International Journal of IJTM Technology & Modeling

INTERNATIONAL JOURNAL OF TECHNOLOGY AND MODELING
Volume 5 Issue 1 Year 2026 Pages 46 - 59
e–ISSN 2964-6847
Url: https://ijtm.my.id

Kernel Security as Process: Vulnerability Analysis and Patching on Ubuntu 16.04 LTS

Annisa Salsabila Aprilya Wijaya [1], Rakhmadi Rahman[2],
Sistem Informasi, Institut Teknologi Bacharuddin Jusuf Habibie
Parepare

*Correspondence to: nissalbila13@gmail.com

Abstract: This study aims to analyze software security vulnerabilities in the Linux kernel of Ubuntu 16.04 and to implement patching solutions that mitigate the identified weaknesses, thereby enhancing system security and stability. The research adopts an experimental approach using a single Ubuntu 16.04. A security audit was conducted using Lynis, an open-source auditing tool, to examine kernel configuration, active kernel modules, file security, and network settings. The audit results were analyzed to identify potential vulnerabilities and to plan patching measures, which included kernel configuration verification, kernel module management, package repository updating, kernel upgrading, system rebooting, and post-update testing. The Lynis audit revealed that the system was generally well-configured, with PAE and NX protections enabled, 58 active kernel modules, and properly set kernel configuration files. However, the kernel version (4.15.0.112) required updating to incorporate the latest security patches. After implementing the patching solutions, the kernel was successfully upgraded to version 4.15.0-142, and post-update testing confirmed that the system operated stably without operational disruption. The study is limited to a single Ubuntu 16.04 system and relies solely on Lynis as the auditing tool; no penetration testing or exploit verification was performed. The findings are therefore not generalizable to other distributions or kernel versions. Nevertheless, the study demonstrates a practical workflow for kernel vulnerability analysis and patching that can be adapted by system administrators. The step-by-step patching procedure covering configuration verification, module management, repository update, kernel upgrade, reboot, and post-update validation provides a replicable guide for maintaining Linux kernel security in operational environments. This study combines vulnerability analysis using Lynis with a practical patching implementation on Ubuntu 16.04, offering an integrated account of the audit-to-patch process and highlighting the importance of continuous security maintenance against evolving threats.

Keywords: Software security, vulnerability analysis, Linux kernel, Lynis, patching, Ubuntu 16.04

Article info: Date Submitted: 05/09/2024 | Date Revised: 02/04/2026 | Date Accepted: 21/04/2026

This is an open access article under the CC BY-SA license

https://doi.org/10.63876/ijtm.v5i1.13

INTRODUCTION

Software security has become a critical concern in the modern digital era[1], [2], as threats continue to grow in sophistication and frequency. Among the various components of a computing system, the operating system kernel occupies a uniquely privileged position: it mediates all interactions between hardware and software, manages system resources, and enforces access control. Consequently, any vulnerability in the kernel can compromise the entire system, making kernel security a top priority for system administrators and security researchers alike. The Linux kernel, despite its reputation for stability and robustness, is not immune to such threats. Its open-source nature, while beneficial for transparency and rapid patching, also means that vulnerabilities are publicly discoverable and can be exploited by malicious actors before patches are widely deployed[3], [4].

Ubuntu 16.04 LTS, one of the most widely adopted Linux distributions in enterprise and educational environments, serves as a relevant case study for examining kernel security practices[5]. Although the distribution has received long-term support, its kernel remains susceptible to vulnerabilities that emerge over time, particularly when security updates are not applied promptly. Identifying these vulnerabilities systematically and implementing appropriate patching solutions are therefore essential steps in maintaining a secure operational environment[6]. However, many organizations still lack a structured approach to vulnerability analysis and patch management, relying instead on reactive measures after incidents occur.

Security auditing tools provide a systematic means of assessing system vulnerabilities [7]. Lynis, an open-source security auditing tool designed for Unix and Linux operating systems, performs comprehensive checks ranging from system configuration and file security to network settings and rootkit detection. Lynis generates detailed reports that include recommendations for remediation, thereby assisting system administrators in maintaining security and compliance with established standards. Despite the availability of such tools, empirical studies that document the complete workflow from audit to patching remain limited, particularly in the context of Indonesian higher education institutions and local operational environments.

Patching, the process of applying software updates to close identified security gaps, is the primary mechanism for remediating kernel vulnerabilities. Kernel patching requires an in-depth understanding of the kernel's structure and the potential impact of changes, as updates can affect system stability and service availability. The process typically involves verifying kernel configuration files, managing active kernel modules, updating package repositories, installing kernel updates, rebooting the system, and conducting post-update testing. Each stage carries risks that must be carefully managed to avoid operational disruption. Nevertheless, the literature emphasizes that timely and well-planned patching, combined with continuous monitoring, is key to maintaining the security of the operating system[8].

Existing studies on Linux security have largely focused on either vulnerability assessment or patch management in isolation, with few offering an integrated account of both processes within a single experimental framework [9], [10]. Furthermore, much of the available literature addresses enterprise-scale deployments or general hardening practices, leaving a gap in practical, step-by-step documentation of kernel vulnerability analysis and patching implementation on specific Ubuntu LTS versions. This gap is particularly significant given that many organizations continue to operate legacy systems that require careful security maintenance.

To address this gap, the present study aims to analyze software security vulnerabilities in the Linux kernel of Ubuntu 16.04 and to implement patching solutions that mitigate the

https://doi.org/10.63876/ijtm.v5i1.13

identified weaknesses. Specifically, the study seeks to: (1) conduct a security audit of the Ubuntu 16.04 kernel using the Lynis tool; (2) identify potential vulnerabilities in kernel configuration, active modules, and related system settings; (3) implement a structured patching procedure encompassing kernel configuration verification, module management, repository updating, kernel upgrading, and post-update testing; and (4) evaluate the effectiveness of the patching solutions in enhancing system security and stability. The findings are expected to contribute both theoretically by documenting an integrated audit-to-patch workflow and practically by providing a replicable guide for system administrators maintaining Linux-based systems in various operational environments.

The remainder of this paper is organized as follows. The next section reviews the literature on software security, the Linux kernel, Lynis, and patching. The subsequent section describes the research methodology, including the experimental design and data collection techniques. The results and discussion section presents the audit findings and the patching implementation, followed by the conclusion, which summarizes the key findings and offers recommendations for future research.

LITERATURE REVIEW

Research on Linux kernel security and vulnerability remediation spans several interconnected domains: security auditing tools, kernel patching practices, and the operational challenges of maintaining legacy systems [11]. This section reviews the relevant literature across these areas, identifying the gaps that the present study addresses.

Security Auditing with Lynis

Lynis has established itself as a widely adopted open-source tool for host-based security auditing on Unix and Linux systems[12]. Its design philosophy centers on inspecting a live system rather than static files, examining kernel settings, authentication configurations, logging, package states, and file permissions [13]. This live-system orientation makes Lynis particularly suitable for identifying configuration weaknesses that may not be captured by network-based scanners.

Recent empirical work has begun to quantify Lynis's utility in structured assessment frameworks. A study deploying Ubuntu nodes on the FABRIC testbed conducted 108 audits using Lynis alongside OpenSCAP and AIDE, demonstrating that aggregating heterogeneous security metrics yields a clearer assessment of system posture than analyzing individual tools in isolation. The authors selected Lynis specifically to cover configuration auditing, complementing compliance validation and integrity monitoring. In a different experimental context, Lynis was used to evaluate the security posture of Ubuntu Server across mail server and web server scenarios, with findings highlighting its capacity to identify issues such as insecure file permissions, vulnerable packages, and service misconfigurations . A pre-test/post-test design on Ubuntu 24.04 LTS reported Lynis Hardening Index improvements from 60 to 83 following CIS Benchmark implementation, alongside the elimination of OpenVAS-detected vulnerabilities [14].

However, these studies share a common limitation: they treat Lynis primarily as a scoring or diagnostic instrument, rarely following through to a complete patching implementation. The transition from audit findings to remediated system state remains underexplored in the empirical literature.

https://doi.org/10.63876/ijtm.v5i1.13

Linux Kernel Patching and Patch Porting

The Linux kernel ecosystem presents unique challenges for patch management due to its distributed maintenance model [15]. Li et al. investigated patch porting practices, characterizing how fixes propagate from the mainline kernel to stable and long-term-support (LTS) branches [16]. Their work highlights the complexity introduced by version divergence: patches developed for one kernel version often require adaptation before they can be applied to older branches. This finding has direct implications for systems running LTS kernels, where security fixes may arrive with delays or require backporting.

Related research has focused on automating the backporting process. TSBPORT, a patch type-sensitive backporting technique, achieved an 87.59% success rate on a dataset of 1,815 Linux kernel patch pairs, outperforming FIXMORPH by 62.96% . While such tools improve the efficiency of patch propagation, they operate at the level of kernel developers and maintainers rather than system administrators. The operational side how end-users apply available patches and verify their effectiveness receives less attention[17].

More recent empirical analysis has examined the temporal dynamics of kernel vulnerability remediation. A study of Linux kernel CVE data found that fixes arrive faster in newer kernel branches, leading the authors to conclude that "moving to recent LTS or stable kernel versions significantly reduces the exposure to vulnerability windows" [18]. Notably, the same study found that CVSS scores have limited predictive power for actual patch urgency in the kernel ecosystem, recommending that organizations reassess triage processes that rely solely on severity scores. This finding challenges the common assumption that vulnerability scoring alone can guide patch prioritization.

Patch Management in Operational Contexts

The broader literature on patch management emphasizes process discipline over tool selection. NIST SP 800-40 defines enterprise patch management as encompassing identification, prioritization, acquisition, installation, and verification of patches. The verification step is particularly relevant: an installer returning a success code constitutes execution evidence, not proof that the vulnerability has been remediated or the service remains healthy. This distinction is often overlooked in practical implementations.

For Linux systems specifically, the multi-mechanism nature of updates operating system repositories, vendor advisories, application release channels creates coordination challenges [19]. The risk of exploitation increases as time passes between vulnerability disclosure and patch application, yet organizations must also contend with side effects, testing requirements, and downtime constraints. Ubuntu 16.04 LTS, the subject of the present study, reached end of standard support in April 2021, and subsequent security updates have been delivered through Ubuntu Pro/ESM . This context is critical: systems operating beyond standard support face a narrowing window for receiving and applying patches through conventional channels.

Research Gap

The reviewed literature reveals a fragmented landscape. Security auditing studies demonstrate the diagnostic value of tools like Lynis but rarely extend into systematic remediation. Kernel patching research addresses the technical mechanics of backporting and propagation but operates largely at the developer rather than administrator level. Patch management frameworks provide normative guidance but lack empirical grounding in specific operational contexts, particularly for legacy Ubuntu LTS deployments.

https://doi.org/10.63876/ijtm.v5i1.13

What remains absent is an integrated account that documents the complete workflow from audit through patching to post-remediation verification on a specific legacy system. The present study addresses this gap by combining Lynis-based vulnerability analysis with a structured patching implementation on Ubuntu 16.04, offering a replicable reference for administrators maintaining similar systems. In doing so, it responds to the observation that kernel security research often "emphasize[s] either detection or patch management at a process level" without synthesizing the two under a unified protocol .

METHODS

Research Design

This study employed an experimental research design to analyze software security vulnerabilities in the Linux kernel and to implement patching solutions addressing the identified weaknesses. The experimental approach was chosen because it allows the researcher to manipulate system conditions (auditing, patching) and observe the resulting changes in system state (kernel version, security posture) under controlled conditions. The design followed a single-system, pre-test–post-test structure: the system was audited in its initial state, patched according to the audit findings, and then re-examined to assess the outcome.

Research Subject and Environment

The subject of this research was a single computer system running Ubuntu 16.04 LTS (Xenial Xerus) with the generic kernel branch. Ubuntu 16.04 was selected for three reasons: (1) its widespread adoption in enterprise, educational, and institutional environments; (2) its status as a long-term support release, which makes it a representative case for legacy system maintenance; and (3) its relevance to the operational context in which the researchers work. The initial kernel version observed was 4.15.0-112-generic, which is the hardware enablement (HWE) kernel branch for Ubuntu 16.04.

The system was configured with standard networking access to official Ubuntu repositories (security.ubuntu.com, us.archive.ubuntu.com) to enable package retrieval. Administrative access was obtained through sudo privileges, which were required for auditing, module inspection, and kernel installation. No additional hardening tools or third-party repositories were introduced, so that the observed baseline reflects a typical default installation.

Data Collection Techniques

Data collection proceeded through three complementary techniques: installation and configuration recording, security auditing, and command-line inspection.

- Security auditing with Lynis. The primary data collection instrument was Lynis, an open-source host-based security auditing tool for Unix and Linux systems. Lynis was selected because it performs live-system inspection examining kernel parameters, authentication settings, logging configuration, file permissions, and package states and produces a structured report with warnings, suggestions, and a hardening index. The audit was executed with root privileges to allow full inspection of system components. The output was captured in full, including the sections on kernel configuration, kernel modules, and file system security, and served as the basis for identifying vulnerabilities and planning remediation.
- Kernel module inspection. Active kernel modules were enumerated using the lsmod command, which lists all currently loaded modules together with their memory footprint and dependency relationships (the "Used by" column). This data was used to determine

https://doi.org/10.63876/ijtm.v5i1.13

which modules were operationally necessary and which represented unnecessary attack surface.

- Version and configuration verification. The running kernel version was recorded using uname -r. Kernel configuration files were located and verified, typically under /boot/config-$(uname -r), to confirm that security-relevant settings were present and correctly configured.

Data Analysis

Audit data were analyzed in three stages. First, the Lynis report was reviewed section by section to separate informational findings from actionable warnings and suggestions. Particular attention was paid to kernel-related items: run level, CPU security features (PAE and NX), kernel version status, module count, configuration file presence, I/O scheduler detection, and core dump settings. Second, each actionable finding was assessed for risk and impact, and findings were mapped to specific remediation actions. Third, the remediation actions were sequenced into a patching plan, ordered from least to most disruptive: configuration verification, module management, repository update, kernel upgrade, reboot, and post-update testing.

Patching Implementation Procedure

The patching procedure was implemented in six stages, following the plan derived from the audit analysis.

1. Kernel configuration verification. The kernel configuration file was checked for presence and correctness, ensuring that security-relevant settings aligned with the system's operational requirements.
2. Active kernel module management. The module list obtained from lsmod was reviewed to identify modules that were loaded but not required for the system's function. Unnecessary modules were flagged for disabling in order to reduce the attack surface and improve performance.
3. Package repository update. The package index was refreshed using sudo apt-get update, retrieving the latest package metadata from the configured Ubuntu repositories. This step ensured that the system had current information about available kernel packages and security updates.
4. Kernel upgrade. Available updates were installed using sudo apt-get upgrade, which included the kernel package and its dependencies. The installation output was recorded to confirm which packages were upgraded and to identify any dependency changes.
5. System reboot. The system was restarted using sudo reboot to activate the newly installed kernel. A reboot is mandatory for kernel upgrades because the running kernel cannot be replaced in memory; the system must boot into the new image for the changes to take effect.
6. Post-update testing. After reboot, the kernel version was re-checked with uname -r to confirm that the system had booted into the updated kernel. System functionality was then verified to ensure that no operational disruption had occurred and that services were running normally.

https://doi.org/10.63876/ijtm.v5i1.13

Evaluation Criteria

The effectiveness of the patching implementation was evaluated against three criteria: (1) successful activation of the updated kernel, verified by version comparison before and after the procedure; (2) absence of operational disruption, verified through post-reboot functional testing; and (3) alignment between the implemented actions and the remediation recommendations derived from the Lynis audit. It should be noted that the study did not perform exploit verification or a second full Lynis audit after patching; these limitations are acknowledged and discussed in the concluding section.

RESULT AND DISCUSSION

Overview of Audit Findings

The Lynis audit was executed on the Ubuntu 16.04 system in its initial state, with the kernel running at version 4.15.0-112-generic. The audit examined the system across multiple dimensions, including kernel configuration, boot and services, authentication, logging, file permissions, and network settings. This section presents the kernel-relevant findings, interprets their security significance, and discusses the patching implementation that followed.

Kernel Vulnerability Analysis

System Run Level and CPU Security Features

The audit reported that the system was operating at run level 5, which corresponds to graphical multi-user mode. From a security standpoint, run level 5 implies that a display manager and graphical stack are active, expanding the attack surface relative to a headless server configuration [20]. For systems that do not require a graphical interface, operating at run level 3 would reduce exposure by eliminating graphical services, display managers, and associated libraries from the running environment.

The CPU was confirmed to support both PAE (Physical Address Extension) and NX (No-Execute) features. The NX bit is particularly significant: it enables hardware-enforced marking of memory pages as non-executable, which mitigates a broad class of memory-corruption exploits that attempt to execute injected code from data regions. PAE allows 32-bit systems to address more than 4 GB of physical memory, which is relevant for the 32-bit Ubuntu 16.04 installation examined here. The presence of both features indicates that the hardware foundation for memory exploit mitigation was in place. However, it should be noted that NX support at the hardware level is only effective if the kernel is configured to use it a condition that the audit confirmed.

Kernel Version Status

The audit reported the kernel as "up-to-date and optimized for security," a finding that requires careful interpretation. Lynis determines kernel version status by comparing the installed kernel against the latest version available in the configured repositories at the time of the audit. The finding therefore means that no newer kernel package was pending in the repository index not that the kernel was free of known vulnerabilities. This distinction is critical: a kernel can be the newest available package in a repository while still containing vulnerabilities that have been patched in newer upstream releases or in other distribution channels[21].

This interpretation is supported by the subsequent patching outcome. After the package index was refreshed with sudo apt-get update, the system reported that 192 packages could be

https://doi.org/10.63876/ijtm.v5i1.13

upgraded, and the kernel was subsequently updated from 4.15.0.112 to 4.15.0-142. The discrepancy between the audit's "up-to-date" assessment and the availability of a substantially newer kernel (30 patch levels ahead) indicates that the repository index on the system was stale at the time of the audit. This is a meaningful finding in itself: it demonstrates that Lynis's kernel version check is only as current as the system's package metadata, and that a stale index can produce a false sense of security. System administrators should therefore refresh the package index before relying on version-status findings from any auditing tool.

Active Kernel Modules

The audit identified 58 active kernel modules. Inspection with lsmod revealed a typical Ubuntu desktop module set, including networking drivers (e1000, pcnet32), filesystem modules (ext4), cryptographic modules (crypto_simd, ghash_clmulni_intel, aesni_intel, cryptd), and virtualization support modules (vmw_balloon, vmw_vmci, vmw_vsock_vmci_transport). The presence of VMware guest modules indicates that the system was running as a virtual machine, which is consistent with an experimental test environment.

From a security perspective, each loaded module represents code executing in kernel space with full privileges, and therefore contributes to the attack surface. The principle of least functionality suggests that modules not required for the system's operation should be unloaded or blacklisted. However, the audit did not produce a definitive list of unnecessary modules, and the module set observed is largely consistent with a standard Ubuntu installation. The practical value of module management in this context is therefore modest: the modules present are common and well-maintained, and unloading them without a clear operational justification could introduce instability. The more consequential module-related risk lies in modules that are auto-loaded on demand but never used a category that requires runtime observation over time rather than a single snapshot. This limitation is acknowledged: a one-time lsmod inspection cannot distinguish between modules that are genuinely needed and those that were loaded incidentally and remain resident.

Kernel Configuration Files and Core Dump Settings

The audit confirmed the presence and proper configuration of kernel configuration files under /boot/config-$(uname -r). The kernel I/O scheduler was detected, and core dump configuration was found to be disabled, with setuid core dumps protected. These findings are favorable: disabled core dumps prevent the inadvertent capture of sensitive memory contents (such as cryptographic keys or credentials) in crash artifacts, and protected setuid core dumps prevent privilege escalation through core dump manipulation. The audit also confirmed that no reboot was pending at the time of the audit a status that would change after the kernel upgrade.

Summary of the Initial Security Posture

Taken together, the audit findings indicate a system with a sound baseline configuration: hardware exploit mitigations were active, core dump handling was appropriately restrictive, and configuration files were present and correct. The principal actionable finding was not a misconfiguration but a maintenance gap: the kernel was 30 patch levels behind the version available in the repositories, and the package index was stale. This finding reframes the nature of the vulnerability: the primary risk was not a flaw in the system's configuration but the absence of timely patch application a process failure rather than a configuration failure.

https://doi.org/10.63876/ijtm.v5i1.13

Implementation of Patching Solutions

The patching implementation followed the six-stage procedure described in the Methods section. This subsection reports the outcome of each stage and discusses its significance.

Stage 1: Kernel Configuration Verification

The kernel configuration file was verified at /boot/config-$(uname -r), confirming its presence and correctness. This verification is a precondition for confident patching: if the configuration file were missing or malformed, the system would boot with default settings after the kernel upgrade, potentially reverting security-relevant options. The verification therefore served as a safety check rather than a remediation action in its own right.

Stage 2: Active Kernel Module Management

The module inventory obtained from lsmod was reviewed. As discussed above, the module set was consistent with a standard Ubuntu installation, and no modules were identified as clearly unnecessary and safe to remove without further runtime observation. The outcome of this stage was therefore a documented baseline rather than an active remediation. This result is itself instructive: module management is often presented as a straightforward hardening step, but in practice it requires operational knowledge of which functions the system performs and which modules support them knowledge that a one-time audit snapshot cannot supply.

Stage 3: Package Repository Update

The package index was refreshed with sudo apt-get update, which retrieved metadata from the configured repositories: security.ubuntu.com (xenial-security), us.archive.ubuntu.com (xenial, xenial-updates, xenial-backports). The command output reported that 192 packages could be upgraded. This stage is the pivotal point in the patching workflow: it is where the stale index identified in the audit analysis was corrected, and it is the stage that revealed the true extent of the maintenance gap. The 192 upgradable packages included not only the kernel but also security-relevant userland components, indicating that the system had not been updated for a considerable period.

Stage 4: Kernel Upgrade

The upgrade was performed with sudo apt-get upgrade. The installation log confirmed the upgrade of the kernel package from 4.15.0.112 to 4.15.0.142, together with associated packages including linux-headers-4.15.0-142, linux-headers-4.15.0-142-generic, linux-image-4.15.0-142-generic, linux-modules-4.15.0-142-generic, and linux-modules-extra-4.15.0-142-generic. The upgrade also updated a large set of userland packages, including OpenSSL (libssl1.1), the GNU C Library (libc6), and various system utilities. From a security standpoint, the OpenSSL and glibc updates are as significant as the kernel update, since vulnerabilities in these libraries can be exploited through userland vectors that do not require kernel privileges. This observation reinforces a point often understated in kernel-focused security studies: kernel patching is necessary but not sufficient, and a complete remediation must address the full dependency chain.

It should be noted that apt-get upgrade was used rather than dist-upgrade. The former will not remove or install packages to resolve dependency changes, which makes it a more conservative choice but also means that certain dependency transitions may remain unresolved. For the kernel upgrade observed here, the upgrade completed successfully, and the new kernel image and modules were installed as expected.

https://doi.org/10.63876/ijtm.v5i1.13

Stage 5: System Reboot

The system was rebooted with sudo reboot. A reboot is mandatory for kernel upgrades because the running kernel image cannot be replaced in memory; the system must boot into the new image for the patched code to execute. This stage carries the greatest operational risk in the entire procedure, because a failed boot would render the system unavailable. Mitigation for this risk includes verifying that the bootloader configuration includes a fallback entry for the previous kernel, and ensuring console or out-of-band access is available in case the new kernel fails to boot. In this implementation, the reboot completed successfully and the system returned to normal operation.

Stage 6: Post-Update Testing

After reboot, the kernel version was re-checked with uname -r, which reported 4.15.0-142-generic, confirming that the system had booted into the updated kernel. Functional testing confirmed that the system operated normally, with no observed disruption to services or system behavior. This verification satisfies two of the three evaluation criteria defined in the Methods section: successful activation of the updated kernel, and absence of operational disruption.

The third criterion alignment between implemented actions and audit recommendations was satisfied by construction, since the patching plan was derived directly from the audit findings. However, it must be acknowledged that no second full Lynis audit was performed after patching, and no exploit verification was conducted. The claim that the system's security posture improved therefore rests on the version change and the successful boot, not on a measured reduction in audit findings. This is a meaningful limitation: a post-patching audit would have confirmed whether the update resolved the specific items flagged in the initial audit, and would have detected any new findings introduced by the updated packages.

Discussion

Interpretation of the Central Finding

The central empirical finding of this study is that the system's principal security weakness was a stale package index masking a 30-patch-level kernel gap, rather than a configuration defect. This finding has a practical implication that extends beyond the specific system examined: vulnerability assessment tools that rely on repository metadata are only as reliable as that metadata, and an audit performed against a stale index can produce a falsely reassuring result. The recommended practice is therefore to refresh the package index before auditing, or to supplement tool-based assessment with direct comparison against upstream security advisories and CVE databases. This aligns with the observation in the kernel vulnerability literature that exposure windows are governed less by the availability of patches than by the speed with which they are applied.

The Limits of Version-Based Assessment

The study also illustrates the limits of version-based security assessment. Updating the kernel from 4.15.0.112 to 4.15.0-142 incorporates the security fixes released in the intervening 30 patch levels, but it does not follow that every known vulnerability affecting 4.15.0.112 is resolved, nor that 4.15.0-142 is free of vulnerabilities. Kernel security is a moving target: new vulnerabilities are discovered continuously, and the relevant question is not whether a given version is "secure" in absolute terms but whether the system is receiving patches within an acceptable window. Framing security as a version milestone as the original audit report's "up-to-date" language implicitly does obscures this dynamic. A more accurate framing is process-

https://doi.org/10.63876/ijtm.v5i1.13

based: the system's security depends on the cadence of index refreshes, patch application, and reboot scheduling.

Ubuntu 16.04 and the End-of-Life Context

A further consideration, absent from the original analysis but essential to an honest interpretation, is the lifecycle status of Ubuntu 16.04. Standard support for Ubuntu 16.04 LTS ended in April 2021, and security maintenance thereafter has been delivered through Ubuntu Pro/ESM rather than the standard repositories. A system relying solely on the standard xenial-security repository after that date would not receive all security updates, and the kernel version reached in this study (4.15.0-142) reflects the state of the standard archive at the time of the experiment rather than the full set of fixes available through extended support. This context qualifies the study's conclusion: the patching procedure demonstrated here is sound as a workflow, but its security value on an end-of-life release is bounded by the channel through which patches are delivered. Organizations maintaining Ubuntu 16.04 systems should verify their entitlement to extended security maintenance, or plan migration to a supported release.

Implications for Practice

The patching procedure documented in this study offers a replicable reference for administrators maintaining legacy Linux systems. Its key elements are: refresh the package index before auditing; verify kernel configuration files before upgrading; inspect loaded modules to establish a baseline; apply upgrades and record the package changes; reboot to activate the new kernel; and verify the new version and system functionality after reboot. Two additions would strengthen the procedure in operational settings: a post-patching audit to confirm remediation, and a rollback plan covering bootloader fallback and configuration backup. The study also suggests that patch management should be scheduled as a recurring process rather than a one-time remediation, since the maintenance gap observed here accumulated over an extended period of neglect.

Limitations

This study has several limitations that constrain the generalizability of its findings. First, it examined a single system running a single distribution version, so the results may not transfer to other distributions, kernel branches, or hardware configurations. Second, the audit relied on a single tool, Lynis; triangulation with a second scanner (for example, OpenSCAP or a vulnerability scanner with CVE matching) would have provided independent confirmation of the findings. Third, no exploit verification was performed, so the study cannot demonstrate that the vulnerabilities addressed were exploitable before patching or definitively closed afterward. Fourth, no post-patching Lynis audit was conducted, meaning the improvement in security posture is inferred from the version change rather than measured. Fifth, the module management stage produced a baseline rather than an active remediation, and a longitudinal observation of module loading would be required to identify genuinely unnecessary modules. Finally, the end-of-life status of Ubuntu 16.04 limits the security value of patching through standard repositories, as discussed above.

Directions for Future Research

Future work could address these limitations in several ways: replicating the procedure across multiple distributions and kernel branches to test generalizability; incorporating a pre-test–post-test Lynis hardening index as a quantitative measure of remediation effectiveness; adding exploit verification in a controlled laboratory environment to confirm that patching closes the targeted vulnerabilities; and extending the workflow to include automated patch

https://doi.org/10.63876/ijtm.v5i1.13

management and configuration management tooling, which would allow the procedure to scale beyond single-system administration. A comparative study of patching cadence across organizational contexts would also help identify the process factors that most strongly influence exposure windows.

CONCLUSION

This study analyzed software security vulnerabilities in the Ubuntu 16.04 Linux kernel using Lynis and implemented patching solutions to address the identified weaknesses. The audit revealed a fundamentally sound baseline configuration PAE and NX protections active, restrictive core dump handling, and correctly configured kernel files but exposed a maintenance gap rather than a configuration defect: the kernel was 30 patch levels behind the repository version, concealed by a stale package index that had produced a misleading "up-to-date" verdict. The patching procedure, comprising configuration verification, module inventory, repository refresh, kernel upgrade, reboot, and post-update testing, successfully upgraded the kernel from 4.15.0.112 to 4.15.0-142 with no operational disruption. The study's central lesson is that kernel security is a process rather than a state: the decisive variable is not the version observed at any moment but the regularity with which systems are refreshed, patched, and verified. However, findings are limited by the single-system scope, reliance on one auditing tool, absence of exploit verification and post-patching audit, and the end-of-life status of Ubuntu 16.04. Continuous maintenance discipline not periodic remediation remains the posture that protects systems against evolving threats.

REFERENCES

[1] Y. Mothanna, W. ElMedany, M. Hammad, R. Ksantini, and M. S. Sharif, “Adopting security practices in software development process: Security testing framework for sustainable smart cities,” *Computers & Security*, vol. 144, p. 103985, Sep. 2024, doi: https://doi.org/10.1016/j.cose.2024.103985.

[2] C. Glasauer, L. Maurer, C. Spreitzer, and R. W. Alexandrowicz, “Development & psychometrics of the SOLID-S – An inventory assessing software security culture in software development companies,” *Computers & Security*, vol. 140, p. 103753, May 2024, doi: https://doi.org/10.1016/j.cose.2024.103753.

[3] G. Duan, Y. Fu, M. Cai, H. Chen, and J. Sun, “DongTing: A large-scale dataset for anomaly detection of the Linux kernel,” *Journal of Systems and Software*, vol. 203, p. 111745, Sep. 2023, doi: https://doi.org/10.1016/j.jss.2023.111745.

[4] D. B. De Oliveira, R. S. De Oliveira, and T. Cucinotta, “A thread synchronization model for the PREEMPT_RT Linux kernel,” *Journal of Systems Architecture*, vol. 107, p. 101729, Aug. 2020, doi: https://doi.org/10.1016/j.sysarc.2020.101729.

[5] K. Wang *et al.*, “Characterizing and optimizing Kernel resource isolation for containers,” *Future Generation Computer Systems*, vol. 141, pp. 218–229, Apr. 2023, doi: https://doi.org/10.1016/j.future.2022.11.018.

[6] N. Cecchinato, A. Toma, C. Drioli, G. Oliva, G. Sechi, and G. L. Foresti, “A Secure Real-time Multimedia Streaming through Robust and Lightweight AES Encryption in UAV Networks for Operational Scenarios in Military Domain,” *Procedia Computer Science*, vol. 205, pp. 50–57, 2022, doi: https://doi.org/10.1016/j.procs.2022.09.006.

[7] S. Kaimov, T. Kaiym, N. Kanagatova, M. Kalimoldayev, A. Kaimov, and T. Mazakov, “Enhancing information Security Audits in Enterprise Systems with

https://doi.org/10.63876/ijtm.v5i1.13

a Recommendation Generation Process," *Procedia Computer Science*, vol. 251, pp. 585–590, 2024, doi: https://doi.org/10.1016/j.procs.2024.11.153.
[8] P. Sun, Y. Wan, Z. Wu, and Z. Fang, "A survey on security issues in IoT operating systems," *Journal of Network and Computer Applications*, vol. 231, p. 103976, Nov. 2024, doi: https://doi.org/10.1016/j.jnca.2024.103976.
[9] S. S. Karim, M. Afzal, W. Iqbal, and D. A. Abri, "Advanced Persistent Threat (APT) and intrusion detection evaluation dataset for linux systems 2024," *Data in Brief*, vol. 54, p. 110290, Jun. 2024, doi: https://doi.org/10.1016/j.dib.2024.110290.
[10] E. Freitas, A. T. De Oliveira Filho, P. R. X. Do Carmo, D. Sadok, and J. Kelner, "A survey on accelerating technologies for fast network packet processing in Linux environments," *Computer Communications*, vol. 196, pp. 148–166, Dec. 2022, doi: https://doi.org/10.1016/j.comcom.2022.10.003.
[11] W. Zheng, P. Deng, K. Gui, and X. Wu, "An Abstract Syntax Tree based static fuzzing mutation for vulnerability evolution analysis," *Information and Software Technology*, vol. 158, p. 107194, Jun. 2023, doi: https://doi.org/10.1016/j.infsof.2023.107194.
[12] B. Seth *et al.*, "Secure Cloud Data Storage System using Hybrid Paillier-Blowfish Algorithm," *Computers, Materials & Continua*, vol. 67, no. 1, pp. 779–798, 2021, doi: https://doi.org/10.32604/cmc.2021.014466.
[13] Nabila Zahrani and Siti Amelia, "Population Data Collection System in the Digital Era by Utilizing the Advantages of the Web to Improve Data Quality," *ijss*, vol. 1, no. 2, pp. 56–72, Jun. 2023, doi: https://doi.org/10.63876/ijss.v1i2.9.
[14] P. S. S. K. Gandikota, D. Valluri, S. B. Mundru, G. K. Yanala, and S. Sushaini, "Web Application Security through Comprehensive Vulnerability Assessment," *Procedia Computer Science*, vol. 230, pp. 168–182, 2023, doi: https://doi.org/10.1016/j.procs.2023.12.072.
[15] M. Pagani, A. Biondi, M. Marinoni, L. Molinari, G. Lipari, and G. Buttazzo, "A Linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration," *Future Generation Computer Systems*, vol. 129, pp. 125–140, Apr. 2022, doi: https://doi.org/10.1016/j.future.2021.11.007.
[16] R. R. Kouchaksaraei, A. Jabbarzadeh, and M. Narenji, "A robust optimization approach to multi-period competitive location problem for bank branches considering first-mover advantage," *Computers & Industrial Engineering*, vol. 190, p. 110008, Apr. 2024, doi: https://doi.org/10.1016/j.cie.2024.110008.
[17] A. Syifani, M. Musyarofah, and T. R. Firdaus, "A Lightweight Interpolation Framework for Real-Time Travel Time Estimation with Incomplete Traffic Observations," *IJTM*, vol. 4, no. 3, pp. 183–192, Dec. 2025, doi: https://doi.org/10.63876/ijtm.v4i3.88.
[18] T. Panker and N. Nissim, "Leveraging malicious behavior traces from volatile memory using machine learning methods for trusted unknown malware detection in Linux cloud environments," *Knowledge-Based Systems*, vol. 226, p. 107095, Aug. 2021, doi: https://doi.org/10.1016/j.knosys.2021.107095.
[19] W. Haider, N. Moustafa, M. Keshk, A. Fernandez, K.-K. R. Choo, and A. Wahab, "FGMC-HADS: Fuzzy Gaussian mixture-based correntropy models for detecting zero-day attacks from linux systems," *Computers & Security*, vol. 96, p. 101906, Sep. 2020, doi: https://doi.org/10.1016/j.cose.2020.101906.

[20] M. Geden and K. Rasmussen, “RegGuard: Leveraging CPU registers for mitigation of control- and data-oriented attacks,” *Computers & Security*, vol. 129, p. 103213, Jun. 2023, doi: https://doi.org/10.1016/j.cose.2023.103213.
[21] M. G. Jordan, B. N. Lignati, G. Korol, M. B. Rutzig, and A. C. S. Beck, “MVSym: Efficient symbiotic exploitation of HLS-kernel multi-versioning for collaborative CPU-FPGA cloud systems,” *Integration*, vol. 93, p. 102052, Nov. 2023, doi: https://doi.org/10.1016/j.vlsi.2023.102052.

https://doi.org/10.63876/ijtm.v5i1.13