INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-29 18:38 UTC

CrackArmor: Nine AppArmor Flaws Break Container Isolation and Root Barriers Across 12.6 Million Enterprise Linux Hosts

CVE VULNERABILITY | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CVE-2026-0012
Type	CVE Vulnerability
CVE ID	CVE-2026-0012
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Linux kernel 4.11+ (all distributions with AppArmor enabled); Ubuntu, Debian, SUSE; Kubernetes on AppArmor-enabled hosts; Sudo; Postfix, estimated 12.6 million enterprise Linux instances
Published	2026-03-14

Executive Summary

Qualys researchers disclosed nine privilege escalation vulnerabilities in Linux AppArmor, collectively named CrackArmor, affecting Linux kernel 4.11 and later across an estimated 12.6 million enterprise Linux instances including Ubuntu, Debian, and SUSE (per Qualys researcher disclosure). An unprivileged local user can exploit these flaws to gain root access, escape container isolation, and bypass kernel address randomization, undermining a foundational security control across containerized workloads and multi-tenant environments. Vendor patches are available as of March 2026; no CVE identifiers have been assigned, which blocks automated scanner detection and requires manual triage and patching.

Technical Analysis

CrackArmor comprises nine confused deputy vulnerabilities (CWE-269, CWE-610, CWE-264, CWE-284) in the Linux kernel AppArmor mandatory access control module, present since kernel 4.11 (2017). Attack vector is local; an unprivileged user manipulates AppArmor security profiles via pseudo-file interfaces exposed by the kernel. Exploitation paths include: local privilege escalation to root (T1068, T1548.001), container escape on Kubernetes and AppArmor-enabled hosts (T1611), bypass of Ubuntu user namespace restrictions, KASLR offset leakage enabling follow-on memory corruption attacks (T1212), and hijack execution flow via profile manipulation (T1574, T1203). CVSS base score is assessed at 9.5 (Critical) by Qualys researchers; official CVSS rating from NVD/vendor pending CVE assignment. No CVE identifiers assigned at time of disclosure,

preventing standard NVD/scanner-based detection. Affected systems: Linux kernel 4.11+ with AppArmor enabled; Ubuntu, Debian, SUSE distributions; Kubernetes deployments on AppArmor-enabled nodes; Sudo and Postfix where AppArmor profiles are active. Ubuntu has published patch guidance in the sources section. No public CVE IDs; track via CrackArmor advisory identifier from Qualys and Ubuntu USN tracking. EPSS and CISA KEV data not yet available at time of this report.

Action Checklist

1. Step 1, Patch immediately: Apply AppArmor kernel patches per Ubuntu advisory and equivalent vendor advisories for Debian and SUSE. Prioritize Kubernetes nodes, multi-tenant hosts, and any system where unprivileged users have local shell access.
2. Step 2, Audit AppArmor enforcement mode: Run 'sudo aa-status' on all Linux hosts to confirm profiles are in enforce mode, not complain mode. Hosts in complain mode receive no active protection and are fully exposed until patched.
3. Step 3, Inventory affected hosts: Enumerate all Linux systems running kernel 4.11 or later with AppArmor enabled. Cross-reference against container orchestration inventory (Kubernetes nodes, Docker hosts) and multi-tenant environments. No CVE IDs exist yet; do not rely on vulnerability scanners alone.
4. Step 4, Restrict local access on unpatched hosts: Where patching cannot occur immediately, restrict local interactive login to privileged accounts only on affected hosts. Review and tighten user namespace policies (sysctl kernel.unprivileged_userns_clone) as a compensating control on Ubuntu systems.
5. Step 5, Notify stakeholders and update detection rules: Inform infrastructure owners and container platform teams of exposure scope. Add CrackArmor-specific monitoring logic (see detection guidance) to SIEM and EDR rules. Reassess AppArmor profile coverage as a standing GRC control after patch deployment and track for CVE assignment to enable future scanner-based validation.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO/incident commander if any unpatched multi-tenant host or Kubernetes production cluster node cannot be patched within 72 hours; escalate to external IR firm if compromise indicators (privilege escalation logs, container escape events, kernel address space modifications) are detected in forensic analysis post-discovery.
Recovery Notes	Post-containment, validate all patched systems with `aa-status --json` to confirm profiles reload correctly at boot and function in enforce mode. Audit all user accounts created or modified during the exposure window (check `/var/log/auth.log` for usermod, useradd, and sudoers changes) and reset passwords for privileged accounts. Perform threat hunt across affected infrastructure for lateral movement artifacts: search logs for unexpected sudo/su attempts, namespace creation, container process escapes, and kernel module loads (check dmesg and `/var/log/audit/audit.log` for apparmor=, cap_*, and execve patterns indicating exploitation attempts).

Forensic Artifacts	/var/log/auth.log (sudo, ssh, user login attempts; search for 'priv' and 'apparmor DENIED') | /var/log/syslog or /var/log/messages (kernel and AppArmor enforcement events; parse for 'apparmor=' and 'audit' lines) | /var/log/audit/audit.log or auditctl output (privilege escalation, namespace creation, and capability violations; requires ausearch queries) | Memory dumps or /proc/*/maps (evidence of kernel address space layout and potential KASLR bypass attempts; search for suspicious module addresses) | Container runtime logs (/var/log/docker.log, /var/log/cri-containerd.log, or kubelet logs /var/log/pods/*/kubelet.log; search for escape/exec anomalies and SELinux/AppArmor denial patterns)

Per-Action IR Details

Step 1, Patch immediately: Apply AppArmor kernel patches per Ubuntu advisory and equivalent vendor advisories for Debian and SUSE. Prioritize Kubernetes nodes, multi-tenant hosts, and any system where unprivileged users have local shell access.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.2.5

Controls: NIST SI-2 (Flaw Remediation), CIS 3.4 (Address Unauthorized Software)

Compensating: For environments without centralized patch management: (1) Use `apt list --upgradable | grep linux` (Ubuntu/Debian) or `zypper list-updates | grep kernel` (SUSE) to identify kernel versions; (2) manually download kernel images from official vendor repositories; (3) test in non-production VM first using `uname -r` to verify kernel version post-reboot; (4) schedule maintenance windows and document each patched host in a spreadsheet with hostname, previous kernel version, new version, and patch date.

Evidence: Before patching: capture `uname -a`, `cat /etc/os-release`, `sudo aa-status --json > apparmor_baseline.json`, and `cat /proc/cmdline` on all target hosts. Preserve kernel version in syslog (check `/var/log/syslog` or `/var/log/messages` for boot records). Post-patch, compare `uname -r` output and re-capture `aa-status` to confirm profile loading. For Kubernetes: `kubectl get nodes -o wide` and `kubectl describe node ` to document kernel versions across cluster.

Step 2, Audit AppArmor enforcement mode: Run 'sudo aa-status' on all Linux hosts to confirm profiles are in enforce mode, not complain mode. Hosts in complain mode receive no active protection and are fully exposed until patched.

NIST Phase: Preparation

Reference: NIST 800-61r3 §3.1.1

Controls: NIST SI-6 (Security Functionality Verification), CIS 4.1 (Ensure Audit Logging is Enabled)

Compensating: On hosts without root access: (1) request `aa-status` output from system owner via ticket system and document in spreadsheet; (2) parse `/var/lib/apparmor/profiles.db` if readable (shows loaded profiles); (3) check `/sys/module/apparmor/parameters/enabled` — if returns '1', AppArmor is loaded; (4) review `/var/log/syslog` or `/var/log/messages` for 'apparmor=' boot parameter to detect complain mode in dmesg; (5) cross-reference against Kubernetes API: `kubectl get nodes -o json | jq '.items[] | {name: .metadata.name, apparmor: .metadata.annotations}'`

Evidence: Capture full `aa-status --json` output (or `aa-status` plaintext) for every host; save to timestamped file per hostname. Record enforcement vs. complain mode decision (grep for 'enforce' vs. 'complain' in output). Extract profile names, loaded count, and unloaded profile list. For Kubernetes: capture node annotations and labels that indicate AppArmor runtime (`container.apparmor.security.beta.kubernetes.io/*`). Document in audit log which nodes were verified and on what date.

Step 3, Inventory affected hosts: Enumerate all Linux systems running kernel 4.11 or later with AppArmor enabled. Cross-reference against container orchestration inventory (Kubernetes nodes, Docker hosts) and multi-tenant environments. No CVE IDs exist yet; do not rely on vulnerability scanners alone.

NIST Phase: Preparation

Reference: NIST 800-61r3 §3.1.2

Controls: NIST CM-8 (Information System Component Inventory), CIS 1.1 (Utilize an Inventory of Hardware Assets)

Compensating: Manual inventory without CMDB: (1) export host lists from DNS (`dig axfr @nameserver domain.com`), DHCP logs, or network switches (MAC address tables via SSH); (2) iterate using `for host in $(cat hosts.txt); do ssh $host 'echo $host; uname -r; aa-status' >> inventory.csv; done`; (3) for Kubernetes, use `kubectl get nodes --all-namespaces -o wide | awk '{print $1, $7}' > k8s_nodes.txt`; (4) correlate Docker hosts by checking `docker ps -a` across orchestration systems; (5) identify multi-tenant systems by checking `/etc/passwd` for unprivileged user count and container runtime socket permissions (`ls -la /run/docker.sock`, `/var/run/cri.sock`).

Evidence: Create baseline inventory document: hostname, IP, kernel version (via `uname -r`), AppArmor enabled status (via `aa-status`), container runtime type and version (via `docker --version`, `cri-cli --version`), and date verified. Export to CSV for cross-reference analysis. For Kubernetes: capture `kubectl get nodes -o json > k8s_baseline.json` and `kubectl describe nodes > k8s_node_details.txt`. Store all inventory snapshots in version control or timestamped backup to detect drift. Record which systems have unprivileged user shell access via `/etc/shells` and active sudo policies (`sudo -l`).

Step 4, Restrict local access on unpatched hosts: Where patching cannot occur immediately, restrict local interactive login to privileged accounts only on affected hosts. Review and tighten user namespace policies (sysctl kernel.unprivileged_userns_clone) as a compensating control on Ubuntu systems.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.2.3

Controls: NIST AC-3 (Access Enforcement), NIST AC-2 (Account Management), CIS 5.2.1 (Ensure permissions on /etc/ssh/sshd_config are configured)

Compensating: (1) On unpatched hosts, disable unprivileged user namespace creation (root user action): `echo 'kernel.unprivileged_userns_clone = 0' | sudo tee -a /etc/sysctl.conf && sudo sysctl -p`; (2) restrict SSH login to specific group: modify `/etc/ssh/sshd_config` and add `AllowGroups admin sudo root` (adjust group names per environment), then `sudo systemctl restart sshd`; (3) audit current sessions via `who`, `w`, `ps aux | grep bash` and log off unauthorized users; (4) restrict console login via `/etc/login.defs` and `pam_listfile` in `/etc/pam.d/login` to block unprivileged accounts; (5) use sudo rules (`sudoedit /etc/sudoers`) to allow only approved escalation paths and log all sudo attempts via `sudo visudo` with `Defaults use_pty, Defaults log_input, Defaults log_output`.

Evidence: Before restriction: capture baseline SSH configuration (`scp root@host:/etc/ssh/sshd_config sshd_config.baseline`), current active user sessions (`w > sessions_baseline.txt`), sudo policies (`sudo -l -U > sudo_baseline.txt` for each user), and sysctl AppArmor/namespace settings (`sysctl -a | grep -E 'apparmor|userns' > sysctl_baseline.txt`). After restriction: verify changes took effect by attempting login as unprivileged user and documenting rejection in `/var/log/auth.log`. Capture new sysctl state and sshd config with timestamps. Retain baseline configs for recovery documentation.

Step 5, Notify stakeholders and update detection rules: Inform infrastructure owners and container platform teams of exposure scope. Add CrackArmor-specific monitoring logic (see detection guidance) to SIEM and EDR rules. Reassess AppArmor profile coverage as a standing GRC control after patch deployment and track for CVE assignment to enable future scanner-based validation.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §3.4.1

Controls: NIST IR-6 (Incident Reporting), NIST SI-4 (Information System Monitoring), CIS 6.2 (Activate audit logging)

Compensating: For organizations without SIEM: (1) configure rsyslog centralization on unpatched hosts — edit `/etc/rsyslog.d/50-default.conf` to forward auth and kernel logs to central syslog server: `*.* @@syslog-server:514`; (2) on central syslog, create alert rules via `grep` to detect CrackArmor exploitation patterns: `apparmor.*DENIED.*ns_capable|capable`, `audit.*priv_capable`, `sudo.*COMMAND.*ns_create`; (3) export syslog alerts to CSV weekly via `grep -E 'pattern' /var/log/syslog > weekly_audit.csv`; (4) set file-integrity monitoring via `aide --init && aide --check` (or `tripwire`, `samhain`) on critical files (`/etc/apparmor.d/*`, `/etc/sudoers*`, kernel binaries); (5) establish manual log review cadence (daily for patched systems, 4x daily for unpatched multi-tenant hosts).

Evidence: Document detection rule logic in plaintext (SIEM query examples or regex patterns); capture baseline alert volumes before and after rule deployment to establish signal-to-noise ratio. Record all rule changes in change

management log with approval chain. Archive pre-detection-rule baseline logs for 90 days to enable retroactive threat hunting post-patch. Document notification date, recipients, and stakeholder acknowledgment in incident tracker. Post-patch, execute full profile audit (`aa-status --json`) monthly and store timestamped snapshots to verify persistent compliance. Maintain CVE tracking spreadsheet with CrackArmor link, planned patch date, actual patch date, and scanner re-validation date once CVE IDs are assigned.

Detection Guidance

No CVE identifiers have been assigned; standard vulnerability scanner detection is not available. Use the following manual and behavioral detection approaches. (1) Kernel and AppArmor log review: Monitor /var/log/kern.log and /var/log/syslog for AppArmor policy load events from unprivileged user contexts; entries containing 'apparmor' with profile load or replace operations initiated outside of root or system processes are anomalous. (2) Pseudo-file access monitoring: Use auditd rules to watch for open/write operations on /sys/kernel/security/apparmor/ by non-root processes. Example auditd rule: '-w /sys/kernel/security/apparmor -p rwxa -k crackarmor_probe'. (3) Privilege escalation indicators: Alert on process trees where a non-root process spawns a root-privileged child without a corresponding sudo or setuid binary in the ancestry chain (T1068, T1548.001). (4) Container escape indicators: On Kubernetes nodes, monitor for processes executing outside expected container namespaces using tools such as Falco. Example Falco rule condition (adapt to your environment): 'spawned_process and not container and container.id != host'. (5) KASLR leak activity: Monitor for unexpected reads of /proc/kallsyms or /proc/modules by unprivileged users, which may indicate reconnaissance preceding a KASLR-bypass exploit chain (T1212). Restrict access: 'sysctl -w kernel.kptr_restrict=2'. (6) Patch verification: After applying fixes, confirm kernel version upgrade with 'uname -r' and verify AppArmor module version with 'apparmor_parser --version'. Cross-reference against patched version identifiers published in the Ubuntu USN advisory.

Framework Mappings

MITRE-ATTACK

- **T1611** — Escape to Host
- **T1574** — Hijack Execution Flow
- **T1068** — Exploitation for Privilege Escalation
- **T1212** — Exploitation for Credential Access
- **T1203** — Exploitation for Client Execution
- **T1548.001** — Setuid and Setgid

NIST-800-53R5

- **AC-6** — Least Privilege
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control

CIS-V8

- **5.4**
- **6.8**
- **6.1**
- **6.2**
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management
- **8.2** — Collect Audit Logs

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets
- **CC9.2** — Manages risks associated with vendors and business partners

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1611	Escape to Host	Privilege-Escalation
T1574	Hijack Execution Flow	Persistence
T1068	Exploitation for Privilege Escalation	Privilege-Escalation
T1212	Exploitation for Credential Access	Credential-Access
T1203	Exploitation for Client Execution	Execution
T1548.001	Setuid and Setgid	Privilege-Escalation

Sources

Source	URL	Tier
Security News	https://staging.techjacksolutions.com/news/security-news/nine-crack...	T3
Nine CrackArmor Flaws in Linux AppArmor Enable Root Escalation ...	https://thehackernews.com/2026/03/nine-crackarmor-flaws-in-linux-ap...	T3
AppArmor vulnerability fixes available - Ubuntu	https://ubuntu.com/blog/apparmor-vulnerability-fixes-available	T3
Ubuntu's AppArmor Hit By Several Security Issues - Phoronix	https://www.phoronix.com/news/Ubuntu-AppArmor-Security-Issues	T3
CrackArmor: Critical AppArmor Flaws Enable Local Privilege ...	https://blog.qualys.com/vulnerabilities-threat-research/2026/03/12/...	T3
CrackArmor Vulnerabilities - Open Chat - openSUSE Forums	https://forums.opensuse.org/t/crackarmor-vulnerabilities/192510	T3
The Linux u201cCrackArmoru201d vulnerabilities: What you need to know	https://www.thestack.technology/linux-apparmor-vulnerabilities-linu...	T3
Critical AppArmor Flaws Enable Local Privilege Escalation ...	https://www.reddit.com/r/netsec/comments/1rsqt48/crackarmor_critica...	T3
CrackArmor AppArmor Vulnerabilities Enable Linux Root ...	https://www.upwind.io/feed/crackarmor-apparmor-linux-privilege-esca...	T3
Unprivileged users could exploit AppArmor bugs to gain ...	https://securityaffairs.com/189487/hacking/unprivileged-users-could...	T3
CrackArmor - Vulnerability knowledge base	https://ubuntu.com/security/vulnerabilities/crackarmor	T3
CrackArmor Flaws Expose Linux Systems to Privilege Escalation	https://www.infosecurity-magazine.com/news/crackarmor-linux-privilege/	T2

Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-03-29 18:38 UTC by TJS Security Command Center