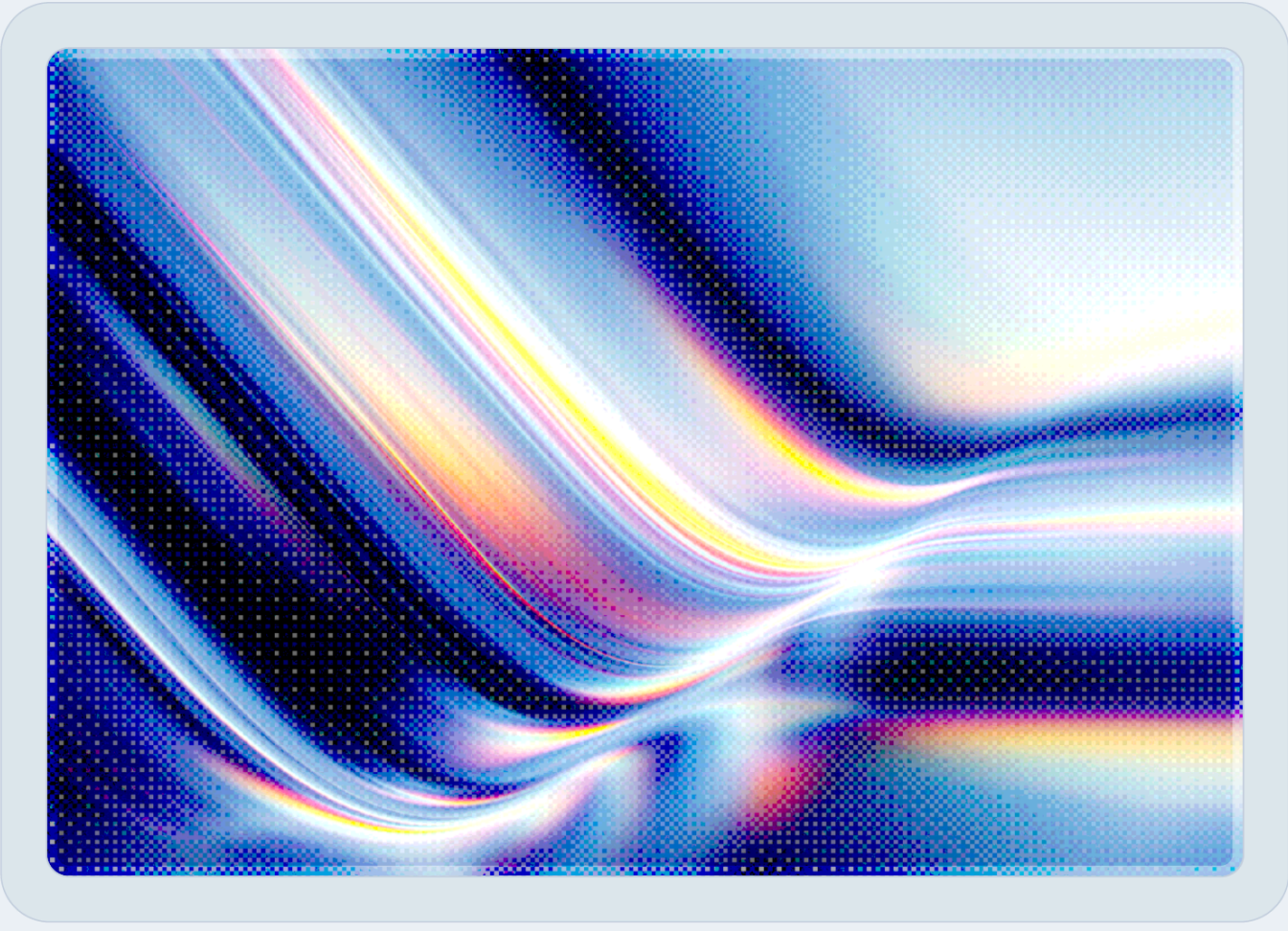
CSAI Foundation | Cloud Security Alliance

Atomic Arch: AUR Supply Chain Attack Deploys eBPF Rootkit

Developer Credential Theft at Scale via Orphaned Package Hijacking

2026-06-17

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

The Atomic Arch campaign, first identified on June 11, 2026, represents one of the most consequential supply chain attacks against the Linux developer community on record. Threat actors systematically claimed abandoned Arch User Repository (AUR) packages through a legitimate governance mechanism, then modified build scripts to deliver a Rust-based credential stealer paired with an optional eBPF rootkit to systems running as root. At peak scope the campaign touched approximately 1,500 AUR packages across two coordinated waves, earning a CVSS score of 8.7 and Sonatype tracking identifiers Sonatype-2026-003775 and Sonatype-2026-003808 [1][2][6].

The malware's credential collection profile reads like a comprehensive map of the modern developer's secrets: SSH keys, GitHub and npm tokens, HashiCorp Vault credentials, session data from Slack, Discord, and Microsoft Teams, Docker and Podman authentication, OpenAI bearer tokens, VPN profiles, and browser session cookies across 27 distinct Chromium-family browser profiles [3][5]. Exfiltration routes traffic to a Tor-hosted command-and-control server, making attribution and infrastructure takedown substantially harder than in campaigns using clearnet infrastructure [5].

Organizations with Arch Linux deployments — common in software engineering and security research teams — should treat any system that installed or updated AUR packages between June 11 and June 13, 2026 as potentially compromised. On systems where the payload ran as root, the eBPF rootkit's kernel-level concealment means normal inspection cannot be trusted; full reinstallation from trusted media is the only defensible response.

Background

The Arch User Repository is a community-maintained collection of package build scripts that complements Arch Linux's official repositories. Unlike the official repositories, the AUR is not subject to formal code review or cryptographic signing requirements. It operates on a social trust model: any registered user may contribute a PKGBUILD — a shell script that defines how to fetch, compile, and install software — and any community member may flag problems through a public comment system. AUR helpers such as `yay` and `paru` automate the process of fetching, reviewing, and building these scripts, but the review step is technically optional and rarely performed in practice [4].

Central to the Atomic Arch campaign is a feature of the AUR's governance model that allows community members to formally claim ownership of packages whose original maintainers have gone inactive. Known as orphan adoption, this mechanism exists to prevent useful packages from accumulating bit-rot when their authors depart. The AUR marks a package as orphaned when its maintainer has not logged in for a threshold period, and any logged-in user may then request stewardship through a self-service interface. The critical property exploited in this campaign is that there is no mandatory review of the incoming maintainer's identity, no cryptographic continuity requirement between old and new maintainer keys, and no automatic notification surfaced to users of the package indicating that ownership recently changed hands [7][9].

This is not an obscure edge case. The AUR's orphan adoption model has long been recognized as a trust boundary — it is documented in the Arch Wiki and accepted as a deliberate design tradeoff favoring contributor accessibility over intake security. The Atomic Arch campaign demonstrates that deliberate design tradeoff can be exploited at scale when a motivated threat actor is willing to systematically enumerate and claim hundreds of target packages in a coordinated operation [2][4].

Security Analysis

Attack Vector and PKGBUILD Manipulation

The threat actors behind Atomic Arch followed a repeatable playbook across both waves of the campaign. After claiming orphaned packages through the standard adoption workflow, they modified PKGBUILD scripts and post-install hook files to inject a call to `npm install atomic-lockfile` during the package build phase. Because AUR helpers execute these scripts in a build environment with the user's own privileges, and because the installation step commonly runs with elevated privileges to write files to system directories, the malicious npm package's preinstall lifecycle hook executed in a context with broad access to the target system [1][3].

The choice to deliver the payload via an npm package — rather than embedding it directly in the PKGBUILD — was deliberate. The AUR package itself contained no malicious code; it merely added a dependency on an external npm package that appeared to be a mundane development utility. This separation of delivery vehicle and payload allowed the compromised AUR packages to pass superficial review. Socket reported that `atomic-lockfile v1.4.2` showed only 134 weekly downloads before its removal, suggesting the package had no legitimate prior history to lend credibility [3].

When the preinstall hook executed, it dropped a stripped 64-bit Linux ELF binary named `deps` (SHA-256: `6144D433F8A0316869877B5F834C801251BBB936E5F1577C5680878C7443C98B`) compiled from a Rust codebase using async state machines [5]. A second wave, tracked as Sonatype-2026-003808, substituted the npm packages `js-digest` and `lockfile-js` after the first wave's packages were taken down, demonstrating that the actors had prepared backup delivery infrastructure in advance [1]. Sonatype also confirmed that the attackers spoofed git commit metadata to make the PKGBUILD changes appear to originate from long-standing maintainer accounts, a technique that would deceive anyone auditing the repository's commit history without cryptographically verifying signatures [3].

Credential Stealer Capabilities

The `deps` binary is purpose-built for developer environment compromise. Static analysis reveals harvesting routines targeting 27 distinct Chromium-family browser profiles — reading SQLite cookie databases, LevelDB local storage, and encrypted credential vaults [5]. The stealer queries live API endpoints at `api.github.com`, `registry.npmjs.org`, `api.openai.com`, `slack.com`, `discord.com`, and `teams.microsoft.com` using any tokens or session cookies it can locate in the filesystem [5]. Beyond browser-based credentials, the binary enumerates SSH key files at standard locations, HashiCorp Vault token files, Docker and Podman authentication configurations, VPN profile directories, and shell history files across all user accounts it can access.

The resulting data collection profile covers essentially the full surface area of a typical software developer's credential estate. This design suggests the actors are not opportunistic; they targeted a population — Linux developers running Arch — whose credential set has high downstream value for software supply chain attacks, cloud infrastructure compromise, and private repository access. OpenAI API token harvesting is a notable addition relative to earlier infostealer campaigns and indicates interest in AI development pipeline access, which may itself be preparation for subsequent attacks against AI toolchains or LLM-integrated applications.

Persistence is established differently depending on privilege level. With root access, the malware installs to a randomly generated directory under `/var/lib/` and registers a systemd service in `/etc/systemd/system/` configured with `Restart=always` and `RestartSec=30`. Without root, it falls back to `~/.config/systemd/user/` for user-level persistence [5]. In both cases the malware survives reboots, and the user-space path means it can persist without ever achieving the root access needed to load the eBPF component.

eBPF Rootkit Architecture

The eBPF component loads only when the process is running with `geteuid() == 0` and holds both `CAP_BPF` and `CAP_SYS_ADMIN` kernel capabilities. When activated, it compiles and loads an eBPF program (`scales.bpf.c`) that intercepts the `getdents64()` system call — the kernel path through which user-space tools like `ls`, `ps`, `ss`, and `netstat` enumerate processes and files [3] [5]. The rootkit pins three BPF maps at `/sys/fs/bpf/hidden_pids`, `/sys/fs/bpf/hidden_names`, and `/sys/fs/bpf/hidden_inodes`; anything in these maps is suppressed from all tools relying on standard Linux APIs, including `/proc` enumeration and socket diagnostic netlink queries [3][5].

Critically, the rootkit also detects and kills debugger attachments via `PTRACE_ATTACH` and `PTRACE_SEIZE`, which complicates live forensic analysis. The practical consequence is that on a root-compromised system, process inspection tools, file system listings, and network connection views cannot be trusted to reflect reality. Standard incident response procedures that rely on these tools may conclude a system is clean when the rootkit is actively concealing activity. Because eBPF programs are loaded into the running kernel, the rootkit does not survive a reboot in isolation — but the systemd service persistence mechanism restores it on the next boot cycle before any normal startup monitoring would observe the process in its uninitiated state.

This combination — user-space persistence via systemd, kernel-level concealment via eBPF — represents a meaningful escalation in sophistication for AUR-distributed malware. Previous supply chain attacks against the AUR have typically delivered simpler payload types with no kernel visibility component.

Exfiltration Infrastructure

Exfiltrated data transits to a Tor v3 hidden service at `olrh4mibs62l6kkuvvjyc5lrercqg5tz543r4lsw3o6mh5qb7g7sneid.onion`, reached through a local SOCKS-style proxy on `127.0.0.1`. The onion address is runtime-decoded from a 32-byte XOR key embedded at binary offset `0x1AA60` [5]. A secondary staging path uploads data to `temp.sh` via HTTP POST for files that may be too large for the primary channel. The use of Tor prevents standard network-based detection and blocks infrastructure-level takedown, since the .onion address resolves independently of DNS and does not traverse clearnet infrastructure.

Recommendations

Immediate Actions

Any organization operating Arch Linux systems — including developer workstations, security research environments, or build infrastructure — should immediately cross-reference installed AUR packages against the community-maintained affected package list published by the Arch Linux maintainer team and mirrored in the `aur-malware-check` tooling [8]. Any package installed or updated between June 11 and June 13, 2026 should be treated as potentially compromised until verified otherwise. The `aur-malware-check` repository provides scripted scanning tools that can be run across a fleet to identify compromised package versions [8].

For systems where the malicious package ran as root, or where subsequent investigation cannot confirm that root privileges were absent, the system should be treated as fully compromised. The eBPF rootkit's concealment capabilities mean forensic output from the running system cannot be trusted. The appropriate response is to preserve volatile memory if forensic analysis is planned, then reinstall from known-good media. Do not rely on malware removal tools or package uninstallation to recover a root-compromised system.

Credential rotation should proceed immediately and in parallel with system investigation. The stealer's broad targeting means every credential class it touches should be considered exposed: SSH keys should be regenerated and all authorized_keys files updated; GitHub, npm, and similar service tokens should be revoked and reissued; HashiCorp Vault tokens should be invalidated and access logs reviewed; Docker and Podman registry credentials should be rotated; and all browser session cookies for development services should be invalidated by logging out and re-authenticating. OpenAI and other AI API tokens warrant particular attention given the campaign's apparent interest in AI toolchain access.

Short-Term Mitigations

Teams using AUR packages should implement a review step for all PKGBUILD scripts before building, with particular attention to recently adopted packages. AUR helpers like `yay` support a `--editmenu` flag and pre-build review prompts; these should be enabled for all AUR installations in team environments. Any PKGBUILD that invokes `npm install`, `pip install`, `cargo install`, or other language-ecosystem package managers in its `build()` or `package()` functions warrants scrutiny, as this is the pattern the Atomic Arch campaign relied on to pull in its payload laterally.

Network monitoring should be configured to alert on Tor traffic from developer workstations and build servers. Most enterprise environments have no legitimate reason for build or developer systems to establish Tor circuits; the presence of Tor network connections on these hosts should trigger immediate investigation. Similarly, connections to `temp.sh` from production or build infrastructure are anomalous and should be flagged.

Enterprises with formal Linux fleet management should consider restricting AUR usage to an internally mirrored and reviewed subset of packages. Tools like `aurutils` support maintaining a local AUR package repository that can be locked at reviewed versions and distributed through a controlled mirror, eliminating the orphan-adoption vector for centrally managed packages [9].

Strategic Considerations

The Atomic Arch campaign exposes a structural tension that is not unique to the AUR. The same trust-delegation model that enables community-maintained package repositories to scale — any contributor can adopt an orphaned package — becomes an attack surface when threat actors are willing to operate at that scale. NPM, PyPI, RubyGems, and similar ecosystems face comparable dynamics around package abandonment, maintainer turnover, and dependency hijacking, and have been the targets of analogous campaigns in recent years.

For organizations where developer toolchain compromise has meaningful downstream consequences — access to private source code, cloud infrastructure credentials, customer data pipelines — the appropriate posture is to treat the open-source package supply chain as a shared-responsibility surface requiring the same governance attention as external software vendors. This includes formal inventorying of packages sourced from community repositories, version pinning with cryptographic integrity checks, and internal mirroring of packages that have been security-reviewed.

The OpenAI token harvesting component of this campaign also deserves strategic attention. As AI-assisted development tools proliferate across engineering teams, the API tokens that authorize access to LLM services represent a new credential class with consequential attack surface. Stolen tokens can be used to submit queries at the organization's expense, probe context from prior conversations in certain API configurations, or serve as a pivot point for attacks against model fine-tuning or retrieval-augmented generation pipelines. AI API tokens should be treated as first-class secrets in credential management programs, subject to rotation policies, least-privilege scoping, and automated revocation on anomaly detection.

CSA Resource Alignment

The Atomic Arch campaign maps to several active CSA frameworks and research areas. The AI Controls Matrix (AICM), which provides a superset of the Cloud Controls Matrix, addresses supply chain security in its software integrity and provenance controls, and applies directly to the PKGBUILD manipulation technique used here. Organizations should review their AICM control implementations for coverage of build-time dependency injection and third-party package validation, particularly in developer toolchain environments where AUR and similar community repositories may be in use.

CSA's MAESTRO framework for agentic AI threat modeling is relevant to the OpenAI token harvesting component of this campaign. MAESTRO's analysis of the developer toolchain as an attack surface for AI agent compromise aligns with the observation that stolen AI API credentials represent a distinct threat class beyond conventional credential theft. Teams deploying AI coding assistants, automated review pipelines, or LLM-integrated build systems should consider MAESTRO's guidance when assessing the blast radius of developer machine compromise.

The CSA Zero Trust guidance is directly applicable to credential rotation and access scope recommendations. The central Zero Trust principle — that compromise of any single credential should not provide broad lateral access — is violated when developers hold long-lived, broadly scoped tokens for development services. Implementing short-lived tokens with tight scope, hardware-backed authentication for high-value services, and automated anomaly detection on API usage are all Zero Trust mitigations that limit the damage from campaigns like Atomic Arch.

Finally, the STAR program's vendor assessment framework provides a mechanism for evaluating cloud and software vendors' documented supply chain controls. Organizations procuring development tooling, CI/CD platforms, or package hosting services should use STAR assessments to verify that vendors have addressed open-source dependency integrity, maintainer vetting, and anomalous maintainer transition detection in their security programs.

References

[1] Sonatype. "Atomic Arch: Attackers Hijack Trusted AUR Packages to Deliver Rootkit-Like Malware." Sonatype Blog, June 2026.

[2] BleepingComputer. "Over 400 Arch Linux packages compromised to push rootkit, infostealer." BleepingComputer, June 12, 2026.

[3] The Hacker News. "Over 400 Arch Linux AUR Packages Hijacked to Deploy Infostealer and eBPF Rootkit." The Hacker News, June 2026.

[4] Truesec. "Supply Chain Attack Compromising Arch Linux AUR Packages with Infostealer and Rootkit." Truesec Blog, June 2026.

[5] ioctl.fail. "Preliminary analysis of AUR malware." ioctl.fail Blog, June 2026.

[6] Privacy Guides. "Around 1,500 AUR Packages Compromised with 'Rootkit-Like' Malware." Privacy Guides, June 12, 2026.

[7] SecurityWeek. "Atomic Arch Supply Chain Attack Hits 1,500 AUR Packages." SecurityWeek, June 2026.

[8] lenucksi (GitHub). "aur-malware-check: Detection tools for the June 2026 atomic-lockfile AUR supply-chain attack." GitHub, June 2026.

[9] StepSecurity. "400+ AUR Packages Hijacked: What the 'Atomic Arch' Campaign Means for Supply-Chain Security." StepSecurity Blog, June 2026.