



YORU: A PRIVILEGE-BOUNDED, CATALOG-CONSTRAINED AI AGENT ARCHITECTURE FOR CONTINUOUS CIS-BASED HARDENING OF UNMANAGED CLOUD SERVERS

Indri Anjar Kartika Sari¹, Onno Widodo Purbo², Abi Julian Saputra Pratama³,
Rahardian Dwi Saputra⁴

¹Program Studi S1 Cyber Security, IDS Digital College

²Institut Teknologi Tangerang Selatan

³Universitas Muhammadiyah Bengkulu

⁴Universitas Muhammadiyah Malang

¹indri.cs2026@ids.ac.id, ²onno@itts.ac.id, ³abijulian@umb.ac.id, ⁴radiandwi2206@gmail.com

Abstract

Small and medium-sized enterprises and independent developers increasingly run production workloads on unmanaged cloud servers yet lack the expertise or budget for continuous hardening. Honeypot studies show internet-facing hosts are attacked within minutes, while CIS benchmarks defining correct hardening contain hundreds of complex, hard-to-automate items. Large language model (LLM) agents introduce a parallel risk: unconstrained tool access lets an agent be manipulated, via prompt injection or reasoning failure, into unauthorized commands – a modern confused-deputy problem. This paper presents Yoru, an AI DevSecOps agent that continuously hardens Ubuntu 24.04 LTS servers against a ten-control CIS-aligned catalog, structurally bounding its reasoning component to a fixed, pre-tested action surface. The architecture separates an LLM layer that prioritizes and explains findings, a static catalog holding all executable commands, and a privilege-bridging dispatcher accepting only a control identifier and one of four fixed actions – bounding a misreasoning or compromised agent to forty pre-vetted, rollback-tested actions rather than arbitrary shell execution. High-risk controls require disclosure-based human approval via a dashboard. We validate the system through exhaustive manual execution of all forty catalog actions, an installer self-test suite, and a network-boundary permission harness, which surfaced four latent defects undetectable through read-only testing, including one control whose apply path was non-functional behind a plausible rejection message. We report these findings, with nine documented limitations, and discuss generalizing the pattern to other domains. The contribution is twofold: an architecture for bounding LLM agent authority, and an empirical case study applying it to unmanaged-server hardening for resource-constrained operators.

Keywords: AI agent security; least privilege; CIS benchmark; DevSecOps; server hardening; human-in-the-loop; confused deputy

Article History

Received: September 2026

Reviewed: September 2026

Published: September 2026

Plagiarism Checker No 235

Prefix DOI :

[10.8734/Kohesi.v1i2.365](https://doi.org/10.8734/Kohesi.v1i2.365)

Copyright : Author

Publish by : Kohesi



This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/)



1. INTRODUCTION

1.1 Background

Cloud computing has made virtual private servers (VPSs) inexpensive and deployable within minutes. Infrastructure once managed by dedicated operations teams is now accessible to individual developers, indie hackers, and small and medium-sized enterprises (SMEs) without formal IT staff. This accessibility is beneficial, but it separates server deployment from the discipline of securing it. Default cloud images are generally not hardened: SSH root login and password authentication may remain enabled, firewall rules may be absent or permissive, and kernel network parameters are typically configured for compatibility rather than defense.

Independent honeypot studies indicate that newly exposed hosts can be detected by automated scanning and attack infrastructure within minutes. The five-minute minimum time-to-compromise figure is drawn from U.S. Patent No. 11,233,804 (2022), while the broader observation that newly exposed hosts encounter hostile activity within minutes is supported by independent honeypot deployments (InfoSecHarry, 2025; Ambrozkiwicz, 2025; Honeypot Software Survey, 2016). These sources are treated here as converging evidence of rapid exposure rather than as directly comparable experiments, because their methodologies and deployment environments differ.

The Center for Internet Security (CIS) Benchmarks provide a widely used community-consensus baseline for hardened configurations. They cover more than one hundred technology families and are mapped to frameworks such as the CIS Critical Security Controls, NIST SP 800-53, and ISO/IEC 27001 (Center for Internet Security, n.d.-a; Center for Internet Security, n.d.-b; Tanium, 2025). Their breadth, however, creates a practical barrier for non-specialists because operating-system benchmarks may contain dozens or hundreds of recommendations requiring both technical interpretation and awareness of operational consequences.

Existing tools reduce part of this burden but remain imperfect for small-scale operators. Canonical's Ubuntu Security Guide (USG) automates CIS Ubuntu auditing and remediation but requires an Ubuntu Pro subscription (Canonical, 2025). Commercial platforms such as Netwrix and Tanium provide continuous CIS compliance monitoring but primarily target enterprise environments (Tanium, 2025; Netwrix, n.d.). Operators managing only a few unmanaged VPS hosts without a dedicated security budget may therefore find these solutions unsuitable.

The target population reflects a documented digital-skills and cybersecurity-capability gap among SMEs and Indonesian MSMEs. Studies of digital-transformation readiness identify limited technical competence, infrastructure access, and cost as barriers (Ghobakhloo et al., 2022; MSME Digital Transformation Readiness Study, 2023). An Indonesian SME digitalization study published in ICMEB Proceedings (2024), citing World Bank data, reports that only seventeen percent of Indonesian SMEs employed staff with adequate digital skills, while more than half identified financial constraints as a barrier to adopting digital tools.

These studies document constraints in digital skills, cybersecurity expertise, staffing, and familiarity with modern security architectures among SMEs and related operator groups. In this study, these constraints provide the context for examining security tooling intended for resource-constrained operators.

Conversational AI does not fully solve the problem because operators must still transcribe commands, execute them in the correct sequence, and understand their effects. Large language model (LLM) agents offer a different model: they can invoke tools, execute commands, and pursue multi-step objectives (Wang et al., 2024; Luo et al., 2025; Schick et al., 2023). In server



administration, an agent could audit a host, interpret findings, prioritize controls, and perform remediation.

This capability creates a second security problem. Granting an LLM agent general-purpose shell or root access makes system safety dependent on the model's future reasoning. Research shows that prompt injection embedded in logs, web content, or files can induce tool-using agents to act outside their intended scope, while over-privileged tools increase potential impact (Shi et al., 2025). This resembles Hardy's confused-deputy problem (Hardy, 1988): a component with excessive authority may be induced to exercise it on an attacker's behalf.

1.2 Problem Statement

This study addresses two overlapping problems. First, continuous CIS-aligned Linux hardening is difficult for individuals and small teams without dedicated security expertise. Second, delegating remediation to an AI agent can introduce privilege-escalation risk when the agent's authority is not bounded independently of its reasoning. Granting broad system privileges may solve the accessibility problem while creating a new security exposure. Removing executive authority, however, returns the system to a chat-assistant model and leaves the diagnosis-to-remediation gap unresolved. The research question is therefore: how should an AI-agent-assisted server-hardening system be designed so that natural-language reasoning supports prioritization, explanation, and risk-aware autonomy, while the worst-case consequences of model error or adversarial manipulation remain limited to a small, enumerable, pre-vetted set of reversible actions?

Three bodies of literature intersect here. LLM-agent-security research identifies the risks of ungoverned agent authority and proposes externally enforced privilege boundaries independent of model judgment (Shi et al., 2025). CIS-automation literature shows both the value of benchmark-based hardening and the residual burden of vendor- or expertise-dependent tools. SME-readiness literature, particularly in Indonesia, documents persistent limitations in technical skills, staffing, and security capability. The authors are not aware of a previously published system that combines these characteristics: LLM-assisted server hardening, an externally enforced and auditable capability catalog, explicit targeting of resource-constrained operators, and disclosed exhaustive action-level validation. This claim is therefore framed as a literature-based research gap rather than the result of a formal systematic literature review.

1.3 Research Objectives and Contributions

This paper documents the design, implementation, and empirical validation of Yoru, a system intended for unmanaged, single-tenant Ubuntu VPS hosts operated by indie developers and SMEs. Yoru was developed in two iterations: an initial hackathon prototype (repository indri007/YORU) and a hardened reference implementation (repository Godzilla129/yoruAgent) introducing a privilege-bridge dispatcher, a risk-classified catalog, an installer self-test suite, and most of the security properties examined here.

The study has five objectives: (1) separate reasoning from authority so that only pre-authorized operating-system actions may execute, even if the reasoning layer is compromised; (2) implement the architecture for a representative subset of the CIS Ubuntu 24.04 LTS Server benchmark, including SSH hardening, firewall policy, patch automation, audit logging, and kernel network parameters; (3) validate all privileged actions through manual execution, rollback testing, and installation-time verification of the privilege boundary; (4) report defects and residual limitations as explicit findings; and (5) examine whether this pattern can generalize to other domains in which LLM agents are granted consequential authority.



The paper makes two contributions. First, it provides an implemented and empirically exercised instance of a catalog-constrained agent architecture for bounding LLM authority, supported by end-to-end validation and disclosed defect history. Second, it presents a case study grounded in the cybersecurity capability gap faced by SMEs, including the Indonesian and broader emerging-market context (IndoSec Summit, 2026; Ghobakhloo et al., 2022; MSME Digital Transformation Readiness Study, 2023; SME Digitalization and Cybersecurity Study, 2025; Bahtiar et al., 2025).

In this paper, “underserved” refers specifically to operators who manage one or a small number of unmanaged servers without dedicated security personnel or a dedicated security budget. Existing CIS-oriented solutions are either subscription-gated, designed for organizations with established security operations, or require familiarity with specialized compliance tooling. This characterization is further supported by SME-readiness literature documenting persistent constraints in technical skills, cybersecurity expertise, staffing, and cost.

2. LITERATURE REVIEW

2.1 LLM Agent Security, Tool Restriction, and the Confused Deputy Problem

The transition of large language models from text generators to autonomous, tool-using agents has been documented in several recent surveys (Wang et al., 2024; Luo et al., 2025; Li et al., 2024). These studies trace the development of LLM agents from early reasoning-acting integration to contemporary frameworks that expose models to thousands of real-world application programming interfaces at runtime (Qin et al., 2023; OpenAI, n.d.). This shift has also introduced security concerns that differ from conventional prompt-only model safety because agentic systems combine autonomy, tool use, and the ability to affect real-world environments (Deng et al., 2024). A recurring concern is that broad tool access increases the consequences of either an ordinary reasoning failure or a successful adversarial injection.

Several strands of this literature are particularly relevant to the architectural decisions discussed in Section 4. Research on over-privileged tool selection reports that agents may select tools with broader authority than required for a task, even when narrower alternatives are available. Evaluations using realistic tool sets have also examined whether existing agent-security benchmarks adequately capture the authority available to an agent.

Related work evaluating agents against realistic tool sets further argues that privilege misuse remains under-measured compared with intent-focused benchmarks such as AgentDojo, Agent Security Bench, and RAS-Eval. These benchmarks primarily examine whether an agent can be manipulated into pursuing a harmful objective, rather than how much authority the agent possesses regardless of its intended goal.

A third line of work addresses the problem more directly by placing an explicit privilege-control layer outside the model. Progent, for example, proposes a programmable policy layer that intercepts and constrains tool calls according to security rules independent of the model's own judgment (Shi et al., 2025). The underlying argument is that model judgment should not serve as the final security boundary when that same judgment may be exposed to adversarial influence.

Other studies extend mandatory-access-control-style reasoning to LLM agent systems, expressing agent security properties through concepts derived from operating-system security rather than model alignment alone (Ji et al., 2026; Siu et al., 2026). A recent position paper similarly argues that LLM agents should be engineered according to established security principles, particularly least privilege, complete mediation, and fail-safe defaults.



This convergence between LLM-agent security and classical systems-security concepts is closely related to the confused deputy problem described by Hardy (1988). In this problem, a program holding authority on behalf of another party may be induced to misuse that authority because intent and authority are combined within the same component. An LLM agent with access to a general-purpose root shell presents a comparable failure mode: manipulation through prompt injection, adversarial content in processed data, or ordinary reasoning errors may directly lead to misuse of privileged capabilities when no independent enforcement boundary exists.

The system presented in this paper applies the Progent/least-privilege line of reasoning to continuous operating-system hardening for unmanaged servers. Rather than relying exclusively on the model to determine both what should be done and what may be executed, the proposed architecture separates these functions through an externally constrained authority mechanism.

2.2 Human-in-the-Loop and Staged Autonomy

A related body of work concerns the calibration of agent autonomy: not only what an agent is technically permitted to do, but also when it should be allowed to act without direct human approval. Collaborative-agent frameworks examine mixed-initiative execution in which humans and agents complete tasks through negotiated handoffs (Collaborative Gym, 2024). In defensive and incident-response contexts, other studies compare fully autonomous systems with human-in-the-loop alternatives in terms of trust, response latency, and the cost of errors (Happer, 2025).

In this study, this literature motivates a staged approach to autonomy. The design uses the consequence and reversibility of an action to determine whether direct human approval is required. Actions with greater potential impact or more difficult rollback therefore receive an explicit human checkpoint. Low-consequence, easily verified, or readily reversible actions may be suitable for automation, whereas actions with a larger blast radius or difficult rollback conditions warrant an explicit human checkpoint. Where approval is required, the potential consequence should be disclosed before consent is given rather than after execution.

This principle informs the risk-tiered approval mechanism evaluated in this paper. Under the proposed design, the catalog, rather than the agent itself, statically assigns the risk tier associated with each control. In this way, the decision about whether human approval is required is separated from the model's runtime judgment.

2.3 CIS Benchmark Automation and Existing Tooling

The CIS Benchmarks are consensus-based, technology-specific hardening baselines maintained by the Center for Internet Security. They span more than one hundred product families and are cross-referenced to broader control and regulatory frameworks, including the CIS Critical Security Controls, NIST, ISO/IEC 27001, and PCI DSS (Center for Internet Security, n.d.-b; Microsoft, n.d.).

The benchmarks also distinguish between Level 1 and Level 2 profiles. Level 1 generally covers foundational recommendations intended to provide a security baseline with relatively limited usability impact. Level 2 introduces additional defense-in-depth controls, including audit-related measures, that may produce greater operational side effects and are therefore intended for environments with higher security requirements (Center for Internet Security, n.d.-c).

Industry analyses have also associated recurring security incidents with configuration weaknesses related to control areas such as asset inventory, patch management, and privilege management (Tanium, 2025). This reinforces a practical distinction between publishing a hardening benchmark and continuously maintaining the configuration states that the benchmark recommends.



The existing CIS automation ecosystem varies considerably in cost, complexity, and intended audience. Canonical's Ubuntu Security Guide (USG) provides CIS auditing and remediation for Ubuntu releases including Ubuntu 24.04 LTS, although the capability is included within the paid Ubuntu Pro subscription tier (Canonical, 2025). The open-source SCAP Security Guide/ComplianceAsCode project provides freely available CIS-aligned profiles through the OpenSCAP toolchain. While this avoids a licensing barrier, operators are still expected to understand SCAP tooling, XCCDF profiles, and the interpretation of scan results (Tech Insider, 2026).

Commercial continuous-compliance platforms such as Netwrix and Tanium combine configuration hardening with continuous monitoring, privileged-access controls, and compliance reporting. Their feature sets are oriented toward organizations with established security operations and procurement capacity (Tanium, 2025; Netwrix, n.d.).

For the single-VPS operator targeted by this paper, these approaches present different practical limitations. Subscription-based products impose a financial requirement, open tooling assumes greater technical familiarity, and enterprise platforms address operational contexts substantially larger than a single unmanaged server. The system proposed here is instead positioned as a free, self-hosted, explanation-oriented alternative for operators who may not already understand the operational implications of individual CIS recommendations.

3. RESEARCH METODOLOGY

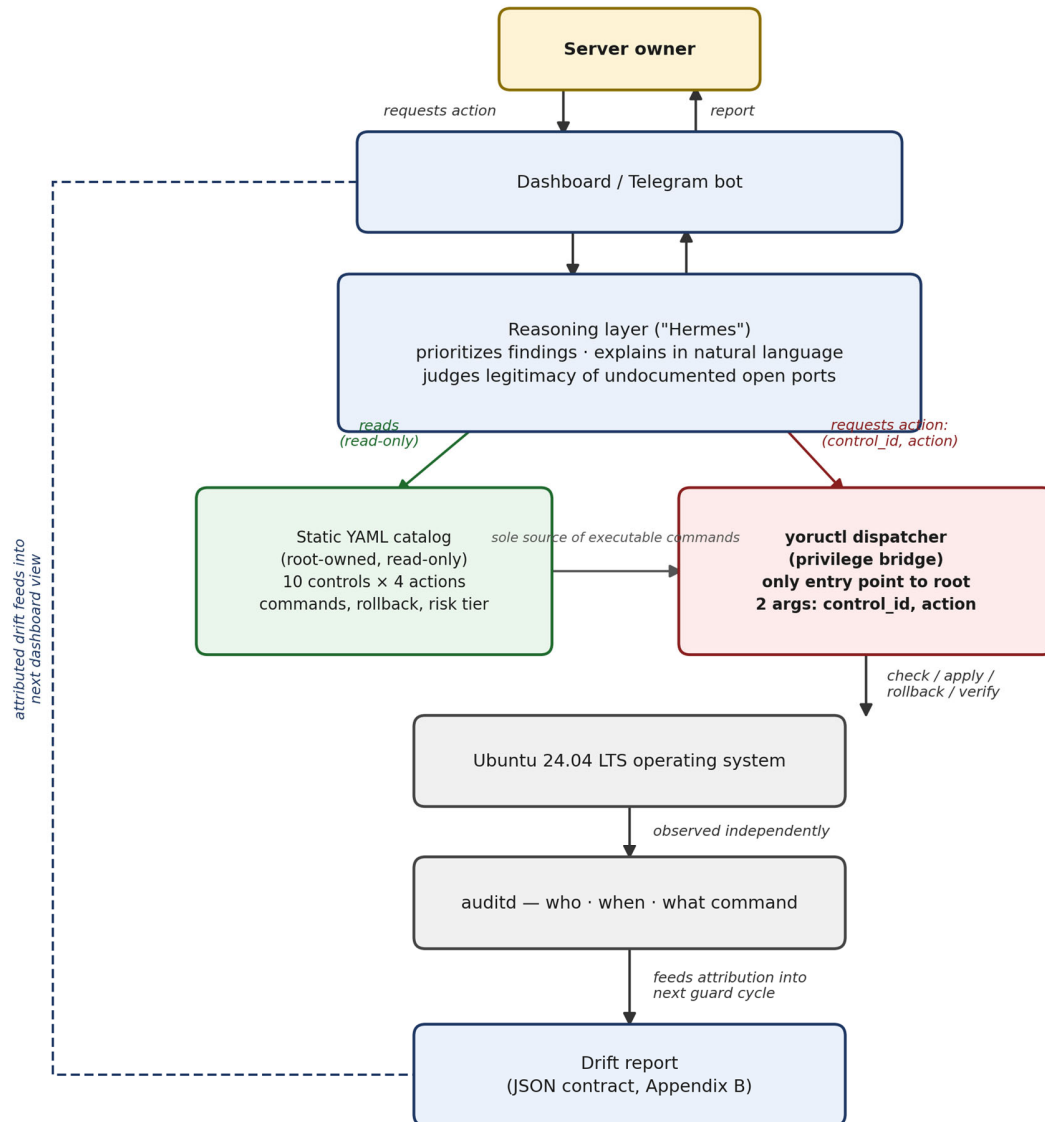
This study adopted a security-oriented systems research methodology to design, implement, and validate Yoru, a privilege-bounded AI agent for continuous hardening of a single Ubuntu 24.04 LTS server. The methodological process began by defining the protected asset, the legitimate server owner, and four relevant adversary categories: remote unauthenticated attackers (A1), adversaries capable of influencing content processed by the reasoning layer (A2), failures originating from the reasoning layer itself (A3), and network-adjacent attackers targeting the dashboard interface (A4).

Based on this threat model, four design principles were derived: privileged authority must remain independent of model correctness (P1); executable actions must originate from static, pre-authored, and individually tested definitions rather than model-generated commands (P2); potentially consequential actions must disclose their effects before human consent is obtained (P3); and each action must be independently verified against the live system state rather than assumed successful from its execution status alone (P4).

These principles were operationalized through an architecture consisting of the Hermes reasoning layer, a static YAML capability catalog, the yoructl privilege-bridging dispatcher, a dashboard and reporting layer, the Ubuntu operating system, and an independent auditd subsystem. The architecture restricts the reasoning layer to reading catalog information and requesting predefined actions, while all root-equivalent execution is mediated by yoructl; the conceptual workflow is illustrated in Figure 1. The reasoning layer was deliberately limited to prioritizing findings, translating technical results into natural-language explanations, and assessing the legitimacy of undocumented open ports. It does not independently generate privileged shell commands.



Figure 1. Conceptual data- and control-flow diagram of the Yoru architecture.



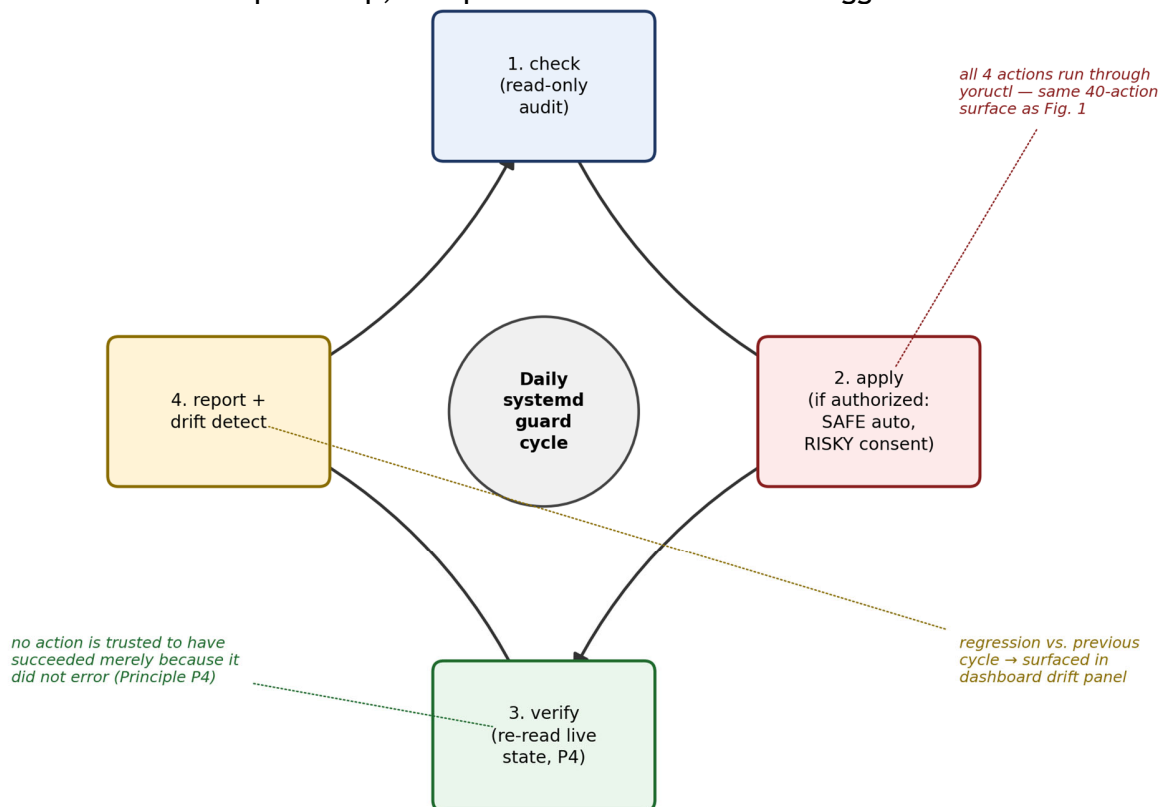
The dispatcher accepts only a control identifier (K01-K10) and one of four actions check, apply, rollback, or verify producing a closed set of forty possible privileged operations. Requests containing unsupported identifiers, additional arguments, or shell metacharacters are rejected. Privilege enforcement is applied through both constrained sudoers permissions and dispatcher-level validation, while the unprivileged yoru-agent account cannot directly modify the root-owned dispatcher or control catalog. The control catalog contains ten CIS-aligned controls covering SSH configuration, firewall policy, open-port restriction, automatic security updates, audit logging, log retention, and kernel network parameters.

Each catalog entry specifies its risk tier, CIS reference, executable action definitions, and, where applicable, the consequences that must be disclosed before execution. The controls were selected to address the A1 threat population while remaining understandable to operators without formal security training. The CIS mappings were derived from the Tenable



CIS_Ubuntu_Linux_24.04_LTS_v1.0.0_L1_Server audit definition rather than directly from the primary CIS benchmark document; this provenance is therefore treated as a methodological limitation rather than represented as formal CIS certification. Continuous operation was implemented through a daily systemd guard cycle following a check → apply → verify → report sequence, with each cycle re-reading the current system state to detect configuration drift, as summarized in Figure 2.

Figure 2. The daily guard cycle: every control passes through the same check → apply → verify → report loop, independent of who or what triggered it.



Validation was conducted on the Ubuntu 24.04 LTS platform targeted by the system. On a clean Ubuntu 24.04 LTS installation, all six installer invariants passed, and re-running the installer on an already-hardened host correctly identified and skipped previously applied steps, providing an additional check of installation idempotency. The installer self-test is designed to execute on every fresh installation, thereby re-evaluating the privilege boundary whenever the system is deployed. The present validation, however, reflects a single development team's testing of a single catalog on one operating-system version and does not include an independent adversarial red-team evaluation. Consequently, the reported results are limited to the tested Ubuntu 24.04 LTS environment and require independent re-validation before being generalized to other platforms or configurations.



4. RESULTS AND DISCUSSION

4.1 Defects Identified Through Action-Level Testing

Applying the validation methodology to the complete forty-action catalog identified four latent defects that would not have been detected through check-only testing. The most consequential defect occurred in the apply path of K02, which disables SSH password authentication. Every invocation of K02 apply returned a rejection, rendering the control effectively non-functional. The rejection message nevertheless resembled the legitimate safety response expected when no verified SSH public key is present. An operator relying only on the apparent plausibility of the message could therefore interpret the behavior as correct enforcement of a safety precondition, although the action was failing independently of whether that precondition had been satisfied.

This finding demonstrates why the `rollback_tested` status should be treated as an individually verified property rather than assumed from a common control template. The remaining three defects were associated with apply or rollback paths and shared a similar failure pattern: the action appeared successful without producing the intended system-state change. Because these failures occurred outside the read-only check path, a validation process limited to configuration inspection would not have revealed them.

4.2 Installer Self-Test and Deployment Reliability

The installer self-test provided a second level of validation. On a clean Ubuntu 24.04 LTS target, all six security invariants described in the implementation passed. Rather than reporting only that installation completed successfully, the installer produced staged outputs corresponding to specific security properties. Re-running the installer on an already-hardened host also correctly detected and skipped previously applied steps, supporting the intended idempotent behavior of the deployment process.

These results complement the action-level tests because they address a different failure surface. While catalog testing examines whether individual hardening operations behave as intended, installer testing verifies whether the privilege boundary and deployment assumptions remain intact when the system is installed. The two forms of validation therefore address catalog correctness and architectural enforcement separately.

4.3 CIS Hardening Coverage and Comparison with Existing Tools

Yoru's ten-control catalog was evaluated in relation to other CIS-hardening approaches using dimensions relevant to the intended single-VPS operator: monetary cost, assumed operator expertise, continuity of enforcement, bounded authority, and availability of natural-language explanation.

Manual application of CIS recommendations is financially free but requires substantial technical knowledge and depends on repeated operator intervention. Ubuntu Security Guide provides automated remediation but requires an Ubuntu Pro subscription, while OpenSCAP/ComplianceAsCode removes the subscription barrier but assumes familiarity with SCAP and XCCDF tooling. Enterprise continuous-compliance platforms such as Netwrix and Tanium provide continuous enforcement and broader organizational capabilities but are designed for environments with established security operations and enterprise licensing. Yoru differs by combining a free self-hosted model, a daily guard cycle, a forty-action bounded dispatcher, and a scoped natural-language reasoning layer.

Yoru currently covers only ten controls and therefore has a much narrower scope than established enterprise compliance platforms. It also lacks features such as extensive role-based access control, enterprise reporting, and broad integration support. These differences reflect the



intended use of the system rather than equivalent product coverage. Yoru was designed for small unmanaged server environments, with emphasis on continuous hardening, understandable explanations, and restricted privileged execution.

4.4 Implications of the Catalog-Constrained Agent Architecture

Yoru separates two functions that are often combined in agent-based systems: deciding what action is appropriate and possessing the authority to execute that action. The reasoning layer may interpret audit results, prioritize findings, and explain recommendations, but it cannot construct arbitrary privileged commands. Execution remains limited to operations defined in the catalog.

This separation reduces the range of actions that can result from an incorrect model decision or manipulated input. However, the four defects identified during testing show that restricting the agent's authority does not guarantee that every authorized operation is correct. The catalog itself remains part of the trusted implementation and must therefore be tested independently.

The architecture is most applicable to tasks in which legitimate privileged operations can be defined before deployment. Similar restrictions could potentially be applied to database maintenance, infrastructure provisioning, or account-remediation workflows, although these use cases would require separate validation.

4.5 Implementation Limitations and Security Tradeoffs

Several limitations remain in the current implementation. First, CIS mappings were obtained from a Tenable audit definition rather than independently verified against the primary CIS benchmark, and the K08 mapping was not cross-checked against the Level 2 benchmark. This restricts formal compliance claims even though the implemented hardening commands remain technically separate from the provenance of their CIS references.

Second, K10 does not address packet-redirect sending, and `rp_filter` was intentionally excluded because strict enforcement may interfere with legitimate asymmetric routing. These issues illustrate two different limitations: an incomplete implementation and a deliberate scope decision based on operational risk.

Other implementation limitations include Telegram notifications that have been tested only in simulation, K05 rollback behavior that clears rather than automatically restores the previous firewall state, K02's inability to parse `AllowGroups` and `DenyGroups`, and a fixed 500 MB log-retention threshold rather than a host-specific calculation.

The system is also limited to operating-system-level hardening. Reverse-proxy configuration, TLS parameters, HTTP security headers, and other application-layer controls remain outside the current catalog. This boundary limits the extent to which Yoru can address the complete attack surface of an internet-facing application.

4.6 Usability, Validity, and Directions for Further Evaluation

The two-step consent mechanism for RISKY controls introduces additional interaction before execution. The design is intended to allow operators to review the concrete consequence of an action before approval, but this usability benefit has not been tested empirically with representative operators. It should therefore be interpreted as a design objective rather than a measured outcome.

The study also has broader validity limitations. Validation was conducted on Ubuntu 24.04 LTS and does not constitute a formal security proof. The reported defect findings originate from a single development team's testing of one catalog, and no independent adversarial red-team evaluation has yet been performed. The results therefore cannot establish that all remaining



defects have been identified or that the catalog behaves equivalently on other operating systems or versions.

Further evaluation should consequently focus on four areas: extending hardening coverage to the web/application layer, verifying CIS mappings against primary benchmark documents, conducting independent adversarial testing of the privilege boundary, and evaluating the consent flow with representative SME and indie-developer operators.

5. CONCLUSION

This study presents Yoru as a privilege-bounded AI agent for continuous hardening of unmanaged Ubuntu 24.04 LTS servers. The system addresses two related problems: the operational difficulty of applying CIS-aligned hardening for users with limited security expertise, and the security risk created when an LLM agent is granted broad execution privileges. Yoru separates reasoning from authority by combining a scoped reasoning layer with a static capability catalog and the `yoructl` dispatcher, restricting privileged execution to a predefined set of forty actions across ten controls. This architecture is intended to ensure that model error or adversarial manipulation cannot directly produce arbitrary system commands. The validation results support the importance of this separation but also show that architectural restriction alone is insufficient. Action-level testing identified four latent defects that would not have been detected through read-only checks, while the installer self-test confirmed the expected privilege-boundary invariants on the tested Ubuntu 24.04 LTS environment. The findings therefore indicate that bounded authority and exhaustive verification should be treated as complementary safeguards. Yoru also occupies a specific practical position among CIS-hardening approaches by combining self-hosted deployment, continuous enforcement, a constrained action surface, and a natural-language explanation layer for resource-constrained operators. At the same time, the study remains limited by secondary-source CIS mapping, incomplete or deliberately excluded controls, unverified notification delivery, rollback and configuration limitations, the absence of application-layer hardening, and the lack of independent red-team and formal usability evaluation. Accordingly, the present results support Yoru as a working and empirically tested implementation of a catalog-constrained agent architecture, while broader claims about generalizability, usability improvement, and cross-platform effectiveness require further validation.

IMPACT AND SOLUTIONS

The implementation suggests that AI-assisted server administration does not require granting an agent unrestricted shell access. For environments in which legitimate privileged operations can be enumerated in advance, a constrained catalog can provide a narrower execution boundary. The testing results also show that this boundary must be accompanied by action-level verification, particularly for remediation and rollback operations. In the current implementation, priorities for improvement include restoring previous firewall states during K05 rollback, verifying CIS mappings against primary benchmark documents, and replacing fixed configuration assumptions such as the 500 MB log-retention threshold with host-specific values.

FURTHER RESEARCH

Further evaluation should extend Yoru in three directions. First, the control catalog should be expanded beyond operating-system hardening to include selected web and application-layer controls such as TLS, reverse-proxy configuration, and HTTP security headers. Second, the existing CIS mappings should be verified against primary CIS benchmark documents and evaluated on



additional Linux environments. Third, the security and usability of the architecture should be tested independently through adversarial evaluation and user studies involving SME or independent-server operators. The latter is particularly important because the current two-step approval mechanism for risky actions has been implemented but has not yet been empirically evaluated with representative users.

REFERENCES

- Ambrozkiwicz, M. (2025). Anatomy of a Linux SSH honeypot attack: Detailed analysis of captured malware. *SANS Internet Storm Center Guest Diary*. <https://isc.sans.edu/diary/32024>
- Bahtiar, Rabbany, Bele, Husna, & Matulesy (2025). Digital transformation towards sustainability: Challenges and opportunities for Indonesian MSMEs. *Jurnal Ekonomi dan Bisnis*, 28(1), 131-150.
- Canonical (2025). Hardening automation for CIS benchmarks now available for Ubuntu 24.04 LTS. *Ubuntu Blog*. <https://ubuntu.com/blog/hardening-automation-for-cis-benchmarks-now-available-for-ubuntu-24-04-lts>
- Center for Internet Security (n.d.-a). *CIS Benchmarks overview*. [cisecurity.org. https://www.cisecurity.org/cis-benchmarks-overview](https://www.cisecurity.org/cis-benchmarks-overview)
- Center for Internet Security (n.d.-b). *CIS Benchmarks*. [cisecurity.org. https://www.cisecurity.org/cis-benchmarks](https://www.cisecurity.org/cis-benchmarks)
- Center for Internet Security (n.d.-c). *Getting to know the CIS Benchmarks*. Center for Internet Security Blog. <https://www.cisecurity.org/insights/blog/getting-to-know-the-cis-benchmarks>
- Collaborative Gym* (2024). Collaborative gym: A framework for enabling and evaluating human-agent collaboration. *arXiv preprint arXiv:2412.15701*.
- Deng, Y., & Inthiran, A. (2026). Towards zero trust architecture: A pilot study on information systems security readiness amongst small and medium enterprises. *arXiv preprint arXiv:2605.18901*.
- Deng, Z., Guo, Y., Han, C., Ma, W., Xiong, J., Wen, S., & Xiang, Y. (2024). AI agent security: Emerging threats and defense paradigms. *arXiv preprint arXiv:2408.29007*.
- Ghobakhloo, M., et al. (2022), as cited in *MSME Digital Transformation Readiness Study* (2023). Micro, small, and medium enterprises' readiness for digital transformation in Indonesia. *Journal of Open Innovation: Technology, Market, and Complexity* (MDPI), 11(6), 156.
- Happer, C. (2025). *Human-in-the-loop vs. fully autonomous AI systems for crisis-driven defense electronics manufacturing*.
- Hardy, N. (1988). The confused deputy: (Or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4), 36-38.
- Honeypot Software Survey* (2016). A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*.
- ICMEB Proceedings (2024). *Proceeding of the International Conference on Management, Entrepreneurship, and Business*. <https://prosiding.arimbi.or.id/index.php/ICMEB/article/download/40/89/357>
- IndoSec Summit (2026). Indonesia's cybersecurity workforce: A skill gap fix. <https://indosecsummit.com/indonesias-cybersecurity-workforce-a-skill-gap-fix/>
- InfoSecHarry (2025). I deployed a honeypot in the cloud – here's what 10+ hours of attacks revealed. *Medium*.



- Ji, Z., et al. (2026). Taming various privilege escalation in LLM-based agent systems: A mandatory access control framework. *arXiv preprint arXiv:2601.11893*.
- Li, Y., et al. (2024). Personal LLM agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*.
- Luo, J., Zhang, W., & Yuan, Y., et al. (2025). Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.
- Microsoft (n.d.). *Center for Internet Security (CIS) Benchmarks*. Microsoft Learn – Compliance Documentation. <https://learn.microsoft.com/en-us/compliance/regulatory/offering-cis-benchmark>
- Netwrix (n.d.). *CIS security benchmarks & CIS configuration guide*. Netwrix Blog. <https://netwrix.com/en/resources/blog/cis-security-benchmarks/>
- OpenAI (n.d.). *Function calling – OpenAI API*. platform.openai.com. <https://platform.openai.com/docs/guides/function-calling>
- Qin, Y., et al. (2023). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.
- Sari, I. A. K., (2026). JobsMatchAI: Platform Generative AI End-to-End untuk Pencocokan Kerja Semantik dan Dukungan Karier di Pasar Tenaga Kerja Indonesia. *Integrative Perspectives of Social and Science Journal*, 3(09 September), 333-340. <https://ipssj.com/index.php/ojs/article/view/2024>
- Sari, I. A. K., Suratnoaji, C., & Widiyarta, A. (2026). Analisis jaringan sosial triliunan rupiah makan bergizi gratis di media sosial X. *Integrative Perspectives of Social and Science Journal*, 3(09 September), 248-257. <https://ipssj.com/index.php/ojs/article/view/2009>
- Sari, I. A. K., Suratnoaji, C., Widiyarta, A., & Kurniawan, R. (2026). Verifying human authorship in AI-mediated communication: A multi-layer framework combining writing behavior, linguistic signals, and AI detection. *Kohesi: Jurnal Sains dan Teknologi*, 11(4). <https://journal.cib.institute/index.php/kohesi/article/view/4403>
- Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Shi, T., He, J., Wang, Z., Wu, L., Li, H., Guo, W., & Song, D. (2025). Progent: Programmable privilege control for LLM agents. *arXiv preprint arXiv:2504.11703*.
- Siu, V., et al. (2026). A framework for formalizing LLM agent security. *arXiv preprint arXiv:2603.19469*.
- SME Digitalization and Cybersecurity Study (2025). Digitalization and cybersecurity concerns in SMEs. In *Springer Nature Book Chapter Series*.
- Tanium (2025). What is CIS compliance? Top controls you can't ignore. *Tanium Blog*. <https://www.tanium.com/blog/what-is-cis-compliance>
- Tech Insider (2026). CIS Benchmarks server hardening: 13 steps. <https://tech-insider.org/cis-benchmarks-server-hardening-2026/>
- U.S. Patent and Trademark Office (2022). *Methods and systems for scalable privacy-preserving compromise detection in the cloud* (U.S. Patent No. 11,233,804).
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18, 186345. <https://doi.org/10.1007/s11704-024-40231-1>



<https://www.jawapos.com/teknologi/2609210065/verifikasi-informasi-mbg-di-media-sosial-kini-didominasi-akun-ai>

<https://portaljtv.com/news/lebih-dari-37-persen-percakapan-makan-bergizi-gratis-di-x-bernada-sindiran-ini-temuan-riset-upn-veteran-jatim?biro=portal-jtv>