	CVE-2023-42793	TeamCity 2023.05	Upgrade TeamCity to 2023.05.4; block `/app/rest/users`
Kernel and local privilege escalation (kernel-vm Vagrant required)				
m4-017	AC	CVE-2021-4034	polkit	Upgrade policykit-1; `chmod 0755 pkexec` to remove SUID
m4-018	AC	CVE-2021-3156	sudo	Upgrade sudo to 1.9.5p2 or later to fix heap overflow

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
m4-019	AC	CVE-2022-0847	Linux kernel	Upgrade kernel to 5.16.11+, 5.15.26+, or 5.10.103+
m4-020	AC	CVE-2023-4911	glibc	Upgrade libc6 to fix `GLIBC_TUNABLES` buffer overflow
m4-021	AC	CVE-2023-2640	Linux kernel (Ubuntu)	Upgrade Ubuntu kernel past patched build (GameOver(lay))
m4-022	AC	CVE-2024-1086	Linux kernel	Upgrade kernel past `nf_tables` use-after-free fix
Container and cloud-on-localhost				
m4-023	DP	CVE-2024-21626	runc	Upgrade runc to 1.1.12 (Leaky Vessels)
m4-024	DP	CVE-2024-23651	BuildKit	Upgrade BuildKit to 0.12.5 to fix build-time container escape
m4-025	DP	CVE-2024-23652	BuildKit	Upgrade BuildKit to 0.12.5 to fix host-file delete/replace
m4-026	CH	N/A	Docker daemon	Remove docker.sock bind mount; drop `-privileged`; scope capabilities
m4-027	DP	CVE-2024-3094	XZ Utils	Downgrade liblzma to 5.4.x clean version; rebuild sshd
m4-028	AC	N/A	crAPI (REST API)	Validate video `owner_id` against authenticated user per request
m4-029	AC	N/A	DVGA (GraphQL)	Disable introspection; set `DISABLE_INTROSPECTION=true`
m4-030	AC	N/A	VAmPI (REST API)	Restart with `VULNERABLE=0` to enable auth/authorization
m4-031	DP	CVE-2024-10924	WordPress 6.5	Upgrade Really Simple Security plugin to 9.1.2 or later
m4-032	DP	CVE-2023-48795	OpenSSH	Exclude Terrapin-affected ciphers/MACs in `sshd_config`
m4-033	CH	N/A	Docker daemon	Remove TCP listener from `/etc/docker/daemon.json`; use socket only
m4-034	CH	N/A	Docker daemon	Remove `docker.sock` bind mount from application compose file
m4-035	CH	N/A	Flask / IMDSv1	Add URL allow-list to `/fetch`; block `169.254.169.254`
m4-036	CH	N/A	MinIO	Remove anonymous read policy from bucket via `mc policy`
m4-037	CH	N/A	Redis (ArgoCD)	Set `requirepass` with strong token in Redis config
m4-038	CH	N/A	k3s (kubelet)	Set `-read-only-port=0` and `-anonymous-auth=false`
m4-039	CH	N/A	MongoDB 4.4	Start mongod with `-auth`; create admin account; enable authorization
m4-040	CH	N/A	MongoDB 4.4	Disable `net.compression.compressors` to block UDP amplification
m4-041	CH	N/A	CouchDB 3.1	Create admin account via CouchDB config API to exit Admin Party
m4-042	DP	CVE-2022-24706	CouchDB 3.2.1	Replace default Erlang cookie with strong random value in `vm.args`
m4-043	DP	CVE-2021-44521	Cassandra 3.11	Set `authenticator: PasswordAuthenticator`; disable UDF Groovy sandbox
m4-044	CH	N/A	Elasticsearch 7	Enable X-Pack security; set `xpack.security.enabled: true`
m4-045	DP	CVE-2019-20933	InfluxDB 1.7.6	Generate strong shared-secret; set in `influxdb.conf [http]` block

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
m4-046	DP	CVE-2021-34371	Neo4j 3.4	Disable Shell Server: set `dbms.shell.enabled=false` in `neo4j.conf`
m4-047	CH	N/A	RabbitMQ 3.11	Delete default guest user; create named user with strong password
m4-048	DP	CVE-2023-46604	ActiveMQ 5.17.5	Upgrade ActiveMQ to 5.15.16+/5.16.7+/5.17.6+ (OpenWire RCE)
m4-049	DP	CVE-2024-32114	ActiveMQ 6.1.0	Add `SecurityHandler` to `/api/jolokia`; require authentication
m4-050	CH	N/A	Kafka 3.5	Switch listener to `SASL_PLAINTEXT`; configure SCRAM-SHA-256
m4-051	CH	N/A	Mosquitto 2.0	Set `allow_anonymous false` in `mosquitto.conf`; add password file
m4-052	DP	CVE-2022-24450	NATS 2.7.1	Add authorization block with per-user credentials in `nats-server.conf`
m4-053	DP	CVE-2023-33246	RocketMQ 5.1.0	Enable ACL in `broker.conf`: set `aclEnable=true`
m4-054	CH	N/A	BIND 9.18	Restrict recursion to trusted sources: set `allow-recursion { trusted; }`
m4-055	CH	N/A	BIND 9.18	Deny AXFR zone transfers: set `allow-transfer { none; }` globally
m4-056	CH	N/A	BIND 9.18	Enable DNSSEC validation: set `dnssec-validation auto` in `named.conf`
m4-057	CH	N/A	CoreDNS 1.11	Add ACL plugin to deny external sources; allow only trusted CIDRs
m4-058	CH	N/A	PowerDNS 4.8	Set strong randomly generated `api-key` (32+ chars) in `pdns.conf`
m4-059	CH	N/A	Unbound 1.21	Restrict recursion: set `access-control: allow_recursion` to LAN only
Application and web-API surfaces				
m4-060	DP	CVE-2023-51764	Postfix 3.7	Add `smtpd_forbid_bare_newline=yes` in `main.cf`
m4-061	DP	CVE-2023-51766	Exim 4.96	Disable CHUNKING advertisement: set `chunking_advertise_hosts=`
m4-062	CC	CVE-2019-11500	Dovecot 2.3.7	Require SSL/TLS for all connections; set `ssl_required=yes` in conf
m4-063	CC	CVE-2023-5631	Roundcube 1.6.3	Add `Content-Security-Policy` header blocking inline scripts
m4-064	CC	N/A	Modbus/TCP	Apply iptables allow-list restricting port 502 to authorized hosts
m4-065	CC	N/A	S7comm	Apply iptables restricting port 102 to engineering workstations only
m4-066	CC	N/A	IPMI BMC	Apply iptables restricting UDP 623 to management VLAN; disable Cipher 0
m4-067	CC	N/A	BACnet/IP	Apply iptables restricting UDP 47808 to BMS network segment

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
m4-068	CH	N/A	OPC-UA	Remove `SecurityPolicy=None`; require Sign+Encrypt security mode
m4-069	CH	N/A	Docker	Remove `-cap-add SYS_PTRACE`; drop container to minimal capabilities
m4-070	DP	CVE-2022-0492	Docker	Apply default seccomp profile; remove `-security-opt seccomp=unconfined`
m4-071	CH	N/A	Docker	Remove `-security-opt seccomp=unconfined`; apply Docker default seccomp
m4-072	CH	N/A	Docker	Remove `-security-opt apparmor=unconfined`; load default AppArmor profile
m4-073	CH	N/A	Docker Registry	Configure htpasswd authentication; set `REGISTRY_AUTH=htpasswd`
m4-074	CH	N/A	Docker image	Remove ENV secret instructions from Dockerfile; use Docker secrets
m4-075	CH	N/A	K8s RBAC	Delete over-privileged `ClusterRoleBinding`; apply least-privilege RBAC
m4-076	CH	N/A	K8s networking	Apply default-deny `NetworkPolicy` to target namespace
m4-077	CH	N/A	K8s resources	Apply `LimitRange` to set default CPU/memory limits on pods
m4-078	CH	N/A	K8s service	Change Service type from `NodePort` to `ClusterIP`
m4-079	CH	N/A	K8s etcd	Create `EncryptionConfiguration` with AES-CBC key; restart kube-apiserver
m4-080	CH	N/A	AWS IAM	Delete over-permissive policy version; apply least-privilege IAM
m4-081	CH	N/A	AWS SSM	Delete plaintext parameters; recreate as `SecureString` with KMS key
m4-082	CH	N/A	AWS S3	Remove public bucket policy or restrict to specific IAM principals
m4-083	CH	N/A	AWS SNS	Update topic policy to restrict `SNS:Subscribe` to specific principals
m4-084	CH	N/A	AWS Lambda/IAM	Replace inline policy; restrict `iam:PassRole` to specific role ARN
m4-085	CH	N/A	HashiCorp Vault	Remove `-dev` flag; write production `vault.hcl` with persistent backend
m4-086	CH	N/A	HashiCorp Vault	Add seal stanza for Transit auto-unseal or cloud KMS in `vault.hcl`
m4-087	CH	N/A	HashiCorp Consul	Set `acl{ enabled=true, default_policy="deny"}` in `consul.hcl`
m4-088	CH	N/A	Docker image	Remove `ENV DB_PASSWORD` from Dockerfile; pass secret at runtime
m4-089	CH	N/A	Docker image	Remove `ARG SECRET_TOKEN`; avoid writing secrets to image layers
m4-090	CH	N/A	Git repository	Use git filter-repo to rewrite history; rotate all leaked credentials
m4-091	CC	CVE-2021-43798	Grafana 8.3.0	Place nginx reverse proxy blocking `..` in `/public/plugins/` paths

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
m4-092	CH	N/A	Prometheus 2.40	Create `web.yml` with HTTP basic auth; start with `-web.config.file`
m4-093	CH	N/A	Jupyter Notebook	Set strong random token in `jupyter_notebook_config.py`
m4-094	DP	CVE-2023-27524	Apache Superset	Generate strong random `SECRET_KEY`; set in `superset_config.py`
m4-095	DP	CVE-2020-17526	Apache Airflow	Generate strong random `secret_key` in `airflow.cfg` webserver section
m4-096	DP	CVE-2020-11978	Apache Airflow	Set `load_examples=False` in `airflow.cfg`; remove example DAGs
m4-097	CC	CVE-2019-7609	Kibana 6.5.4	Disable Timelion plugin: add `timelion.enabled: false` in `kibana.yml`
m4-098	CH	N/A	Apache Druid	Set `druid.javascript.enabled=false` in coordinator `runtime.properties`
m4-099	CH	N/A	Hadoop YARN	Enable ACL enforcement in `yarn-site.xml`: set `yarn.acl.enable=true`
m4-100	CC	CVE-2020-11651	SaltStack 3000	Block ports 4505/4506 to untrusted hosts via iptables; upgrade when feasible
m4-101	CC	CVE-2021-29441	Nacos 1.4.0	Deploy nginx proxy overwriting User-Agent to non-Nacos-Server value
m4-102	CC	CVE-2022-23131	Zabbix 5.0	Change default admin password; place nginx reverse proxy with auth
m4-103	CH	CVE-2024-47177	cups-browsed	Stop and disable cups-browsed; mask the systemd unit
m4-104	CC	CVE-2023-38646	Metabase 0.46.6	Switch DB backend from H2 to PostgreSQL; remove H2 JDBC driver access
m4-105	CC	CVE-2025-29927	Next.js	Deploy nginx reverse proxy stripping `x-middleware-subrequest` header
m4-106	DP	CVE-2020-13945	Apache APISIX	Change default admin API key to strong random secret in `config.yaml`
m4-107	CC	CVE-2025-32433	Erlang/OTP SSH	Block port at firewall; restrict to public-key auth and allowed users
m4-108	CC	CVE-2025-3248	Langflow	Block `/api/v1/validate/code` at nginx; enforce auth on API
m4-109	CC	CVE-2020-14882	WebLogic Console	Deploy nginx blocking `/console` paths; normalize URL encoding
m4-110	DP	CVE-2023-34152	ImageMagick	Enforce restrictive `policy.xml`; block dangerous delegates (eps, mvg)
m4-111	NS	CVE-2018-1000115	Memcached 1.6	Start Memcached with `-U 0` to disable the UDP listener
m4-112	CC	CVE-2021-41773	Apache httpd 2.4.49	Install mod_security with OWASP CRS rules; block path-traversal patterns
m4-113	CC	CVE-2022-26134	Confluence (sim.)	Configure nginx reverse proxy to localhost:8090; block `/rest/api/latest`

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
m4-114	CC	CVE-2022-46169	Cacti $\leq$1.2.22	Configure nginx reverse proxy to localhost:8080; block `remote_agent.php`
m4-115	DP	CVE-2019-0193	Apache Solr 8.8	Disable DataImportHandler; remove `/dataimport` requestHandler from config
m4-116	CH	N/A	rsync daemon	Add `auth users` to each module; set `secrets file`; disable anon access
m4-117	CC	CVE-2026-31431	Linux kernel	Upgrade kernel to fix AF_ALG AEAD page-cache overwrite (Copy Fail)
Active Directory VM scenarios (advm-01–advm-20)				
advm-01	AC	CVE-2020-1472	Netlogon/DC	Set `FullSecureChannelProtection=1`; restart Netlogon (Zerologon)
advm-02	AC	CVE-2021-42278	Active Directory	Lower `MachineAccountQuota` to 0; apply KB5007247+ patches
advm-03	CC	N/A	Active Directory	Set strong password on svc_sql; restrict Kerberos to AES encryption
advm-04	CC	N/A	Active Directory	Disable `DONT_REQ_PREAUTH` on account; set strong Kerberos password
advm-05	CC	N/A	Active Directory	Remove unconstrained delegation from computer account
advm-06	AC	N/A	Active Directory	Remove DCSync rights (*Replicating Directory Changes*) from non-admin
advm-07	CH	N/A	ADCS	Remove `ENROLLEE_SUPPLIES_SUBJECT` flag from certificate template
advm-08	CH	N/A	ADCS	Remove Any Purpose / no-EKU certificate template from enrollment
advm-09	CH	N/A	ADCS	Remove Enrollment Agent capability from vulnerable template
advm-10	CH	N/A	ADCS	Disable `EDITF_ATTRIBUTESUBJECTALTNAME2` flag on CA policy module
advm-11	CH	N/A	ADCS	Disable NTLM on IIS Web Enrollment; require HTTPS + Kerberos
advm-12	CC	N/A	Active Directory	Require LDAP signing: set `LDAPServerIntegrity=2` in DC registry
advm-13	CC	N/A	Active Directory	Require SMB signing: set `RequireSecuritySignature=1` for LanmanServer
advm-14	CC	N/A	Active Directory	Set `LmCompatibilityLevel=5` to require NTLMv2; disable NTLMv1/LM
advm-15	NS	N/A	Active Directory	Disable LLMNR via Group Policy; disable NetBIOS over TCP/IP
advm-16	DP	CVE-2021-34527	Print spooler	Disable Print Spooler on DC; apply KB5005010 (PrintNightmare)
advm-17	CH	CVE-2021-36942	Active Directory	Apply ADV210003 mitigation; disable unneeded RPC interfaces
advm-18	AC	CVE-2014-1812	Group Policy	Remove cpassword XML from SYSVOL; rotate exposed account passwords
advm-19	AC	N/A	Active Directory	Deploy Microsoft LAPS; rotate local Administrator password on member hosts

continued on next page

Table 10 continued

ID	Cat	CVE	Service	Canonical remediation
advm-20	AC	N/A	Active Directory	Remove unauthorized ACE from AdminSDHolder; trigger SDProp to propagate

C.7 `hivestorm` (16 scenarios)

Hivestorm scenarios are free-roam: a single image plants several artifacts of mixed type, scored with weighted partial credit. Planted artifact identities (account names, binary paths, cron schedules, SUID filenames, registry key names) are randomized at build time from a per-scenario `roles.json`. The table below lists the 16 scenarios by platform and primary artifact mix.

Table 11: Per-scenario index for the `hivestorm` suite. Artifact mix codes: **B** backdoor account, **T** trojan path or malicious binary, **C** rogue cron entry, **S** SUID plant, **R** registry persistence (Windows), **D** domain object abuse (Active Directory).

ID	Platform	Artifact mix	Notes
hs-01	Debian 9	B, T, C, S	Rogue UID-0 account, SUID plant, trojan binary, reverse-shell cron
hs-02	Ubuntu 16.04	B, T, C, S	Samba share; rogue SSH authorized key; four planted artifact classes
hs-03	Windows 10	B, R, T	Rogue admin; HKLM Run autorun; trojan binary; scheduled-task persistence
hs-04	WinServer 2019	B, R	Rogue admin; SMB signing off; NTLMv1 regression; WDigest credential caching
hs-05	WinServer 2016	B, R, T	Rogue admin; SMBv1 re-enabled; LLMNR on; trojan binary; rogue Backup Operator
hs-06	Debian 9 (DB)	B, T	PostgreSQL; rogue DB user; Perl backdoor script planted under `/usr/local`
hs-07	Ubuntu 18.04	B, T	Samba; hidden user account; Perl backdoor script; rogue share
hs-08	WinServer (IIS)	B, R, T	IIS + PHP; rogue admin; scheduled-task persistence; cryptominer trojan
hs-09	Ubuntu (Nginx)	B, T, C	Nginx + phpBB; hidden user; socat trojan; rogue cron; prohibited package
hs-10	Ubuntu (PAM)	B	PAM faillock; hidden user; weak admin password; prohibited package planted
hs-11	WinServer (DC sim.)	B, R	Registry-planted DC misconfigs: LDAP signing, Kerberos, NTLMv1, WDigest
hs-12	CentOS 7 (LAMP)	B, T, C, S	LAMP; extra UID-0 user; trojan binary; rogue cron; SUID plant; rogue yum repo
hs-13	AD DC 2019	B, D, R	Real AD DS; rogue Domain Admin; Kerberoastable service; unconstrained delegation
hs-14	FreeBSD 13	B, T, S	Backdoor user; periodic drop; KLD kernel module; trojan; rogue pkg
hs-15	Docker host	B, T	Ubuntu Docker host; backdoor user; rogue container with sock-mount escape
hs-16	Ubuntu (Nginx+PHP-FPM)	B, T, C	PHP-FPM; backdoor user; webshell; trojan binary; rogue cron listener

D Additional experiments

D.1 Naive heuristic baseline

Heuristic. We run, on each scenario, the script `unattended-upgrades --minimal-upgrade-steps` followed by `systemctl daemon-reload` and a sweep of `systemctl restart` across every running service. No configuration files are edited, no permissions are changed, no accounts are added or removed, no firewall rules are added, no services are disabled, and no kernel reboots are performed. We then invoke the scenario's standard `verify.sh` and record exit 0 as a pass.

Suite	PASS	Total	Floor (%)
`ccdc/`	0	50	0.00
`meta2/`	0	40	0.00
`vulnhub/`	1	30	3.33
`meta3/ubuntu/`	0	19	0.00
`meta3/windows/`	0	21	0.00
`meta4/` (Docker)	1	117	0.85
`meta4/ad-vm/`	0	20	0.00
`hivestorm/`	0	16	0.00
Overall	**2**	**313**	**0.64**

Table 12: Naive baseline (`unattended-upgrades` + service restart) pass rate per suite. Two scenarios pass: one is a Shellshock rebuild on Debian 11 whose stock `bash` ships already patched (`vulnhub/scenario-20`); the other is regreSSHion (`meta4/scenario-11`) where the unheld `openssh-server` package has a bookworm-security backport.

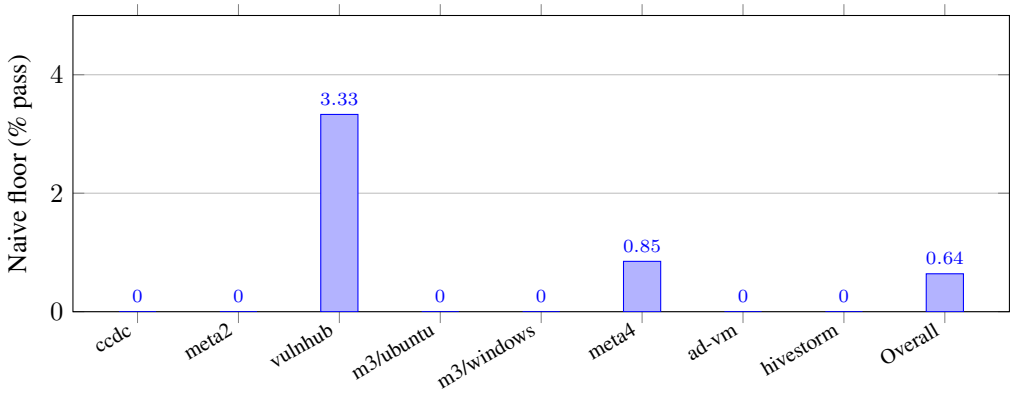

Figure 3: Naive heuristic baseline pass rate per suite. The floor is $\leq 1\%$ across every suite, confirming that the benchmark cannot be solved by blanket package upgrades and service restarts; meaningful progress requires the per-scenario configuration, permission, account, firewall, and compensating-control work that distinguishes a real remediation agent.

Per-suite results. Table 12 reports the naive baseline pass rate per suite; Figure 3 plots the same data. The heuristic establishes a floor that any LLM agent must clear to demonstrate non-trivial remediation behaviour.

Why the floor is so low. Across all 313 scenarios, 311 fall outside what blanket package upgrades can remediate. The dominant failure modes are: (i) configuration hardening (`sshd_config`, `nginx.conf`, `my.cnf`, `pg_hba.conf`, `php.ini`, sysctl); (ii) access control (`chmod`/`chown`, sudoers, account/group management, backdoor account removal); (iii) explicit *Compensating Controls* scenarios that forbid upgrade by construction (firewall scoping, loopback bind, WAF rule, removing an unsafe directive); (iv) vendored or source-built components (`log4j` JARs, GitLab/Jenkins/Atlassian images, Drupal/WordPress trees) that the distro package manager does not own; (v) kernel-coupled scenarios where a container cannot upgrade its host kernel (`meta4` S19/S21/S22/S117); and (vi) Windows targets (all of `meta3/windows/`, `meta4/ad-vm/`, and several `hivestorm/` scenarios) on which `apt` / `unattended-upgrades` / `systemctl` simply do not exist and the heuristic is a no-op.

E Host requirements

`meta2/`. Linux host only. Requires a kernel booted with `vsyscall=emulate`, because the Ubuntu 8.04 base image depends on the legacy `vsyscall` page, which was removed from upstream Linux in 5.18 and is disabled in the Docker Desktop and WSL2 kernels on Windows and macOS. Scenarios `scenario-37` and `scenario-39` require `-cap-add=NET_ADMIN`.

`meta3/windows/`. Windows 10 or 11 Professional or Enterprise, or Windows Server 2019 or later, with Docker Desktop in Windows Containers mode or a native `dockerd`. Approximately 40 GB free disk. Hyper-V isolation is the default for mixed build numbers.

`meta4/kernel-vm/`. Scenarios S21 and S22 ship a Vagrant virtual machine with a pinned kernel; replacing the kernel invalidates the proof-of-concept.

F `verify.sh` and JSON Lines scorer anatomy

F.1 Binary scorer (`verify_sh_scorer`)

Every non-Hivestorm scenario ships a `verify.sh` with two mandatory test phases: a *PoC check* that confirms the vulnerability is no longer exploitable, and a *regression check* that confirms the affected service is still running. Exit 0 only if both pass.

SSH empty-password login (CCDC scenario-02). The script calls `sshd -T` for the effective post-`Match` configuration, then performs a live login attempt via `sshpass` to catch configs that look correct on disk but are overridden at runtime. The regression phase repeats the probe with a valid credential to ensure the daemon was not simply disabled.

Exposed installer pages (meta3 scenario-05). `curl` drives both phases: `/drupal/install.php` and `/phpmyadmin/setup/index.php` must return 403, while the application homepages must remain 200. A blanket Apache shutdown satisfies the PoC but fails the regression.

Apache path-traversal CVE (meta4 scenario-09). Two remediation paths are accepted: upgrading `httpd` to $\geq$2.4.51, or blocking the double-encoded traversal string at the WAF/rewrite layer. The script resolves version via `httpd -v`, falls through to a live exploit probe if the version is too old, and finally retries the root endpoint up to six times before declaring the service broken.

Exim4 open relay (VulnHub scenario-15). If `dc_local_interfaces` still contains `0.0.0.0`, the script accepts an `iptables DROP` rule on port 25 as an equivalent fix, avoiding a false failure for agents that mitigate at the firewall layer rather than reconfiguring Exim.

F.2 Weighted scorer (`hivestorm_weighted_scorer`)

Hivestorm `verify.sh` scripts emit one JSON object per check on stdout:

```
{"check":"<id>","weight":<int>,"pass":<bool>,"reason":"<str>"}
```

Service-survival probes additionally carry `"category":"service"` and `"weight":0`. The scorer computes

$$S = \frac{\max(0,\ R - 10F_{\text{svc}})}{T},$$

where R is the sum of weights of passing non-service checks, T is the total non-service weight, and F_{svc} is the count of failing service probes. This penalises agents that disable services to prevent planted processes from respawning.

The planted artifacts fall into four classes.

Class 1: Rogue accounts. Each scenario plants a backdoor user with a `sudoers` drop-in; scenario-01 also plants an extra UID-0 account. Checks are layered: user-database lookup (`getent passwd`), live SSH login probe via `sshpass`, and a `sudo -n` behavioural test. All three must clear before the check emits `"pass":true`.

Class 2: Persistence mechanisms. Scenario-01 plants a cron drop-in and scans for the beacon IP across all cron directories, catching agents that move rather than delete the entry. Scenario-14 (FreeBSD) plants a `periodic` script and a `loader.conf` KLD entry; each is checked by file existence and a `grep` on the relevant config.

Class 3: Trojanized and SUID binaries. The trojan check is behavioural rather than hash-based: the script invokes the suspicious binary with `-h` and matches the `netcat` help output, so a clean replacement binary passes even if its inode differs from the original. The SUID-plant check combines a mode-bit test (`-u $file`) with an effective-UID probe executed as the legitimate admin account.

Class 4: Covert listeners and webshells. Listener checks use both the socket-table (`ss`/`sockstat`) and a direct TCP-connect probe (`/dev/tcp`), handling the race between a killed process and a supervising unit that immediately respawns it. Scenario-14 adds an nginx webshell check (`shell.php` in the document root). Service-survival probesfor SSH and the web server carry `"category":"service"` and appear at the end of every Hivestorm script; a failure there subtracts 10 points from the raw score regardless of how many other checks passed.

G Datasheet

We provide a datasheet for SysRepair-Bench following the framework of Gebru et al. [3]. We adopt the conventional question-and-answer format and group questions under the seven canonical headings.

G.1 Motivation

For what purpose was the dataset created? SysRepair-Bench was created to evaluate AI agents on the task of remediating real-world system vulnerabilities while preserving service availability. This task is the defensive counterpart of offensive security benchmarks such as Cybench [26], NYU CTF Bench [18], and CVE-Bench [28], and it is distinct from source-level program repair benchmarks such as SWE-bench [6]. To our knowledge, no prior public benchmark evaluates the joint constraint of closing a vulnerability on a running system and not breaking the service the vulnerability protects, which is the dual constraint under which production incident responders work.

Who created the dataset and on behalf of which entity? Anonymized for review

Who funded the creation of the dataset? Anonymized for review

Any other comments? Anonymized for review

G.2 Composition

What do the instances represent? Each instance represents a vulnerable system, materialized as a Dockerized container or a Vagrant virtual machine, together with a machine-readable threat description and a deterministic verification harness. Container scenarios are reproducible via `docker build` from a pinned `Dockerfile`; kernel-coupled scenarios in `meta4` ship a Vagrant virtual machine with a pinned kernel.

How many instances are there in total? SysRepair-Bench contains 313 scenarios in the released distribution: 297 binary-scored scenarios across the `ccdc`, `meta2`, `vulnhub`, `meta3/ubuntu`, `meta3/windows`, and `meta4` suites, and 16 partial-credit scenarios in the `hivestorm` suite. The full per-suite breakdown is given in Table 1 and the per-scenario index in Appendix C.

Does the dataset contain all possible instances or is it a sample of instances from a larger set? The dataset is a sample. The `ccdc`, `meta2`, `vulnhub`, and `meta3` suites cover documented findings in their respective source materials, decomposed at the per-vulnerability granularity and de-duplicated where the source flagged the same underlying issue under multiple identifiers. The `meta4` suite is deliberately a curated sample of modern Common Vulnerabilities and Exposures rather than an exhaustive catalogue, with selection priorities described in Section 4: alignment with OWASP Top 10 [13], the MITRE CWE Top 25 [9], and the CISA Known Exploited Vulnerabilities catalog [2], weighted toward high-impact and operationally common classes (language and library, kernel and local privilege escalation, container and cloud-on-localhost, application and web-API surfaces). The `hivestorm` suite mirrors the platform mix of the source competition.

What data does each instance consist of? Binary-suite scenarios are directories with three files: a `Dockerfile` that builds the vulnerable container deterministically; a `threat.md` file containing severity, CVSS v3.1 score where applicable, Common Vulnerabilities and Exposures identifier where one exists, the affected service, and the canonical remediation steps; and a `verify.sh` script returning exit code 0 if the vulnerability is closed and the service still works and exit code 1 otherwise. Hivestorm scenarios replace `verify.sh` with a JSON Lines scorer that emits one weighted check per planted artifact and a per-build `roles.json` that randomizes artifact identities.

Is there a label or target associated with each instance? Yes. For the 297 binary scenarios the label is the conjunction of three deterministic checks performed by `verify.sh`: the proof-of-concept no longer succeeds, every previously operational service remains operational, and (for the 39 compensating-control scenarios) the required mitigation is in place. For the 16 Hivestorm scenarios the label is a continuous score in $[0, 1]$ obtained by summing weighted per-artifact checks normalized to the unit interval.

Is any information missing from individual instances? A subset of `ccdc` scenarios document misconfigurations rather than disclosed Common Vulnerabilities and Exposures, and therefore have no CVE identifier in `threat.md`. Where this is the case the `CVE` field is set to `N/A` and the threat description is written from the upstream hardening guidance.

Are relationships between individual instances made explicit? Yes, through suite membership, remediation category (Access Control, Configuration Hardening, Dependency and Package Management, Network Security and Firewall Policy, Compensating Controls), affected service or application type, and CVSS severity bucket. These attributes appear in `threat.md` and are summarized in Table 1 and Appendix A.

Are there recommended data splits? SysRepair-Bench is an evaluation-only benchmark, so there is no train, validation, or test split in the supervised-learning sense. Two evaluation conditions are defined: black-box (only the running container is exposed) and report-informed (the `threat.md` description is provided as context).

Are there any errors, sources of noise, or redundancies in the dataset? We are not aware of any errata at submission time. Errata, when discovered, will be tracked in `CHANGELOG.md` and addressed in versioned releases (see the maintenance subsection below). The Hivestorm randomization is an intentional source of stochasticity that varies artifact identities at build time without changing the underlying remediation difficulty.

Is the dataset self-contained, or does it link to or rely on external resources? Each scenario builds locally from a self-contained `Dockerfile` or Vagrantfile. Build artifacts pull from upstream Docker Hub and Linux distribution repositories at build time, and where the underlying vulnerability is tied to a specific upstream version (for example `httpd:2.4.49` for CVE-2021-41773) the scenario pins to that version. Pins are listed in the per-suite manifests in Appendix C. We recommend that users mirror upstream images locally before evaluation runs to guard against upstream image deletion or repository unavailability.

Does the dataset contain data that might be considered confidential? No. All scenarios are constructed from public material: published competition hardening checklists, the public OpenVAS scan of Metasploitable 2, public VulnHub images, the public Rapid7 Metasploitable 3 cookbook, public Common Vulnerabilities and Exposures records, and the public Hivestorm scoring methodology.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? Each scenario contains exploitation primitives by construction, since remediation requires a verifiable exploit to fail. These primitives are identical in nature to those that appear in offensive security benchmarks and the public Common Vulnerabilities and Exposures advisory record. They are embedded in isolated containers or virtual machines and are framed in a defensive context. The dataset does not contain content directed at any person or group.

Does the dataset relate to people? No. Scenarios target software artifacts: services, packages, kernel features, configuration files, and accounts on isolated systems. Account names thatappear in scenarios (for example `root`, `vagrant`, `admin`) are conventional system identifiers, not references to real individuals.

Does the dataset identify any subpopulations, or contain data that might allow identification of individuals, or contain sensitive data? No. Not applicable.

G.3 Collection process

How was the data associated with each instance acquired? Each suite was constructed by manual scenario authoring, drawing on the following public sources, listed by suite. The `ccdc` suite decomposes published Collegiate Cyber Defense Competition blue-team hardening materials [10] into per-vulnerability containers. The `meta2` suite is seeded by the OpenVAS scan of the canonical Metasploitable 2 image [16], with the scan output included for auditability under `openvas-scan-reports/`. The `vulnhub` suite is built from per-virtual-machine analysis of fourteen community VulnHub images [21]. The `meta3/ubuntu` and `meta3/windows` suites vendor portions of the Rapid7 Metasploitable 3 Chef cookbook [17] under its BSD-3-Clause license, with full acknowledgment in `meta3/ubuntu/shared/UPSTREAM_LICENSE`. The `meta4` suite was authored by us against public Common Vulnerabilities and Exposures advisories, vendor security bulletins, and proof-of-concept disclosures. The `hivestorm` scoring methodology follows the public Hivestorm scoring engine [5]; planted artifacts and randomization logic were authored by us.

What mechanisms or procedures were used to collect the data? Scenarios were authored, built with Docker or Vagrant, and validated by the authors through a two-stage check: a positive check that the scenario reproduces the documented exploit on the unhardened image, and a negative check that `verify.sh` returns exit 0 after the canonical remediation in `threat.md` is applied. Findings from the OpenVAS scan of Metasploitable 2 were re-run against our reproduced container to confirm fidelity to the upstream image.

If the dataset is a sample from a larger set, what was the sampling strategy? For `ccdc`, `meta2`, and `meta3`, the strategy is exhaustive coverage of distinct findings in the source material with de-duplication. For `vulnhub`, the strategy is per-image decomposition, with each VulnHub image contributing between one and

four scenarios depending on the number of independent vulnerabilities it exposes. For `meta4`, the strategy is purposive sampling along the four coverage axes described in Section 4, prioritizing high-impact, operationally common, and post-cutoff Common Vulnerabilities and Exposures. For `hivestorm`, the strategy is platform-mix parity with the source competition, covering Linux distributions (Debian, Ubuntu, CentOS, FreeBSD), Windows Server Core, and Active Directory.

Who was involved in the data collection process? Anonymized for review

Over what timeframe was the data collected? Anonymized for review

Were any ethical review processes conducted? SysRepair-Bench contains no human-subjects data, no personally identifiable information, and no surveillance or behavioral data; institutional review board review is not applicable. Dual-use considerations are discussed in Section 8: the benchmark contains no novel exploit code, every scenario builds from material that already exists in indexed public form, all scenarios run in isolation, and the benchmark rewards closing vulnerabilities while penalizing service disruption.

G.4 Preprocessing, cleaning, and labeling

Was any preprocessing, cleaning, or labeling of the data done? Yes. Preprocessing differs by suite. For `ccdc`, the source hardening checklists, which are written for full-system competition images, were decomposed into per-vulnerability container scenarios so that each scenario isolates a single misconfiguration; this prevents cross-contamination of remediation actions across scenarios. For `meta2`, the OpenVAS scan output was de-duplicated by service and finding type, then each finding was rebuilt as a per-vulnerability container reproducing the issue. For `vulnhub`, each upstream image was analyzed for distinct vulnerabilities, each of which was isolated into its own container. For `meta3`, scenarios were ported from the upstream Rapid7 Chef cookbook and re-keyed to the `Dockerfile` plus `threat.md` plus `verify.sh` format. For `meta4`, scenarios were authored from CVE disclosures and vendor advisories. All scenarios were labeled with a remediation category (Access Control, Configuration Hardening, Dependency and Package Management, Network Security and Firewall Policy, Compensating Controls) and a CVSS severity bucket where the underlying CVE provides one.

Was the raw data saved in addition to the preprocessed data? Yes. The OpenVAS scan reports that drove the `meta2` and `meta3/ubuntu` scenario scopes are included in `openvas-scan-reports/` for auditability. The upstream Rapid7 Chef cookbook is preserved alongside its license in `meta3/ubuntu/shared/UPSTREAM_LICENSE`.

Is the software that was used to preprocess, clean, or label the data available? Yes. Per-suite build scripts, the Inspect AI [20] harness, and the Hivestorm scorer reference implementation are distributed with the benchmark.

G.5 Uses

Has the dataset been used for any tasks already? Yes. The baseline experiments reported in Section 7 of this paper are the first use, with two families covering three models (MiniMax-M2.7, Qwen3.5-35B-A3B, and Qwen3.5-9B) under a ReAct [25] solver in black-box and report-informed conditions.

Is there a repository that links to any or all papers or systems that use the dataset? Anonymized for review

What other tasks could the dataset be used for? The dataset is suited to several downstream tasks beyond agent evaluation. (a) Curriculum design for defensive agent training, where scenarios provide graded difficulty across remediation categories. (b) Studies of agent scaffolding, where the harness allows controlled substitution of solvers, prompts, and tool affordances. (c) Studies of regression risk in automated remediation, where the dual-objective scoring isolates the service-availability constraint as a first-class evaluation axis.

Is there anything about the composition of the dataset or the way it was collected and preprocessed that might impact future uses? Two considerations are material. First, public security material, including the source corpora that seed `ccdc`, `meta2`, `vulnhub`, and `meta3`, is highly indexed and has likely appeared in the pretraining data of large models. Second, scenarios reproduce vulnerabilities by construction and include exploit primitives. Researchers using the benchmark to train defensive agents should be aware that the exploit primitives are also present in the training signal.

Are there tasks for which the dataset should not be used? SysRepair-Bench should not be used as a training corpus for the same agent that is then evaluated on it. It should not be used to evaluate offensive capability, because while the underlying scenarios overlap with offensive benchmarks, the scoring rewards remediation rather than exploitation. It should not be used to evaluate live cloud remediation, since scenarios run in isolation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 enumerate six concrete contributions (313 binary-scored scenarios across six suites, dual-objective scoring extended by compensating-control adequacy, the Meta4 modern-CVE suite, the Hivestorm partial-credit and randomization track, an Inspect AI harness, and baselines on four models in zero-knowledge and one-day variants with two contamination ablations), each delivered by a corresponding section (§3–§7).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 8 contains explicit *Scope*, *Contamination*, and *Limitations* paragraphs disclosing Linux skew, the minority share of Windows and Active Directory scenarios, and the deliberate exclusion of source-code modification, zero-days without remediation, hardware/firmware issues, and live cloud-IAM evaluation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper presents a benchmark and empirical baselines and does not state or rely on formal theorems.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Each scenario builds deterministically from a versioned `Dockerfile` or `Vagrantfile` (§3); the harness configuration (per-scenario time limit 1800 s, per-command `bash` timeout 180 s, verification timeout 3000 s, `bash`-only tool surface) is specified in §6; models, seeds, solver, and the two knowledge variants are listed in §7; and host requirements (e.g., `vsyscall=emulate` for `meta2/` and the pinned-kernel Vagrant VM for `meta4/kernel-vm/`) are documented in Appendix E.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Scenarios, scan reports, the grading harness, and the evaluation code are released under an open license at the anonymous URL given in the abstract (`https://anonymous.4open.science/r/sysrepair-bench-425F/`); per-suite license details and the BSD-3-Clause acknowledgment for the vendored Rapid7 Chef cookbook will be finalized in Appendix.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: No model training is performed; baselines use off-the-shelf checkpoints (MiniMax 2.7, Qwen3.6-35B-A3B, Qwen3.5-9B) under a single ReAct scaffold, with inference settings, timeouts, and the tool surface given in §6, the zero-knowledge versus one-day prompt construction described in the same section.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report success@1 and success@5 over 5 random seeds (§7, following the protocol of Zhu et al. [28]) and report the mean with standard error for the Hivestorm partial-credit track in Tables 1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Host hardware and OS requirements for the harness (Docker, Vagrant, the `vsyscall=emulate` kernel option for `meta2/`, Windows-Containers configuration for `meta3/windows/`, and the pinned-kernel VM for `meta4/kernel-vm/`) are documented in Appendix E, and per-scenario wall-clock and token-cost statistics are reported in Table 2; model inference for the open-weight baselines was served via vendor APIs and required no local accelerators.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The benchmark uses publicly available red-team materials and intentionally vulnerable artifacts that already exist in indexed form (§8, *Dual-use*); no human subjects, no personal data, and no novel exploit code are introduced, and all execution is sandboxed in containers or virtual machines with no default network exposure.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 8 (*Dual-use*) frames SysRepair-Bench as a defensive counterweight to existing offensive-security benchmarks (rewarding closure of vulnerabilities and preserving service availability while penalizing destructive collateral) and addresses the negative-impact surface, namely that better defensive evaluation could indirectly inform attackers, by noting that all underlying vulnerability material is already public and that the benchmark contains no novel exploit code.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: Scenarios run in isolated Docker containers or Vagrant VMs with no default network exposure (§8), the grading harness is separated from agent code so that leaderboard submissions cannot tamper with scoring, and the per-build identity randomization on Hivestorm doubles as a misuse-resistance feature because memorized artifact names confer no advantage and the suite cannot be repurposed as a fixed exploitation recipe.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Upstream sources (Metasploitable 2, Metasploitable 3, VulnHub, the CCDC hardening materials, and the Hivestorm scoring engine) are cited in §2 and §3; the BSD-3-Clause acknowledgment for the vendored Rapid7 Chef cookbook.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Two new assets are introduced, the Meta4 suite (§4) and the Hivestorm partial-credit track (§5), with each scenario shipping a `Dockerfile` (or `Vagrantfile`), a `threat.md` metadata file (CVSS, CVE, affected service, canonical remediation), and a `verify.sh` grader (or JSON Lines scorer for Hivestorm); per-suite indices, annotated `verify.sh` walkthroughs, and the full datasheet (per Gebru et al. [3]) are provided in Appendices C, F, and G.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: No crowdsourcing was used; the only human-in-the-loop step is the failure-trajectory annotation by two of the paper's authors (§7, Table 2), and no external participants were recruited or compensated.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work involves no human subjects, no participant recruitment, and no collection of personal data, so IRB review was not required.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are the central object of evaluation rather than an authoring aid; the models, scaffold, and inference configuration are disclosed in §7 (primary baseline MiniMax 2.7 paired with a ReAct solver, with comparison runs on DeepSeek V4 Pro, Qwen3.6-35B-A3B, and Qwen3.5-9B), and the rationale for ReAct as the chosen scaffold is given in §2.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.