International Journal of Information Security manuscript No.
(will be inserted by the editor)

Autonomous System Vulnerability Remediation: A Survey of Agentic AI, Reinforcement Learning, Benchmarks, and Operational Safety

Abanisenioluwa Kolawole Orojo · **Webster Chiedozie Elumelu** · **Emmanuelli El-Mahmoud** · **Erika Leal**

Received: date / Accepted: date

Abstract Autonomous vulnerability remediation is moving from static playbooks and human-led patch queues to agents that can reason about system state, select defensive actions, make changes, and verify that risk has been reduced without operational failures. This survey reviews this emerging area for enterprise, cloud, network, container, and operating-system environments. We screened a corpus of 470 records using PICOS criteria that emphasized system-level vulnerabilities, LLM or agentic AI interventions, remediation execution, comparative or empirical outcomes, and publication quality. We also explored research on LLM-based infrastructure-as-code repair, autonomous cyber-defense benchmarks, agentic SOAR, reinforcement-learning patch optimization, and real-system remediation benchmarks. As this field is emerging the evidence base remains small, but it is coherent enough to survey. We organize the literature into six streams: LLM-based configuration and IaC remediation; autonomous cyber-defense agents in cyber ranges; reinforcement-learning vulnerability and patch management; agentic SOAR and self-healing operations; traditional patch automation and human practice; and benchmark infrastructure. Across these streams, five recurring themes define the field: state acquisition, policy and knowledge grounding, planning, guarded execution, and post-action verification. We then formalize autonomous system repair as a closed-loop task model that separates observation, diagnosis, action synthesis, safety gating, execution, verification, rollback, and evidence capture. The main research gaps lie in model capabilities, safety envelopes, rollback semantics, benchmark realism, service regression testing, evaluation comparability, and accountability for actions that change production systems.

Keywords Autonomous cyber defense · Vulnerability remediation · Large language models · Multi-agent systems · Patch management · Security operations

Abanisenioluwa Kolawole Orojo
Department of Computer Science, Baylor University, Waco, TX, USA
E-mail: Abanisenioluwa_Orojo@baylor.edu (corresponding author)

Webster Chiedozie Elumelu
Georgia Institute of Technology, Atlanta, GA, USA
E-mail: welumelu3@gatech.edu

Emmanuelli El-Mahmoud
Department of Computer Science, Baylor University, Waco, TX, USA
E-mail: Emmanuelli_ElMahmoud@baylor.edu

Erika Leal
Department of Computer Science, Baylor University, Waco, TX, USA
E-mail: Erika_Leal@baylor.edu

1 Introduction

Vulnerability remediation is the final operational step after discovery, scoring, and prioritization [1]. It is the step where risk in cyber & change-management cross [2]. A patch can remediate a Common Vulnerabilities and Exposures (CVE), but it can also interrupt a service by, e.g., invalidating its dependencies [3]. Configuration hardening can reduce the attack surface for an ingress path, but cut off a legitimate workflow [4]. This tension explains why remediation remains slow even when detection is highly automated [5].

Large language models (LLMs), tool-using agents, multi-agent systems (MAS), reinforcement learning (RL), and security orchestration are evolving the research question. Instead of asking whether a scanner can identify a vulnerability, recent work asks whether agents can interpret system context, develop a defensive action plan,

execute it through administrative tools, and verify the result. This is different from source-code vulnerability repair. The target is not an isolated source function or benchmark repository; it is a running system, network, cloud configuration, container orchestrator, enterprise endpoint estate, or cyber-range representation of those environments.

This survey focuses on autonomous system vulnerability remediation: fully or semi-autonomous workflows that remediate system-level vulnerabilities, misconfigurations, missing patches, insecure services, unsafe permissions, or exposed infrastructure states. We use the term system broadly to include enterprise IT, cloud and container platforms, networks, operating systems, security operations centers (SOCs), and high-fidelity cyber ranges. We exclude source-code repair, vulnerability detection without action, and static playbooks without adaptive reasoning.

The core finding explores this young and fragmented field. A few studies now evaluate hybrid DRL/LLM autonomous cyber-defense agents in cyber-range settings [6,7,8]. LLM-based configuration repair has appeared for Kubernetes, containers, and infrastructure-as-code (IaC) misconfigurations [9,10,11,12,13,14]. Agentic SOAR work has begun to demonstrate LLM-driven investigation and response execution [15]. RL-based patch and vulnerability management studies optimize the choice and timing of mitigation under uncertainty [16,17,18]. Benchmark work such as CAGE / CybORG and SysRepair-Bench provides repeatable evaluation environments for blue-agent defense or system-remediation tasks [19,20,21]. Yet most candidate literature still falls outside the scope: it is detection-only, code-level program repair, generic LLM cybersecurity discussion, traditional automation, patent literature, product marketing, or broad operational guidance.

This survey contributes:

1. A precise definition and PICOS eligibility model for autonomous system vulnerability remediation.
2. A screened literature map separating core studies, comparators, and benchmark artifacts.
3. A formal task model for autonomous system repair, grounded in vulnerability-management, patch-management, incident-handling, and autonomic-computing literature.
4. A capability taxonomy covering state acquisition, grounding, planning, execution, and verification.
5. A synthesis of six evidence streams and the architectural patterns that connect them.
6. A benchmark and metric agenda that joins security success with service continuity and rollback.
7. A deployment-oriented safety analysis for agents that can change production systems.

2 Scope and Definitions

2.1 System-level remediation

System-level remediation concerns operational assets and configurations rather than source code alone. Examples include upgrading a vulnerable package, disabling an unsafe daemon, changing file permissions, modifying SSH or web-server configuration, adding firewall scoping, repairing Kubernetes or Terraform misconfigurations, rolling back a bad change, or restoring a compromised host. The target is a system state: the vulnerability should no longer be exploitable, while intended service behavior should remain available. Figure 1 provides a clear picture of this survey's scope.

This scope deliberately differs from automated program repair (APR). APR can be relevant when a system vulnerability is caused by application logic, but most APR benchmarks evaluate source-code patches through compilation or unit tests [22,23,24]. By contrast, system remediation requires administrative actions, environmental constraints, service health checks, and rollback semantics. SysRepair-Bench makes this distinction explicit by excluding source code modifications and scoring only system administration fixes that pass both proof-of-concept and regression checks [21].

2.2 Autonomy and agency

We treat a workflow as autonomous or agentic when it performs at least three of the following: interpret context, choose among candidate actions, use tools, execute or prepare deployable changes, monitor outcomes, revise a plan after failure, and explain its action trace. This operational test draws on the agent view that an autonomous system perceives its environment and acts on it over time to pursue goals, organized around the profiling, memory, planning, and action modules that recent surveys treat as the working architecture of an LLM-based agent [25]. It also tracks the cyber autonomy tradition, which formalizes self-healing and self-adaptive defense around a control loop that monitors the environment, analyzes signals, plans responses, and executes them with progressively reduced human oversight as the system earns trust [26,27]. Static automation that runs a predetermined playbook against a hardcoded trigger is therefore a comparator, not a core agentic system, since rule-based scripted responses sit explicitly at a lower tier of cyber autonomy maturity than systems that reason about context and select among candidate actions [27]. Human-in-the-loop systems are still included when the machine performs adaptive analysis and prepares or executes concrete remediation actions under approval

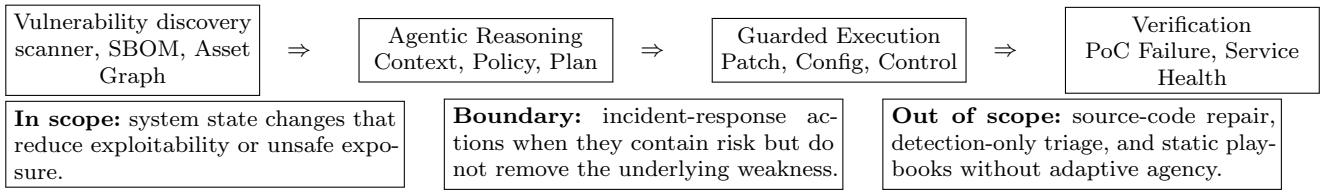

Fig. 1 Scope of autonomous system vulnerability remediation. The survey is centered on reasoning about a vulnerable system, changing it, and verifying that risk has been reduced without operational damage.

gates, since both lines of work treat human oversight as a waypoint along the maturity ladder rather than its negation [26,27].

2.3 Remediation versus response

While often conflated in security automation literature, remediation and incident response serve distinct operational goals. Incident response primarily focuses on containing or mitigating an active security event, prioritizing immediate triage and business continuity over eliminating the architectural weakness [28]. The response actions include blocking a malicious IP address, isolating a compromised host, or deploying a network decoy. Conversely, remediation is the removal of a vulnerability or unsafe system state to prevent initial or recurring exploitation [29]. Remediation addresses root causes through actions such as patching a software package, correcting a misconfigured Kubernetes manifest, or restructuring a privilege boundary. As [30] note in their review of automated incident-response solutions, proposed systems often describe their response actions in vague terms, making it difficult to identify the specific defensive technique being applied; we therefore include incident-response literature only when it supplies architectural frameworks or evaluation evidence directly applicable to autonomous remediation, ensuring that response-only studies do not dominate the corpus.

3 Formal Task Model for Autonomous System Repair

A clearer task formalization helps distinguish autonomous system repair from adjacent activities such as vulnerability detection, risk prioritization, incident containment, and automated source-code repair. We draw the formalization from four areas. Enterprise patch guidance defines patching as identifying, prioritizing, acquiring, installing, and verifying patches across an organization [1]; the NCCoE patching practice guide adds routine patching, emergency patching, emergency mitigation, isolation of unpatchable assets, and protection of patch-management infrastructure [31]. Current NIST incident-response guidance integrates incident response into cybersecurity risk management and emphasizes preparation, reduction of incident likelihood and impact, detection, response, recovery, and continuous improvement [32]. Vulnerability-management books separate asset and vulnerability information, scanning, risk analysis, recommendations, implementation, and organizational support [33,34]. Finally, autonomic and self-adaptive systems frame repair as a feedback loop over monitored state, analysis, planning, execution, and knowledge [26, 35,36].

3.1 Repair episode

We model one autonomous system-repair episode as a tuple:

$$E = \langle S, F, K, P, A, C, V_s, V_o, H \rangle \tag{1}$$

Here S is the observed system state; F is one or more vulnerability findings or unsafe states; K is security and operational knowledge; P is local policy; A is the available action set; C is the constraint set; V_s is the security verifier; V_o is the operational verifier; and H is the human or organizational approval interface. In distributed environments, S may be represented as an attributed dependency graph $G = (V, E)$ whose vertices carry attributes such as operating system, packages, known CVEs, privileges, and business criticality, and whose edges encode functional, network, or data-flow dependencies; this representation lets diagnosis (T2) and safety gating (T5) reason about the blast radius of a change before it is applied [18]. A successful repair produces a new state S' such that the unsafe condition is removed or acceptably mitigated, policy constraints are satisfied, intended service behavior remains acceptable, and evidence is recorded. In short:

$$V_s(S', F) = 1 \quad \wedge \quad V_o(S') = 1 \quad \wedge \quad P(S') = 1. \tag{2}$$

This definition is stricter than the recommendation. It also separates direct remediation from compensating control. A compensating control can be valid when direct patching is unsafe or impossible, but it should be represented as a time-bounded risk decision with monitoring and owner accountability.

3.2 Task families

Table 1 formalizes the task families that an autonomous system-repair agent must perform or coordinate. These tasks are not meant to imply a rigid pipeline; many systems will iterate, branch, or escalate. The table instead gives the field a common vocabulary for describing what is being automated.

3.3 Action vocabulary

The action set A should be typed. Patch actions install, remove, downgrade, or upgrade packages, firmware, images, or platform components. Configuration actions edit service, OS, network, identity, cloud, container, or IaC settings. Isolation actions change reachability, segmentation, access, or runtime placement. Compensating-control actions reduce exposure without removing the underlying vulnerability. Restore and rollback actions reverse a change or recover a known-good state. Evidence actions create tickets, explanations, proofs, and monitoring hooks. This vocabulary lets a paper state whether it evaluates a patch, a configuration repair, a containment response, a temporary compensating control, or a verified full repair.

3.4 Objective function

A repair policy should not optimize security in isolation. It should choose a plan π that reduces residual security risk while respecting service, cost, and governance constraints:

$$\min_{\pi} \; R(S', F) + \lambda_1 D(S, S') + \lambda_2 \mathrm{Cost}(\pi) + \lambda_3 \mathrm{Unc}(\pi) \quad (3)$$

subject to $P(S') = 1$, $V_o(S') = 1$, and the required approval state in H. Here R is residual risk, D is operational disruption, Cost is deployment or human cost, and Unc is uncertainty. These terms are deliberately abstract, so that different evidence streams can be read as instantiating different components; concrete operationalizations are nonetheless possible. Residual risk can be expressed as exploitability-weighted impact,

$$R(S', F) = \sum_{f \in F} \mathrm{EPSS}(f, t) \cdot \mathrm{CVSS}_{\mathrm{env}}(f, S'), \quad (4)$$

so that actively exploited weaknesses are not dominated by high-severity but unexploited findings [37,38]. Deployment cost can include socio-technical coordination overhead,

$$\mathrm{Cost}(\pi) = \sum_{a_i \in \pi} \big(C_{\mathrm{compute}}(a_i) + C_{\mathrm{downtime}}(a_i)\big) + \gamma \, N_{\mathrm{teams}}, \quad (5)$$

where N_{teams} is the number of organizational teams required to approve and coordinate the change and all terms share a common cost unit [5]. Uncertainty can be expressed as the complement of the plan's success probability,

$$\mathrm{Unc}(\pi) = 1 - \prod_{a_i \in \pi} P_{\mathrm{success}}(a_i), \quad (6)$$

under an independence approximation, so that high-uncertainty plans are routed to the approval interface H rather than executed autonomously [14].

3.5 Implications for evaluation

The task model elevates three evaluation requirements. First, studies should identify the highest task level they actually complete; a system that reaches T4 is not equivalent to one that reaches T7 or T9. Second, benchmark reports should preserve failure provenance: state failure, diagnosis failure, unsafe plan, denied gate, execution failure, security-verification failure, operational-regression failure, rollback failure, or evidence failure. Third, claims of autonomy should specify where adaptation occurs. Static execution of a known patch is automation; autonomous repair requires adaptive state reconstruction, action selection, replanning, or verification-driven recovery.

4 Related Survey Areas and Boundary

Adjacent surveys motivate this paper, but do not replace it. Broad LLM cybersecurity surveys catalog applications, vulnerabilities, and defense techniques across many tasks [39]. Agentic-security surveys examine agent workflows and agent-specific risks across offensive and

Table 1 Formal task families for autonomous system repair

Task	Name	Formal Object	Evaluation Question
T0	Intake	$F \leftarrow$ scanner report, alert, ticket, or policy report.	Is there a concrete unsafe state to repair?
T1	State reconstruction	$\hat{S} = g(S, F, K)$ from assets, packages, configs, logs, topology, and dependencies.	Did the agent gather enough state before acting?
T2	Diagnosis and scope	$U = d(\hat{S}, F, K)$ identifies root unsafe condition, affected assets, and blast radius.	Is the target condition actually remediable at system level?
T3	Repair-intent selection	$I \in \{\text{patch}, \text{configure}, \text{isolate}, \text{compensate}, \text{restore}, \text{defer}, \text{escalate}\}$.	Does the selected intent match risk, policy, and feasibility?
T4	Plan synthesis	$\pi = \langle a_1, \ldots, a_n, r \rangle$ where $a_i \in A$ and r is rollback.	Is the plan executable, minimal, reversible, and policy-compliant?
T5	Safety gating	$G(\pi, \hat{S}, C, H) \in \{\text{allow}, \text{approve}, \text{deny}\}$.	Were privileges, maintenance windows, canaries, and approvals respected?
T6	Execution	$S' = \text{apply}(\pi, S)$ through typed administrative tools.	Did tool use succeed without destructive side effects?
T7	Verification	$V_s(S', F)$ and $V_o(S')$ test security and service outcomes.	Did the exploit/scanner fail while service regression checks pass?
T8	Recovery and rollback	If V_s or V_o fails, apply r or a safe fallback.	Can the system return to an acceptable state after a bad repair?
T9	Evidence and learning	$B = \langle F, \hat{S}, U, I, \pi, G, S', V_s, V_o, r \rangle$.	Is there an auditable evidence bundle for humans and later improvement?

Table 2 Relationship between this survey and adjacent bodies of work

Area	Representative sources	Contribution	Why separate remediation analysis is needed
LLM cybersecurity	[39]	Broad map of LLM use in security.	Remediation execution, rollback, and service regression are not the organizing lens.
Agentic security	[40]	Agent workflow and agent-risk taxonomy.	Covers many security tasks; does not isolate vulnerability-to-change pipelines.
Autonomous cyber defense	[19, 41, 20]	Repeatable blue-agent environments.	Actions are often abstract compared with package, configuration, and service repair.
Patch-management practice	[3, 42, 1]	Human and organizational constraints.	Rarely evaluates adaptive agents that plan and execute changes.
Traditional automation	[43, 44]	Operational baselines and SOAR-style orchestration.	Often rule-bound; useful comparator rather than full agentic remediation.

defensive settings [40]. Autonomous cyber-defense benchmarks study blue-agent decision making under adversarial dynamics [19, 41, 20]. Human patching and patch-management studies explain why organizations do not patch immediately, even when remediation advice is available [3, 42, 1]. This survey is narrower: it asks whether adaptive systems can move from a vulnerability or unsafe state to a verified safer system state. See Table 2.

5 Review Protocol

5.1 Research questions

The review is structured around nine research questions. They are designed to synthesize existing literature while exposing and exploring the current gaps. Table 3 maps each research question to the evidence used in synthesising this survey.

RQ1

What system targets and vulnerability classes are

Table 3 Mapping of research questions to synthesis outputs

RQ	Evidence used	Output in this survey
RQ1	LLM configuration repair, ACD, RL patching, SOAR, benchmarks	Target and evidence-stream taxonomy
RQ2	IaC repair, scanner-to-fix systems, SOAR actions, SysRepair-Bench	Action-concreteness and lifecycle analysis
RQ3	LLM/RAG agents, MARL, DRL, rule supervisors, SOAR loops	Architecture patterns and reference architecture
RQ4	OpenVAS, Trivy, ATT&CK, NIST, knowledge graphs, policy layers	Grounding and governance taxonomy
RQ5	Cyber ranges, IaC corpora, patch studies, human-practice work	Metric families and evaluation ladder
RQ6	Rollback, regression, approvals, prompt/tool security	Safety case elements
RQ7	CybORG, CAGE, SysRepair-Bench, IaC corpora	Benchmark comparison
RQ8	Cross-stream synthesis	Research agenda and RQ answers
RQ9	NIST patch and incident guidance, vulnerability-management books, autonomic/self-adaptive systems	Formal repair task model and task-level evaluation criteria

being remediated by autonomous or semi-autonomous systems?

RQ2
What remediation actions are actually performed or prepared, and how concrete are those actions?

RQ3
What agent architectures are used, including single-agent, multi-agent, hierarchical, hybrid LLM/RL, and rule-constrained systems?

RQ4
How do systems ground decisions in scanners, documentation, threat intelligence, policy, asset context, and human approvals?

RQ5
What comparators, baselines, and evaluation metrics are used, and where are they inadequate?

RQ6
What safety, reliability, rollback, and audit mechanisms are reported?

RQ7
What benchmarks and artifacts support reproducible evaluation of real-system remediation?

RQ8
What research gaps must be closed before enterprise-grade deployment is credible?

RQ9
How can autonomous system repair be formalized as a set of tasks, action types, constraints, and success conditions?

5.2 Eligibility criteria

Table 4 summarizes the PICOS criteria. The inclusion boundary is intentionally narrow because the surrounding cybersecurity literature is large and noisy.

5.3 Search and screening process

The resulting search across academic databases yielded 470 sources. See Section A for search queries used. Each record was screened by title, abstract, venue, year, DOI, and notes when present. Screening used inclusion highlights such as LLM, agentic AI, multi-agent, autonomous cyber defense, CybORG, ACO gym, CVE, patching, remediation, Kubernetes, container, cloud, network, rollback, and verification. Exclusion highlights included SAST, source-code repair, SQL injection, XSS, buffer overflow, detection-only, triage-only, threat intelligence-only, static playbook, patent, product manual, and broad best-practice material. Full-text exclusion reasons were ordered to minimize ambiguity: not a system-level remediation target; no adaptive agentic component; no remediation execution; detection or triage only; code-level repair only; no empirical evaluation or comparator; and weak publication type or insufficient metadata. After adjudication, we identified 28 records for inclusion.

Year	Sources	Distribution
2021	1	
2022	3	
2023	4	
2024	6	
2025	11	
2026	3	

Fig. 2 Year distribution of the 28 sources actually cited in the survey. Unlike the screened-corpus distribution, no cited source has an unknown year. Older sources are retained for autonomic computing, self-adaptive systems, vulnerability management, and patch/incident-management foundations.

Table 4 PICOS criteria used to screen autonomous system vulnerability remediation studies

Dimension	Include	Exclude
Population	Enterprise IT, cloud, data-center, network, container, OS, endpoint, cyber-range, or SOC environments; CVEs, misconfigurations, missing patches, exposed services, weak permissions.	Isolated application code, mobile-only settings, SAST-centered source flaws, SQL injection/XSS/buffer-overflow repair without system workflow.
Intervention	LLM agents, MAS, RL/MARL, hybrid LLM/RL systems, adaptive SOAR, automated configuration or patch generation, guarded execution, verification.	Static scripts or playbooks without adaptive reasoning; recommendation-only tools; detection, prioritization, or threat-intelligence-only systems.
Comparator/context	Manual remediation, static automation, pure RL, single-agent and multi-agent alternatives, vulnerability-management lifecycle studies.	Descriptive frameworks with no baseline, operational outcome, or concrete remediation design.
Outcome	Fix success, patch/config correctness, failed deployment rate, rollback, service health, MTTR, tool success, planning quality, cost, residual risk.	Detection-only metrics, code-quality metrics alone, or developer productivity without system-level remediation.
Study type	Peer-reviewed studies, dissertations, empirical case studies, comparative experiments, high-quality technical reports and benchmarks.	Patents, product manuals, marketing pages, opinion pieces, editorials, weak metadata.

Table 5 Feature-extraction schema for synthesis

Dimension	Extracted fields	Why it matters
System target	OS, package, network, cloud, container, IaC, SOC, cyber range.	Separates system remediation from code-only repair.
Vulnerability class	Missing patch, CVE, misconfiguration, exposed service, weak policy, compromised host.	Determines whether remediation removes a vulnerability or only mitigates symptoms.
Action semantics	Recommendation, patch, script, config change, firewall rule, isolation, restore, rollback.	Distinguishes advice from executed remediation.
Architecture	LLM, RAG, ReAct, multi-agent, RL, MARL, rule supervisor, SOAR.	Explains where autonomy resides.
Grounding	Scanner output, asset graph, CVE/CWE, threat intelligence, docs, policy, approval.	Constrains hallucination and unsafe action.
Evaluation	Baseline, benchmark, case study, cyber range, real system, ablation, metric suite.	Controls strength of evidence.
Safety	Least privilege, sandbox, approval, canary, rollback, regression, audit.	Essential for production relevance.

5.4 Data extraction and quality appraisal

For included records, we extracted system target, vulnerability class, action semantics, architecture, grounding source, evaluation design, safety mechanism, and reproducibility artifact. This schema matters because a paper can be strong on one dimension and weak on another. For example, an IaC repair paper may produce concrete configuration diffs but lack live service regression tests; a cyber-range paper may have strong sequential decision evaluation but abstract away package-manager and rollback semantics. The screening outcome was informative. Autonomous remediation is a small subset of several larger areas: vulnerability management, automated cyber defense, LLMs for cybersecurity, SOAR, DevSecOps, and program repair. Many papers use the vocabulary of automation or autonomy while stopping at detection, prioritization, or recommendations. The remainder of this survey, therefore, uses a layered evidence model rather than pretending that all included work has the same maturity. See Section A.

5.5 Evidence roles

We classify sources into four roles.

Core evidence
: Directly studies automated or autonomous remediation, mitigation, or response actions at the system level.

Comparator evidence
: Optimizes or automates remediation decisions without a full LLM-agent loop, or supplies traditional automation baselines.

Context evidence
: Explains human patching behavior, cyber-defense benchmarks, safety constraints, or architecture concepts.

Watchlist evidence
Relevant frontier work that is preprint-only, weakly evaluated, or not yet strong enough to anchor claims.

6 Landscape of the Evidence

6.1 Included corpus

Table 6 summarizes representative studies used in the synthesis. The table is not exhaustive of every cited source; rather, it shows the main evidentiary spine.

6.2 Maturity pattern

The literature follows a maturity gradient. Static automation and enterprise patch management are operationally mature but weak on adaptive reasoning. RL vulnerability-management and cyber-range studies are strong on sequential decision making but often abstract away enterprise change management. LLM/IaC remediation studies generate concrete fixes but often stop at static validation or developer-facing guidance. Agentic SOAR papers provide tool execution and operational flow, but the evaluated tasks are usually incident-response demonstrations rather than vulnerability remediation campaigns. Benchmarks are improving, but most still emphasize either cyber-defense reward or isolated repair success instead of the combined security-and-service objective. See Table 7 for a clear maturity comparison across evidence streams.

7 Capability Taxonomy

Across the included and adjacent studies, autonomous remediation systems can be decomposed into five capabilities: state acquisition, grounding, planning, guarded execution, and verification. Table 8 summarizes the taxonomy, while Table 9 explores the capabilities covered across evidence streams.

7.1 State acquisition

A remediation agent needs a representation of the current system and the vulnerability condition. In cyber-range work, this state may be an observation space over hosts, services, attacker progress, and defensive affordances [6,19,41]. In configuration repair, the state is a concrete artifact such as Kubernetes YAML, Terraform, CloudFormation, or scanner-normalized JSON plus static-analysis findings [9,11,12,14]. In SOC-style response, the state combines alerts, logs, SIEM data, endpoint telemetry, and threat-intelligence enrichment [15].

State acquisition remains under-specified. Many prototypes assume clean input: a manifest, a vulnerability report, a simulated observation, or a curated incident dataset. Real enterprise remediation requires joining multiple imperfect sources: asset inventory, scanner results, exploit intelligence, package state, service ownership, business criticality, maintenance windows, dependency graphs, and monitoring signals. The absence of a shared state schema limits benchmark transfer and makes it difficult to compare an RL policy in a cyber range with an LLM agent that edits cloud configuration.

7.2 Knowledge and policy grounding

Remediation requires knowledge beyond the local observation. Systems ground decisions in vulnerability frameworks, ATT&CK mappings, cybersecurity knowledge graphs, static-analysis rules, organizational policies, and prior incident memory. Loevenich et al. use knowledge graphs linking infrastructure, CTI, CVEs, and ATT&CK techniques to support autonomous defense and analyst interaction [6]. Liu et al. enhance ATT&CK to connect attacks and mitigations before optimizing deployment [48]. LLMSecConfig grounds LLM output in retrieval and static-analysis checks for container orchestrator misconfiguration repair [9]; zero-touch remediation work extends this pattern with OpenVAS normalization, threat-intelligence retrieval, LLM verification, CI/CD execution, and rollback routing [14]. This grounding layer is the main difference between a remediation agent and a generic chatbot with shell access.

Grounding has two meanings. Technical grounding connects a finding to artifacts, documentation, and executable constraints. Organizational grounding connects the same finding to ownership, service criticality, compliance obligations, and acceptable downtime. The literature is stronger on technical grounding than organizational grounding. Human patching studies suggest that this gap is not incidental; operators delay patches because the missing knowledge is often social and operational rather than purely technical [42,3].

7.3 Planning and agent organization

LLM agents are attractive because they can interleave reasoning and action in patterns related to ReAct-style tool use [49]. Multi-agent organization appears in two forms. The first is cyber-defense MARL, where agents coordinate over a network defense problem and learn policies under attacker pressure [7,47]. The second is

Table 6 Representative core, comparator, context, and benchmark sources

Study	Environment	Mechanism	Remediation or response action	Role
Loevenich et al. [6]	Critical network segments	DRL, augmented LLM, rules, knowledge graph	Monitoring, decoy deployment, service removal, recovery	Core
Towhid et al. [8]	Cyber threat mitigation setting	Knowledge-infused RL with LLM-guided policies	Context-aware mitigation policy selection	Core
Ye et al. [9]	Kubernetes/container configs	LLM with RAG and static-analysis validation	Repair of container misconfigurations	Core
Low et al. [11]	IaC repositories	GPT-4 plus scanner and human context	Repaired IaC code for cloud misconfigurations	Core
Hsieh et al. [14]	OpenVAS and CI/CD	RAG LLM, verifier, orchestration	Validated patches with rollback routing	Core/frontier
Reyes et al. [13]	Terraform IaC	Fine-tuning and in-context learning	Corrected Terraform misconfiguration blocks	Core
Ismail et al. [15]	SOC-like response	Agentic LLM SOAR	Commands, blocking, MFA enforcement, monitoring	Adjacent core
Deep VULMAN [16]	CSOC vulnerability data	DRL plus integer programming	Selects vulnerability instances for mitigation	Comparator
Jia et al. [17]	Enterprise patching	Action-decomposed PPO	Optimizes patch timing	Comparator
Saad et al. [18]	Interconnected critical systems	Dependency modeling plus PPO	Dependency-aware patch strategy	Comparator
PALANTIR [45]	Cloud/edge/on-prem NFV	Ontology and orchestration	Automated threat mitigation deployment	Comparator
Polonio et al. [44]	SDN networked systems	SOAR-style orchestration and risk ranking	Automated mitigation of high-risk network vulnerabilities	Comparator
Paulraj et al. [46]	Critical infrastructure	Hybrid AI framework	Real-time threat mitigation and automated remediation	Core context
Jenkins et al. [42]	System administrators	Empirical study	Human patching behavior	Context
Dissanayake et al. [3]	Patch-management practice	Empirical interviews	Automation limits across patch workflow	Context
CybORG/CAGE [19,20,47]	Cyber ranges	ACO/MARL benchmark environments	Blue-agent defense actions	Benchmark
SysRepair-Bench [21]	Containers and VMs	Real-system remediation benchmark	PoC plus service regression checks	Benchmark
Malul et al. [10]	Kubernetes configs	LLM detection, localization, reasoning	Suggested misconfiguration remediation (advisory)	Context

LLM-agent decomposition, where specialized roles investigate, validate, execute, and monitor a response [15]. IaC workflows sometimes decompose retrieval, detection, and report generation into distinct agents [12].

Planning is where LLM and RL approaches differ most. RL optimizes an action policy under a reward function; it can learn timing, resource allocation, or mitigation selection under uncertainty [16,17,18]. LLM agents generate plans that can be read and adapted by humans, but are prone to hallucination and weak global optimality. A plausible architecture uses RL or constrained optimization for low-level action selection under formal objectives, and LLM agents for context interpretation, exception handling, explanation, and workflow integration.

7.4 Guarded execution

Execution is the defining boundary of this survey. A system that only recommends remediation is not enough for core evidence. Included systems perform or model actions such as service removal, recovery, decoy deployment, mitigation deployment, YAML repair, IP blocking, MFA enforcement, patching, network blocking, or workaround selection [6,9,15,50]. SysRepair-Bench explicitly scopes actions to system-administration primitives such as editing configuration files, installing or removing packages, changing permissions, managing services, and applying firewall rules [21].

Guardrails matter because the same tool access that makes remediation useful also makes it risky. Current studies typically rely on constrained action spaces, static-analysis validation, human-in-the-loop options, or cyber-range isolation. Production-grade rollback, blast-radius estimation, policy-as-code approval, canary deployment,

Table 7 Maturity comparison across evidence streams

Stream	Concrete actions	Adaptive reasoning	Realistic systems	Safety evidence	Interpretation
LLM configuration/IaC repair	High	Medium	Medium	Medium	Closest to direct remediation, but deployment realism varies.
ACD cyber-range agents	Medium	High	Medium	Low–medium	Strong dynamics, weaker concrete patch semantics.
RL patch/vulnerability management	Medium	Medium	Low–medium	Low	Useful policy layer, rarely end-to-end execution.
Agentic SOAR/self-healing	Medium	Medium–high	Medium	Medium	Operationally plausible, but response/remediation boundary is blurred.
Traditional automation	High	Low	High	Medium	Baseline and process guardrail, not agentic by itself.
Benchmarks	Medium–high	Variable	Medium	Medium	Essential infrastructure; realism and standard metrics still emerging.

Table 8 Capability taxonomy for autonomous system vulnerability remediation

Capability	Function	Typical artifacts	Failure mode
State acquisition	Build a current view of vulnerable assets, services, dependencies, attacker progress, and constraints.	Scanner JSON, OpenVAS reports, Kubernetes YAML, Terraform, logs, topology graphs, CybORG observations.	Stale or incomplete state causes wrong fix or unsafe action.
Grounding	Connect state to policy, vulnerability knowledge, standards, organizational constraints, and safe action rules.	ATT&CK mappings, CVEs, CTI, CIS benchmarks, NIST guidance, knowledge graphs, RAG indexes.	Hallucinated or policy-inconsistent remediation.
Planning	Select action sequence under risk, cost, dependency, and service constraints.	LLM plans, RL policies, integer programs, playbooks, approval proposals.	Optimizes local risk while ignoring service or dependency harm.
Guarded execution	Apply changes through least-privilege tools with approval gates, canaries, and rollback hooks.	Shell commands, IaC patches, CI/CD pipelines, SOAR actions, Ansible/Puppet, firewall rules.	Overbroad command, destructive change, failed deployment.
Verification	Prove vulnerability removal and service continuity; update evidence and residual risk.	PoC checks, scanner reruns, regression probes, health checks, rollback tests, audit bundles.	False closure: vulnerability persists or service broke.

Table 9 Capability coverage in representative evidence streams

Source family	State	Ground	Plan	Execute	Verify	Notes
LLM configuration/IaC repair [9, 11, 13, 10]	Y	P	P	P	P	Concrete artifacts; deployment and service realism vary.
Scanner-to-remediation [14, 44]	Y	Y	P	Y	P	Strong orchestration, but adaptive reasoning depth varies.
Hybrid LLM/DRL ACD [6, 8]	Y	P	Y	Y	P	Strong sequential evaluation; actions often abstract.
RL patch management [16, 17, 18]	P	P	Y	P	–	Optimizes policy but rarely executes system changes.
Agentic SOAR [15]	Y	P	Y	Y	P	Tool-rich SOC workflow; response/remediation mixed.
SysRepair-Bench [21]	Y	P	Y	Y	Y	Real-system PoC plus regression scoring.

and least-privilege tool wrappers remain underdeveloped. The field should treat execution as a safety-critical control plane, not merely as the final step after planning.

7.5 Verification and learning

Post-action verification closes the remediation loop. LLM-SecConfig validates syntax and security properties after LLM-generated fixes [9]. Cyber-defense agents evaluate mission outcomes, attack mitigation, and system recovery in simulated environments [6]. SOAR-oriented work uses validation and active monitoring stages [15]. SysRepair-Bench requires both a proof-of-concept check and a service-regression check, giving the field a concrete benchmark pattern for distinguishing real remediation from damaging change [21]. Verification must eventually

cover both security and service health: the vulnerability should be removed, the business function should survive, and the action should be reversible or auditable. This is also where learning should occur. A failed patch should update future planning, not merely produce a failed benchmark episode. A successful compensating control should be tied to residual risk and expiration, not treated as a permanent fix.

8 Evidence Stream 1: LLM-Based Configuration and IaC Remediation

The most direct LLM remediation evidence concerns configuration and IaC artifacts. Cloud, container, and IaC settings are structured enough for parsing and validation, but semantically rich enough that static rules can miss context-dependent errors. LLMSecConfig addresses security misconfigurations in container orchestrators, using static-analysis tools to detect and validate issues, retrieval to ground LLM outputs, and an LLM to generate repairs [9]. The architecture: generation is surrounded by detection, retrieval, and validation. The system therefore treats the LLM as one component in a remediation pipeline rather than as an unconstrained operator. A closely related system, GenKubeSec, addresses the same Kubernetes setting but stops earlier in the remediation loop: it detects a wide range of configuration-file misconfigurations, localizes each one, explains why it is a concern, and produces suggested remediation, with detection precision and recall competitive with industry rule-based tools and remediation quality judged correct and useful by a Kubernetes security expert [10]. Because that remediation is advisory rather than automatically applied and re-verified, it illustrates the recommendation end of the execution ladder rather than guarded execution.

Low et al. extend LLM repair to IaC repositories by feeding scanner findings, IaC code, and human-provided context to GPT-4 [11]. Their results are useful because they report both scanner-alarm reduction and hallucination-like failures: some generated fixes pass one check but fail syntax, schema, or semantic correctness. This is exactly the kind of evidence the field needs. A remediation system should not be evaluated only on whether the scanner stops complaining; it must also check whether the change is valid and whether the underlying risk was actually addressed.

Toprani and Madisetti present a multi-agent RAG workflow for CloudFormation/IaC vulnerability detection and remediation guidance [12]. The work is relevant because it decomposes retrieval, analysis, and reporting roles, and because it focuses on cloud framework misconfigurations. Its strength lies less in direct autonomous remediation evidence because the described output is closer to actionable remediation guidance than to automatically applied system changes.

Reyes et al. focus on Terraform vulnerability remediation using Trivy outputs, parameter-efficient fine-tuning, and in-context learning [13]. The construction of thousands of corrected Terraform blocks is especially valuable for future benchmark design. However, common NLP metrics such as BLEU and ROUGE are only weak proxies for secure infrastructure repair. Future IaC remediation evaluations should include provider validation, policy checks, deployment simulation, service regression, and exploitability checks.

Hsieh et al. propose a zero-touch framework combining OpenVAS findings, threat intelligence, RAG-enhanced LLMs, dual-model verification, CI/CD patch execution, confidence routing, and rollback [14]. This is one of the closest matches to the full remediation loop. It also exposes the complexity of the loop: scanner normalization, retrieval quality, confidence thresholds, verifier disagreement, deployment routing, and rollback policy each become possible points of failure.

Overall, LLM-based configuration remediation is the most practically near-term branch of the field. It benefits from structured artifacts, existing CI/CD gates, scanner outputs, and policy-as-code tools. Its main limitation is runtime validation. Static checks can show that a YAML or Terraform file is clean to run, but they do not prove that the deployed service still works or that the attacker path is closed.

9 Evidence Stream 2: Autonomous Cyber Defense Agents

Autonomous cyber defense (ACD) research provides the strongest evidence for sequential decision-making agents under adversarial pressure. CybORG was introduced as a gym for developing autonomous cyber agents [19]. CAGE Challenge 2 and CAGE Challenge 4 provide benchmark tasks for blue agents that observe, analyze, restore, remove malicious access, deploy decoys, or coordinate across subnets [41,20,47]. These environments do not look like vulnerability scanners, but they force researchers to confront temporal tradeoffs: when to observe, when to disrupt an attacker, when to restore a host, and when an action harms legitimate users.

Loevenich et al. design and evaluate an ACD agent that integrates DRL, an augmented LLM, rules, and knowledge grounding for defensive action in critical network segments [6]. The action space includes monitoring, analysis, decoy deployment, service removal, and recovery. The evaluation compares DRL variants in a cyber-

operations gym and separately evaluates knowledge-graph and RAG components. This work is highly relevant because it treats remediation-like action as part of a dynamic defensive loop, not as static advice.

The MILCOM work by Loevenich et al. is more architectural and training-oriented, but it identifies constraints that matter for deployment: tactical networks, incomplete data, resource limitations, and adversarial adaptation [7]. Towhid et al. connect LLM guidance with RL policy learning for cyber threat mitigation [8]. Together, these papers suggest a design pattern: use LLMs and knowledge bases to enrich sparse cyber observations, then use RL or rule-constrained policies to select actions in a bounded environment.

CAGE/CybORG also highlights the benchmark problem. A blue agent can learn high reward in a simplified topology, but enterprise remediation requires maintenance windows, dependencies, service owners, compliance constraints, and real package managers. The value of ACD benchmarks is repeatability and adversarial dynamics; the limitation is operational realism. The research frontier is the translation of cyber-range actions into safe, auditable change operations.

10 Evidence Stream 3: RL for Vulnerability and Patch Management

RL and optimization studies provide a bridge between traditional vulnerability management and agentic remediation. They often do not execute patches directly, but they model decisions that any autonomous remediation system must make: which vulnerabilities to address, when to patch, how many resources to allocate, and how to account for dependencies.

Deep VULMAN frames vulnerability management as sequential decision making under uncertain vulnerability contexts and constrained mitigation resources [16]. Its combination of a DRL agent with integer programming is important because it separates resource allocation from the selection of vulnerability instances. The system is not an LLM agent, but it directly addresses a core remediation bottleneck: selecting mitigation actions under uncertainty.

Jia et al. model enterprise security patch management as an MDP and propose an action-decomposed PPO to balance cost, security risk, and uncertainty [17]. Saad et al. extend this style of work to dependency-aware patch management for critical interconnected systems, modeling cascading risk and using PPO for adaptive decisions [18]. Liu et al. optimize mitigation deployment over an enhanced ATT&CK model using soft actor-critic [48]. Ur-Rehman et al. model patching, network controls, and workarounds for hybrid networks under cost and effectiveness constraints [50].

These studies define decision variables that LLM agents should not ignore: action cost, critical node selection, mitigation redundancy, dependency propagation, operational constraints, and residual risk. In many cases, an LLM should probably not choose a patch schedule by itself. A hybrid architecture can use optimization or RL to propose a policy, while an LLM explains it, checks policy exceptions, gathers missing context, and packages the change request for humans or tools.

11 Evidence Stream 4: Agentic SOAR and Self-Healing Operations

Traditional SOAR systems automate response workflows, but most are rule-bound. They are useful comparators, not core agentic evidence. Agentic SOAR becomes relevant when LLM or adaptive agents dynamically investigate alerts, plan technical steps, execute commands, and validate outcomes.

Ismail et al. propose an agentic LLM SOAR architecture with an Investigation-Validation-Active Monitoring workflow and demonstrate autonomous response actions such as blocking an IP and enforcing MFA in a SOC-like environment [15]. Rather than being limited to vulnerability remediation, this framework provides the essential operational machinery that remediation agents require: tool routing, task decomposition, response validation, and continuous monitoring.

The broader self-healing operations literature is relevant when it provides recovery, dependency, or rollback patterns. However, reliability self-healing and security remediation are not the same. Restarting a service can restore availability while leaving a vulnerability exposed; disabling a vulnerable service can remove risk while breaking a business process. Security remediation agents need both dimensions.

Agentic SOAR also raises an evidence-quality issue. Many demonstrations validate a small set of incident types. For vulnerability remediation, the same architecture should be tested against scanner findings, cloud posture violations, exposed services, missing patches, and configuration drift, with repeated trials, adversarially perturbed inputs, and clear failure categories.

12 Evidence Stream 5: Traditional Automation and Human Practice

Traditional patch automation matters because it is the baseline an autonomous agent must outperform. Satellite and Puppet integration, for example, can automate

Table 10 Benchmark comparison for autonomous remediation research

Benchmark family	Measures well	Undermeasures	Needed extension
CybORG/CAGE [19, 41, 20]	Sequential defense, adversarial dynamics, multi-agent coordination.	Concrete package/configuration repair and service regression.	Add executable remediation tasks and rollback outcomes.
IaC/configuration corpora [11, 13, 9]	Patch generation for structured infrastructure artifacts.	Runtime drift and multi-service dependencies.	Combine static validation with deploy-and-test harnesses.
Scanner-to-remediation pipelines [14, 44]	End-to-end scanning, orchestration, and mitigation.	Cross-organization reproducibility and independent baselines.	Standardize logs, actions, and safety metrics.
SysRepair-Bench [21]	Real-system vulnerabilities, PoC checks, service regression checks.	Peer-reviewed analysis and scenario coverage still need growth.	Pair with failure taxonomy and leaderboards.

Linux patch deployment [43]. Enterprise patch guidance from NIST describes planning, prioritization, testing, deployment, and verification practices that any autonomous agent must respect [1]. Empirical studies show why organizations do not simply patch everything immediately [3, 42].

The human-practice literature reframes the problem. Operators need confidence about compatibility, service impact, ownership, timing, rollback, and auditability. An autonomous remediation agent that produces only a shell command or a YAML diff is therefore incomplete. It must produce an evidence bundle: what was observed, what was selected, why alternatives were rejected, which policy allowed the action, what was tested, how rollback works, and what residual risk remains.

This literature also warns against measuring only speed. A lower mean time to remediate is useful only if the remediation is correct, safe, and durable. Automation that closes tickets faster while hiding failed deployments or compensating-control debt may reduce visible backlog without reducing actual risk.

13 Evidence Stream 6: Benchmarks and Evaluation Infrastructure

Benchmarks determine what the field optimizes. CAGE / CybORG benchmarks reward blue-agent behavior under adversarial dynamics and help researchers compare algorithms [19, 20, 47]. SysRepair-Bench provides a complementary evaluation style: real-system remediation scenarios where success requires both vulnerability proof-of-concept failure and service regression success [21]. These benchmark families should converge.

A benchmark for autonomous system vulnerability remediation should include the following properties.

1. The vulnerable state should be reproducible from source or image definitions.
2. The agent should operate through realistic administrative interfaces.
3. The security check should test exploitability or attack-surface reduction, not only scanner disappearance.
4. The regression check should test service availability and intended functionality.
5. The benchmark should record command count, wall-clock time, tool errors, destructive actions, and rollback behavior.
6. The task should distinguish direct remediation from compensating control.
7. The scoring should preserve failure categories, not only binary pass/fail.

The most important benchmark principle is joint scoring. A fix that removes the vulnerability but breaks the service is a failed remediation. A change that preserves the service but leaves the exploit open is also a failed remediation. Benchmarks should therefore distinguish fixed-and-healthy, fixed-but-broken, unchanged-but-safe, unchanged-and-vulnerable, and unsafe-side-effect outcomes. See Table 10 for a detailed benchmark comparison.

14 Architectural Patterns

14.1 Scanner-to-fix pipelines

The simplest architecture starts with a scanner finding and ends with a generated fix. OpenVAS/RAG and IaC repair systems follow this pattern [14, 11, 13]. The scanner provides a structured finding, the agent retrieves remediation context, the generator proposes a change, validators check it, and an orchestrator routes it to CI/CD or a human. This pattern is attractive because it maps well to existing DevSecOps pipelines.

14.2 Policy-optimized remediation

RL and optimization studies use a different architecture: represent the environment state, learn or solve a

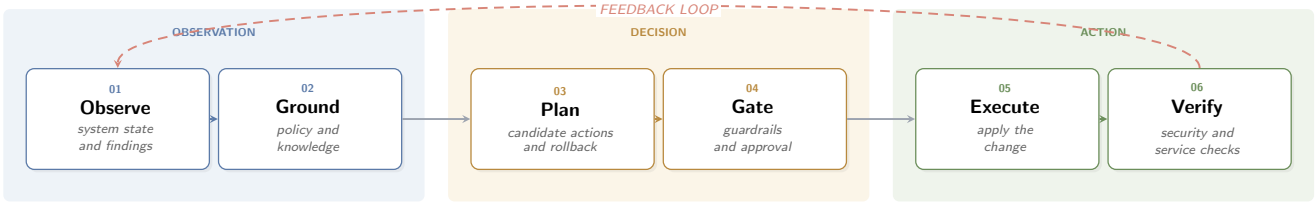

Fig. 3 Reference architecture: a closed-loop control plane—observe, decide, act, verify. The model-facing planning layer is separated from deterministic guardrails and verification, so a generated plan cannot directly become an unsafe production change.

policy, and output a mitigation or patch decision [16, 17,18]. These systems are valuable when decisions have temporal coupling or resource constraints. The action may not be a concrete patch file; it may be a decision about what to patch now.

14.3 Cyber-range blue agents

Cyber-range agents interact with simulated adversaries and learn defensive policies over time [6,19,20]. They are strong for studying adversarial dynamics, partial observability, and recovery actions. Their main weakness is that the action space is usually simplified compared with production systems.

14.4 Agentic SOAR loops

Agentic SOAR loops start from alerts or findings, investigate context, validate hypotheses, execute a response, and monitor results [15]. This architecture is closest to SOC practice, but it must be specialized for vulnerability remediation. In particular, it needs explicit change-management integration, rollback, and service regression checks.

14.5 Hybrid control planes

The most promising future architecture is hybrid. An LLM agent gathers context, explains choices, and handles exceptions. A retrieval layer grounds it in policy and technical documentation. A constrained planner or RL policy selects actions under cost and risk. A deterministic executor applies changes through least-privilege wrappers. Validators and monitors decide whether the action succeeded, failed, or requires rollback. This architecture reduces reliance on any single model and makes safety properties explicit.

15 Evaluation Metrics

Current studies report heterogeneous metrics. Table 11 groups the main metric families, while Table 12 presents an evaluation ladder for autonomous system vulnerability remediation, ranging from basic detection through governed autonomy. The key gap is that security success and operational safety are often evaluated separately. A patch that closes a vulnerability but breaks a service is not a successful remediation. Conversely, a safe change that leaves the vulnerability exploitable is also not successful. Benchmarks such as SysRepair-Bench point in the right direction by requiring both proof-of-concept and regression checks [21]. Future evaluations should also report rollback, false closure, and unsafe-action rates.

16 Safety and Reliability Challenges

Autonomous remediation agents create a new safety problem: they are authorized to alter systems. Table 13 organizes the safety case into claims, required evidence, and failure consequences; the following challenges recur across the corpus and adjacent literature.

Bounded autonomy. Agents need constrained action spaces, least-privilege credentials, approval thresholds, and environment-specific policies. High-impact actions should require stronger evidence or human approval.

Rollback and reversibility. Remediation plans should include pre-change snapshots, inverse operations, and rollback tests. The zero-touch OpenVAS/RAG framework and enterprise patch-management guidance make rollback explicit, but most autonomous cyber-defense papers still treat recovery as an action rather than as a separately verified safety property [14,1].

Prompt and tool security. LLM agents are vulnerable to prompt injection, malicious tool outputs, compromised documentation, and multi-agent control-flow attacks. Surveys on LLM cybersecurity and agentic security show that these risks are now a central part of deploying agents in security workflows [39,40].

Table 11 Metric families for autonomous remediation evaluation

Metric family	Examples	Why it matters
Remediation efficacy	Vulnerabilities fixed, exploit blocked, scanner finding removed, risk reduction.	Measures security objective, but can hide service damage.
Operational safety	Service health, failed deployment, rollback success, policy violation, destructive action.	Prevents agents from trading security for outage.
Efficiency	MTTR, tool calls, command count, compute cost, analyst effort.	Captures operational value beyond correctness.
Agent performance	Planning validity, tool success, reward, action validity, reasoning trace quality.	Diagnoses model and orchestration failure.
Comparative value	Manual baseline, static playbook, pure RL, LLM-only, hybrid architecture.	Shows whether autonomy adds value.
Durability	Recurrence, drift, patch persistence, compensating-control expiration.	Prevents temporary or brittle closure.
Auditability	Evidence completeness, approval trace, reproducible transcript, residual risk.	Supports compliance and incident review.

Table 12 Evaluation ladder for autonomous system vulnerability remediation

Level	Evidence type	Minimum requirement	Typical source
0	Detection or triage	Identifies vulnerability or unsafe state.	Excluded from core.
1	Recommendation	Produces human-readable remediation advice.	LLM assistants.
2	Candidate change	Produces patch/config/script/IaC diff that passes static checks.	IaC/container repair.
3	Executed change	Applies change in a controlled environment.	SOAR or CI/CD pipelines.
4	Verified remediation	Proves vulnerability no longer works and service still works.	SysRepair-Bench-style tasks.
5	Governed autonomy	Adds rollback, audit, approvals, policy compliance, and monitoring.	Desired enterprise target.

Hallucinated fixes. Grounding and validation reduce but do not eliminate hallucinated commands or configuration changes. Low et al. show that generated IaC fixes may reduce scanner alarms while still failing syntax, schema, or semantic checks [11]. Static checks, policy-as-code, canary deployment, and differential testing should surround generation.

Auditability. Security teams need a durable record of observations, retrieved evidence, plans, commands, outputs, approvals, and verification results. Reasoning traces alone are not enough; logs must be structured and tamper-evident.

Benchmark leakage and realism. Cyber-range agents may overfit to simplified topologies. Configuration-repair agents may overfit to public examples. CAGE / CybORG papers have improved repeatability for blue-agent evaluation, while CAGE Challenge 4 adds a multi-agent enterprise-scale structure [19, 20, 47]. Benchmarks should still separate training and evaluation artifacts and include unseen products, policies, and failure modes.

Compensating-control debt. Some vulnerabilities cannot be directly patched because of legacy dependencies or operational constraints. Compensating controls should be tracked as temporary risk decisions with expiration, monitoring, and owner accountability.

17 Research Agenda

17.1 A common task model

The field needs a shared task schema for system remediation: finding, affected asset, environmental context, candidate actions, safety constraints, validation checks, rollback plan, and post-action evidence. Such a schema would let researchers compare cyber-range agents, cloud-configuration agents, SOC agents, and real-system benchmarks under a common vocabulary.

17.2 Benchmarks that include change risk

Current benchmarks often reward attack mitigation or static repair. SysRepair-Bench is a useful emerging counterexample because it scores real-system remediation scenarios only when both the vulnerability proof-of-concept and service regression checks pass [21]. Future benchmarks should include service-health probes, dependency checks, blast-radius estimates, adversarially misleading context, and realistic rollback requirements.

Table 13 Safety case elements for autonomous remediation systems

Safety claim	Evidence needed	Failure if omitted
The agent understood the vulnerable state.	Scanner result, asset context, logs, package/config state, uncertainty.	Acts on wrong host, wrong version, or false positive.
The selected action is authorized.	Policy match, owner approval, maintenance window, least-privilege tool.	Unauthorized or mistimed production change.
The action is likely safe.	Static validation, dependency analysis, dry run, canary plan.	Service outage or data loss.
The action fixed the vulnerability.	Exploit PoC fails, scanner resolves, attack path reduced.	Cosmetic change or residual exploitability.
The service remains healthy.	Regression tests, health checks, monitoring, user-impact metrics.	Secure but broken system.
The action is reversible.	Snapshot, rollback command, rollback test, residual-risk note.	Persistent damage after failure.
The action is accountable.	Evidence bundle, audit log, approver, model/tool versions.	No post-hoc explanation or compliance trace.

17.3 Hybrid control architectures

LLMs should not be the only control layer. The most promising systems combine LLMs for context interpretation, retrieval, explanation, and exception handling with constrained planners, RL policies, policy-as-code validators, and deterministic execution wrappers. This hybrid design aligns with the strongest core studies [6, 8, 9].

17.4 Evidence-centered remediation

Every autonomous remediation should produce an evidence bundle: source finding, affected assets, retrieved references, selected action, expected effect, safety tests, execution transcript, post-action validation, rollback state, and residual risk. This bundle can support human trust, compliance, and later incident review.

17.5 From single actions to lifecycle integration

Real vulnerability management includes finding, prioritizing, remediating, verifying, accepting risk, and monitoring drift. Most studies isolate one step. Future systems should evaluate end-to-end lifecycle performance, including how an agent handles conflicting scanner data, failed patches, partial rollouts, recurring vulnerabilities, and compensating controls.

17.6 Human-agent collaboration

The human should not be reduced to a rubber stamp. Operators can provide missing business context, approve high-risk changes, and interpret ambiguous service impacts. Agents should reduce toil by preparing evidence and low-risk changes, while escalating decisions that require accountability. Evaluation should therefore measure handoff quality, not only autonomous pass rates.

17.7 Security of the remediation agent

A remediation agent is part of the attack surface. It reads untrusted logs, documentation, tickets, web pages, and scanner outputs. It may hold credentials capable of changing infrastructure. Research should therefore include prompt-injection resistance, tool-output validation, least privilege, credential scoping, and tamper-evident action logging.

18 Answers to the Research Questions

18.1 RQ1: Targets and vulnerability classes

The strongest direct evidence concerns containers, Kubernetes, Terraform/IaC, scanner findings, package-level issues, SDN-managed network devices, and system administration benchmark tasks [9, 11, 13, 44, 21]. Cyber-range studies target compromised hosts, adversary footholds, and defensive posture rather than specific CVE patching [19, 20]. The least-developed target is heterogeneous enterprise remediation across OS, identity, cloud, network, and service dependencies in one benchmark.

18.2 RQ2: Action concreteness

Actions range from advice to verified system changes. LLM/IaC studies produce candidate configuration changes; scanner-to-remediation and SOAR systems execute or orchestrate mitigations; cyber ranges use abstract actions such as restore, isolate, remove, or decoy; SysRepair-Bench requires concrete system fixes verified by PoC

and regression checks [14,15,21]. The field should report action concreteness explicitly.

18.3 RQ3: Architectures

Architectures include LLM repair, RAG, ReAct-style tool use, multi-agent workflows, DRL/MARL policies, LLM-guided RL, SOAR loops, and deterministic automation baselines [49,12,6,8,15]. The most credible deployment path is hybrid rather than model-only.

18.4 RQ4–RQ8: Grounding, metrics, safety, benchmarks, and gaps

Grounding uses scanners, IaC artifacts, OpenVAS/Trivy outputs, CVE/CWE knowledge, ATT&CK, vendor documentation, threat intelligence, and patch-management guidance [14,13,1]. Metrics remain fragmented across scanner reduction, reward, patch cost, MTTR, and human effort. Safety mechanisms are acknowledged more often than measured: rollback success, failed deployment rate, credential containment, and audit completeness are rarely first-class outcomes. CybORG/CAGE and SysRepair-Bench are complementary benchmark directions; future work should combine adversarial dynamics with concrete system repair. RQ9 is answered by the formal task model in Sect. 3: autonomous system repair can be decomposed into intake, state reconstruction, diagnosis, repair-intent selection, plan synthesis, safety gating, execution, verification, rollback, and evidence capture.

19 Threats to Validity

Preprints were tracked as frontier signals but not treated as primary archival evidence unless supported by venue information. Repository artifacts such as SysRepair-Bench are useful benchmark sources but are not peer-reviewed publications. Terminology drift is another limitation. Autonomous cyber defense, agentic SOAR, self-healing, automated remediation, vulnerability management, and patch management overlap but are not synonyms. The review therefore errs on the side of explicit inclusion criteria rather than broad keyword matching.

20 Conclusion

Autonomous system vulnerability remediation is an emerging research area at the intersection of vulnerability management, autonomous cyber defense, LLM agents, RL, security operations, and DevSecOps. The strict evidence base is still small, but it now reveals a coherent shape. Useful remediation agents need more than vulnerability detection and more than free-form generation. They require grounded state representations, policy-aware planning, constrained execution, post-action verification, rollback, and auditable evidence.

The most promising near-term path is hybrid. LLMs can interpret context and generate candidate changes; retrieval and policy layers can ground those changes; RL and optimization can handle sequential tradeoffs; deterministic wrappers can execute safely; and benchmarks can verify both exploit removal and service health. The next phase of research should move from impressive individual demonstrations toward reproducible, safety-aware benchmarks that measure whether an agent can reduce security risk without creating operational damage.

Acknowledgements The authors thank the Central Texas Cyber Range (CTCR) at Baylor University for the research environment and infrastructure that supported this work, and the maintainers of the public benchmark and software repositories cited in this survey.

Statements and Declarations

Funding This work was supported by the Central Texas Cyber Range (CTCR) at Baylor University and by the U.S. Department of Education (NSCA Delta).

Competing interests Erika Leal serves as Director of Research and Development of the Central Texas Cyber Range, which supported this work; this relationship is disclosed in the interest of transparency. The authors have no other financial or non-financial competing interests relevant to the content of this article.

Ethics approval Not applicable. This study is a review of previously published literature and did not involve human participants or animals.

Consent Not applicable.

Data availability Data sharing is not applicable to this article, as no new datasets were generated or analysed. The list of screened and included records is available from the corresponding author on reasonable request.

Author contributions All authors contributed to the conception and design of the survey. The literature search, screening, and analysis were performed by Abanisenioluwa Kolawole Orojo, Webster Chiedozie Elumelu, and Emmanuelli El-Mahmoud. The work was supervised

Table 14 Condensed answers to RQ1–RQ9

RQ	Short answer	Most important gap
RQ1	Targets include containers, Kubernetes, IaC, scanner findings, cyber ranges, SDN networks, patch portfolios, and system benchmarks.	Heterogeneous enterprise environments.
RQ2	Actions range from advice to generated diffs to executed and verified system fixes.	Action concreteness is inconsistently reported.
RQ3	Architectures include LLM/RAG, multi-agent, DRL/MARL, LLM-guided RL, SOAR loops, and hybrids.	Few ablations identify which component prevents failure.
RQ4	Grounding uses scanners, docs, threat intelligence, policy, and observations.	Local business and change context is thin.
RQ5	Metrics vary by stream and are hard to compare.	Shared metric suite needed.
RQ6	Safety appears through validation, rollback, approvals, and regression checks.	Rollback and failed-deployment rates rarely measured.
RQ7	CybORG/CAGE and SysRepair-Bench are complementary.	Need benchmarks with both adversarial dynamics and concrete repair.
RQ8	Deployment blockers are operational safety, reproducibility, governance, and accountability.	Need safety-case-driven evaluation.
RQ9	Autonomous repair is a closed-loop task over state, findings, knowledge, policy, actions, constraints, verification, rollback, and evidence.	Papers should report the highest completed task level and failure provenance.

by Erika Leal. The first draft was written by Abanisenioluwa Kolawole Orojo, and all authors commented on previous versions. All authors read and approved the final manuscript.

References

1. M. Souppaya, K. Scarfone, Guide to enterprise patch management planning: Preventive maintenance for technology. Tech. Rep. NIST SP 800-40 Rev. 4, National Institute of Standards and Technology (2022). DOI 10.6028/NIST.SP.800-40r4. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r4.pdf
2. Joint Task Force, Security and privacy controls for information systems and organizations. NIST Special Publication NIST Special Publication 800-53 Rev. 5, National Institute of Standards and Technology, Gaithersburg, MD (2020). DOI 10.6028/NIST.SP.800-53r5. URL https://doi.org/10.6028/NIST.SP.800-53r5
3. N. Dissanayake, A. Jayatilaka, M. Zahedi, M.A. Babar, in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Association for Computing Machinery, New York, NY, USA, 2023), ASE '22. DOI 10.1145/3551349.3556969. URL https://doi.org/10.1145/3551349.3556969
4. P. Stöckle, M. Sammereier, B. Grobauer, A. Pretschner, in *Proceedings of the 2023 IEEE/ACM International Conference on Automation of Software Test (AST)* (IEEE, 2023), pp. 90–100. DOI 10.1109/AST58925.2023.00013. URL https://doi.org/10.1109/AST58925.2023.00013
5. N. Dissanayake, M. Zahedi, A. Jayatilaka, M.A. Babar, Proceedings of the ACM on Human-Computer Interaction **6**(CSCW2), 1 (2022). DOI 10.1145/3555087. URL https://doi.org/10.1145/3555087
6. J.F. Loevenich, E. Adler, T. Huerten, R.R.F. Lopes, Computer Networks **262**, 111162 (2025). DOI 10.1016/j.comnet.2025.111162. URL https://doi.org/10.1016/j.comnet.2025.111162
7. J.F. Loevenich, E. Adler, A. Becue, A. Velazquez, K. Wrona, V. Boshnakov, J. Falkcrona, N. Nordbotten, O.L. Worthington, J. Roning, R.R.F. Lopes, in *MILCOM 2024 - 2024 IEEE Military Communications Conference* (IEEE, 2024), pp. 158–163. DOI 10.1109/MILCOM61039.2024.10773923. URL https://doi.org/10.1109/MILCOM61039.2024.10773923
8. M.S. Towhid, S. Iqbal, E.C.P. Neto, N. Shahriar, S. Buffett, M. Sultana, A. Taylor, in *2025 22nd Annual International Conference on Privacy, Security and Trust (PST)* (IEEE, 2025), pp. 1–10. URL https://doi.org/10.1109/PST65910.2025.11268866
9. Z. Ye, T.H.M. Le, M.A. Babar, in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)* (IEEE, 2025), pp. 629–641. DOI 10.1109/MSR66628.2025.00099. URL https://doi.org/10.1109/MSR66628.2025.00099
10. E. Malul, Y. Meidan, D. Mimran, Y. Elovici, A. Shabtai. GenKubeSec: LLM-based kubernetes misconfiguration detection, localization, reasoning, and remediation (2024). DOI 10.48550/arXiv.2405.19954. URL https://arxiv.org/abs/2405.19954
11. E. Low, C. Cheh, B. Chen, in *2024 IEEE Secure Development Conference (SecDev)* (2024), pp. 20–27. DOI 10.1109/SecDev61143.2024.00008
12. D. Toprani, V.K. Madisetti, IEEE Access **13**, 69175 (2025). DOI 10.1109/ACCESS.2025.3560911. URL https://doi.org/10.1109/ACCESS.2025.3560911
13. R.Y. Reyes, B.M. Ampel, H. Chen, in *Proceedings of the 20th Pre-ICIS Workshop on Information Security and Privacy* (2025). URL https://aisel.aisnet.org/wisp2025/1
14. C.H. Hsieh, C.Y. Cheng, Y.C. Wang, Mathematics **14**(6) (2026). DOI 10.3390/math14061072. URL https://www.mdpi.com/2227-7390/14/6/1072
15. Ismail, R. Kurnia, Z.A. Brata, G.A. Nelistiani, S. Heo, H. Kim, H. Kim, Information **16**(5) (2025). DOI 10.3390/info16050365. URL https://www.mdpi.com/2078-2489/16/5/365
16. S. Hore, A. Shah, N.D. Bastian, Expert Systems with Applications **221**, 119734 (2023). DOI 10.1016/j.eswa.2023.

119734. URL https://doi.org/10.1016/j.eswa.2023.119734
17. Q. Jia, X. Qu, Z. Jiang, C. Wang, in *ICIS 2024 Proceedings* (2024), 6. URL https://aisel.aisnet.org/icis2024/security/security/6/
18. U. Sa'ad, W. Na, N.N. Dao, S. Cho, Journal of Network and Computer Applications **248**, 104436 (2026). DOI 10.1016/j.jnca.2026.104436. URL https://doi.org/10.1016/j.jnca.2026.104436
19. M. Standen, M. Lucas, D. Bowman, T.J. Richer, J. Kim, D. Marriott, CoRR **abs/2108.09118** (2021). URL https://arxiv.org/abs/2108.09118
20. M. Kiely, M. Ahiskali, E. Borde, B. Bowman, D. Bowman, D. Van Bruggen, K.C. Cowan, P. Dasgupta, E. Devendorf, B. Edwards, A. Fitts, S. Fugate, R. Gabrys, W. Gould, H.H. Huang, J. Jacobs, R. Kerr, I.J. King, L. Li, L. Martinez, C. Moir, C. Murphy, O. Naish, C. Owens, M. Purchase, A. Ridley, A. Taylor, S. Farmer, W.J. Valentine, Y. Zhang, AI Magazine **46**(3), e70021 (2025). DOI https://doi.org/10.1002/aaai.70021. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.70021
21. Baylor Security Lab. SysRepair-Bench: A benchmark for AI agents' ability to remediate real-world system vulnerabilities. GitHub repository (2026). URL https://github.com/BaylorSecurityLab/sysrepair-bench. Verified remote HEAD 4d58d2ca23b01b110aa4c944e3adf11e0733876e on 2026-05-06; README describes 313 system-remediation scenarios with PoC and regression checks
22. C.E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, K. Narasimhan, arXiv preprint arXiv:2310.06770 (2023)
23. X. Deng, J. Da, E. Pan, Y.Y. He, C. Ide, K. Garg, N. Lauffer, A. Park, N. Pasari, C. Rane, K. Sampath, M. Krishnan, S. Kundurthy, S. Hendryx, Z. Wang, V. Bharadwaj, J. Holm, R. Aluri, C.B.C. Zhang, N. Jacobson, B. Liu, B. Kenstler, arXiv preprint arXiv:2509.16941 (2025)
24. J. Yang, K. Lieret, J. Ma, P. Thakkar, D. Pedchenko, S. Sootla, E. McMilin, P. Yin, R. Hou, G. Synnaeve, D. Yang, O. Press. Programbench: Can language models rebuild programs from scratch? (2026)
25. L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W.X. Zhao, Z. Wei, J.R. Wen, Frontiers of Computer Science **18**(6), 186345 (2024). DOI 10.1007/s11704-024-40231-1
26. J.O. Kephart, D.M. Chess, Computer **36**(1), 41 (2003). DOI 10.1109/MC.2003.1160055. URL https://doi.org/10.1109/MC.2003.1160055
27. R.K.L. Ko, in *Emerging Technologies and International Security* (Routledge, 2020). DOI 10.4324/9780367808846-14. URL https://doi.org/10.4324/9780367808846-14
28. P. Cichonski, T. Millar, T. Grance, K. Scarfone, Computer security incident handling guide. Tech. Rep. NIST Special Publication (SP) 800-61 Rev. 2, National Institute of Standards and Technology (2012). URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf
29. M. Souppaya, K. Scarfone, Guide to enterprise patch management technologies. Tech. Rep. NIST Special Publication (SP) 800-40 Rev. 3, National Institute of Standards and Technology (2013). URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r3.pdf
30. H. Karlzen, T. Sommestad, Proceedings of the 18th International Conference on Availability, Reliability and Security (2023). DOI 10.1145/3600160.3605066. URL https://doi.org/10.1145/3600160.3605066
31. M. Souppaya, A. Kerman, K. Scarfone, K. Stine, B. Johnson, C. Peloquin, V. Ruffin, T. Diamond, M. Simos, S. Sweeney, Improving enterprise patching for general IT systems: Utilizing existing tools and performing processes in better ways. Tech. Rep. NIST SP 1800-31, National Institute of Standards and Technology (2022). DOI 10.6028/NIST.SP.1800-31. URL https://doi.org/10.6028/NIST.SP.1800-31
32. A. Nelson, S. Rekhi, K. Scarfone, M. Souppaya, Incident response recommendations and considerations for cybersecurity risk management: A CSF 2.0 community profile. Tech. Rep. NIST SP 800-61 Rev. 3, National Institute of Standards and Technology (2025). DOI 10.6028/NIST.SP.800-61r3. URL https://doi.org/10.6028/NIST.SP.800-61r3
33. A. Magnusson, *Practical Vulnerability Management: A Strategic Approach to Managing Cyber Risk* (No Starch Press, San Francisco, CA, 2020). URL https://nostarch.com/PracticalVulnerability
34. P. Foreman, *Vulnerability Management* (CRC Press, Boca Raton, FL, 2019). URL https://www.routledge.com/Vulnerability-Management/Foreman/p/book/9781138197404
35. M. Salehie, L. Tahvildari, ACM Transactions on Autonomous and Adaptive Systems **4**(2), 14:1 (2009). DOI 10.1145/1516533.1516538. URL https://doi.org/10.1145/1516533.1516538
36. D. Garlan, S.W. Cheng, A.C. Huang, B. Schmerl, P. Steenkiste, Computer **37**(10), 46 (2004). DOI 10.1109/MC.2004.175. URL https://doi.org/10.1109/MC.2004.175
37. J. Jacobs, S. Romanosky, I. Adjerid, W. Baker, Digital Threats: Research and Practice (2021). VERIFY exact volume/issue/article-number before submission
38. FIRST.org, Inc. Common vulnerability scoring system version 4.0: Specification document. https://www.first.org/cvss/v4-0/specification-document (2023). Forum of Incident Response and Security Teams
39. N.O. Jaffal, M. Alkhanafseh, D. Mohaisen, AI **6**(9), 216 (2025). DOI 10.3390/ai6090216. URL https://doi.org/10.3390/ai6090216
40. A. Shahriar, M.N. Rahman, S. Ahmed, F. Sadeque, M.R. Parvez. A survey on agentic security: Applications, threats and defenses (2025). URL https://arxiv.org/abs/2510.06445. Preprint; used only for adjacent-risk context
41. M. Kiely, D. Bowman, M. Standen, C. Moir, arXiv preprint arXiv:2309.07388 (2023). DOI 10.48550/arXiv.2309.07388. URL https://arxiv.org/abs/2309.07388
42. A.D.G. Jenkins, L. Liu, M.K. Wolters, K. Vaniea, in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Association for Computing Machinery, New York, NY, USA, 2024), CHI '24. DOI 10.1145/3613904.3642456. URL https://doi.org/10.1145/3613904.3642456
43. U. Rani, J. Mathew. The recent automating system patching via satellite and puppet integration (2025)
44. J. Polonio, J. Moura, R.N. Marinheiro, IEEE Access **13**, 181957 (2025). DOI 10.1109/ACCESS.2025.3622497. URL https://doi.org/10.1109/ACCESS.2025.3622497
45. M. Compastie, A.L. Martinez, C. Fernandez, M.G. Perez, S. Tsarsitalidis, G. Xylouris, I. Mlakar, M.A. Kourtis, V. Safran, Sensors **23**(3), 1658 (2023). DOI 10.3390/s23031658. URL https://doi.org/10.3390/s23031658

46. J. Paulraj, B. Raghuraman, N. Gopalakrishnan, Y. Otoum, in *2025 IEEE/ACIS 29th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (2025), pp. 925–931. URL https://api.semanticscholar.org/CorpusID:280230639
47. M. Kiely, M. Ahiskali, E. Borde, B. Bowman, D. Bowman, D. van Bruggen, K. Cowan, P. Dasgupta, E. Devendorf, B. Edwards, Proceedings of the AAAI Conference on Artificial Intelligence **39**(28), 28907 (2025). DOI 10.1609/aaai.v39i28.35158
48. Y. Liu, Y. Guo, R. Ranjan, D. Chen, Computing **106**(12), 4015 (2024). DOI 10.1007/s00607-024-01344-4. URL https://doi.org/10.1007/s00607-024-01344-4
49. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K.R. Narasimhan, Y. Cao, in *The Eleventh International Conference on Learning Representations* (2023). URL https://openreview.net/forum?id=WE_vluYUL-X
50. A. Ur-Rehman, I. Gondal, J. Kamruzzaman, A. Jolfaei, Electronics **11**(2), 238 (2022). DOI 10.3390/electronics11020238. URL https://doi.org/10.3390/electronics11020238

A Literature Review Search Queries

This appendix documents the structured literature search executed to identify scholarly work at the intersection of large language models, agentic artificial intelligence, and system vulnerability remediation.

A.1 Databases and Scope

Queries were executed across Google Scholar, Scopus, IEEE Xplore, the ACM Digital Library, Springer Link, ScienceDirect, and arXiv. Grey literature (vendor whitepapers, blog posts, patents, and product manuals) was not pre-filtered at the query stage. Such materials were removed during screening rather than at retrieval.

A.2 Tiered Query Design

The search protocol is organized into five tiers. Each tier reflects a distinct retrieval objective, progressing from foundational context to narrow capability analysis, and concluding with exclusionary filters that suppress source code-level research. The tiered structure was adopted in preference to a single composite query because the target literature spans several disconnected vocabularies (operational security, classical vulnerability management, foundation model research, multi-agent systems), none of which can be expressed in a single Boolean expression without unacceptable losses in either recall or precision.

A.2.1 Tier 0: Foundational Remediation

```
("vulnerability management" OR "vulnerability assessment")
AND ("automated remediation" OR "automated patch management")
AND ("case study" OR "large-scale" OR "enterprise")
```

A.2.2 Tier 1: Foundational LLM Queries (Broad Scope)

```
("large language model" OR "LLM")
AND ("system vulnerability remediation"
     OR "automated patch management")

"foundation models" AND "vulnerability management"
AND ("system misconfiguration" OR "network security")

"AI" AND "autonomous cyber defense"
AND "vulnerability remediation"
```

A.2.3 Tier 2: Agent Focused Queries (Core Research Direction)

```
("LLM agent" OR "agentic AI" OR "multi-agent system")
AND "system vulnerability"
AND ("remediation" OR "patching")

"autonomous agent" AND "large language model"
AND "cybersecurity operations" AND "patch management"

"generative agents" AND "security orchestration"
AND "vulnerability management"
```

A.2.4 Tier 3: Advanced Capability Queries

```
("LLM" OR "large language model")
AND ("reasoning and planning" OR "tool use")
AND "cybersecurity" AND "dynamic environment"

"autonomous agent" AND "tool-use"
AND ("Nessus" OR "OpenVAS" OR "vulnerability scanner")
AND "remediation"

"LLM" AND "reinforcement learning"
AND "autonomous cyber defense" AND NOT "code"
```

A.2.5 Tier 4: Exclusionary Queries (Filtering Source Code Research)

```
"LLM agent" AND "vulnerability remediation"
-("source code" OR "SAST"
  OR "static analysis" OR "code vulnerability")

"automated patching" AND "large language model" AND
"system" NOT ("application code" OR
"software vulnerability")
```

B Acronyms and Abbreviations

Table 15 expands every acronym and abbreviation used in this survey. Entries are listed alphabetically. Where an acronym has more than one common expansion in the literature, the expansion used in this paper is given.

Table 15 Acronyms and abbreviations used in this survey (A–Z)

Acronym	Expansion
ACD	Autonomous Cyber Defense
ACO	Autonomous Cyber Operations
APR	Automated Program Repair
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge (MITRE)
BLEU	Bilingual Evaluation Understudy
CAGE	Cyber Autonomy Gym for Experimentation
CI/CD	Continuous Integration / Continuous Delivery (or Deployment)
CIS	Center for Internet Security
CSOC	Cyber Security Operations Center
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
CybORG	Cyber Operations Research Gym
DevSecOps	Development, Security, and Operations
DOI	Digital Object Identifier
DRL	Deep Reinforcement Learning
GPT	Generative Pre-trained Transformer
IaC	Infrastructure as Code
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LLM	Large Language Model
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
MDP	Markov Decision Process
MFA	Multi-Factor Authentication
MILCOM	Military Communications Conference
MTTR	Mean Time To Remediate (also Repair / Resolution)
NCCoE	National Cybersecurity Center of Excellence
NFV	Network Functions Virtualization
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
OpenVAS	Open Vulnerability Assessment Scanner
OS	Operating System
PEFT	Parameter-Efficient Fine-Tuning
PICOS	Population, Intervention, Comparator, Outcome, Study design
PoC	Proof of Concept
PPO	Proximal Policy Optimization
RAG	Retrieval-Augmented Generation
ReAct	Reasoning and Acting
RL	Reinforcement Learning
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
RQ	Research Question
SAC	Soft Actor-Critic
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SDN	Software-Defined Networking
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation, and Response
SOC	Security Operations Center
SQL	Structured Query Language
SSH	Secure Shell
VM	Virtual Machine
XSS	Cross-Site Scripting
YAML	YAML Ain't Markup Language (recursive acronym)