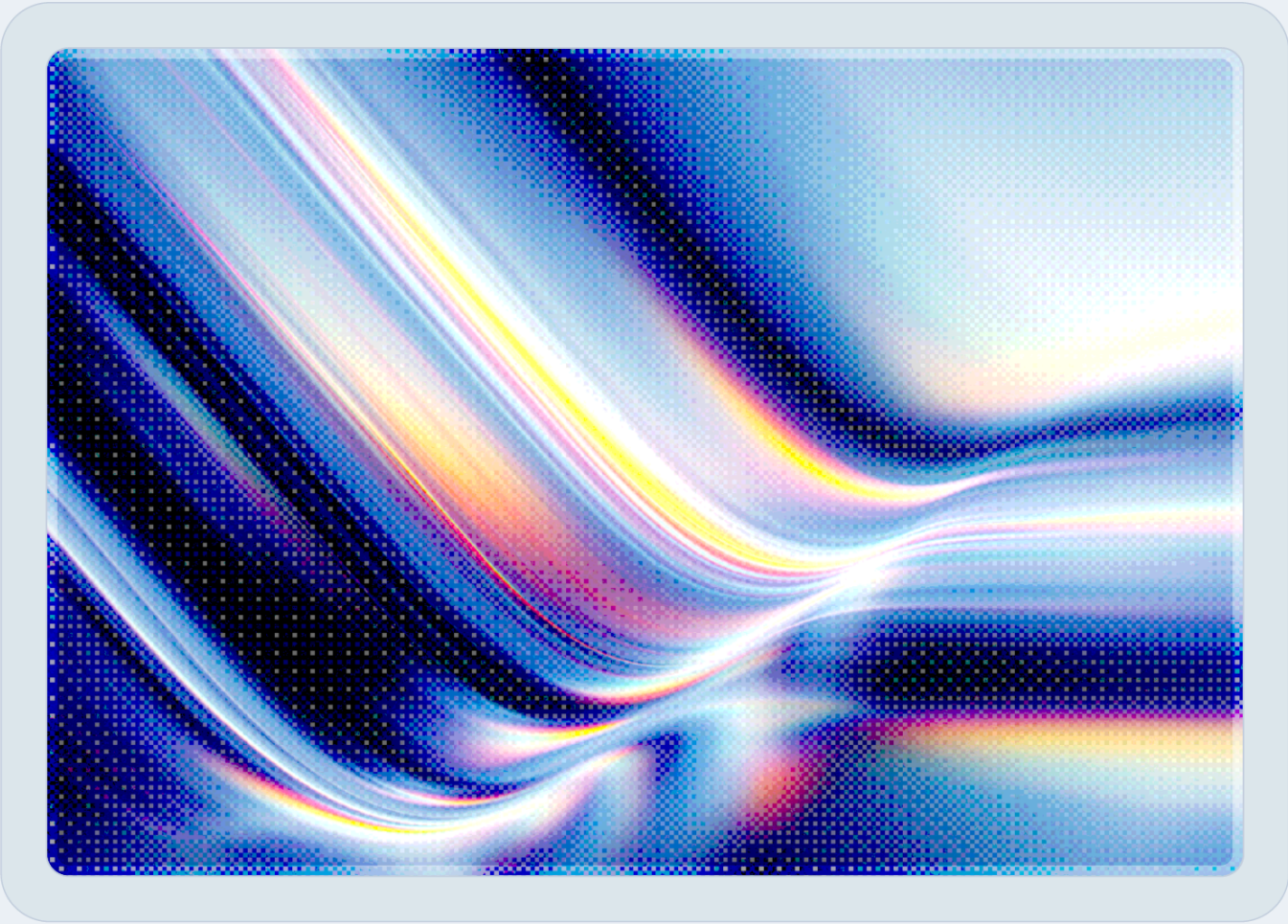
CSAI Foundation | Cloud Security Alliance

Atomic Arch: AUR Supply Chain Attack Deploys eBPF Rootkit

How Orphaned Package Hijacking Weaponized 400+ Arch Linux Packages Against Developer Credentials

2026-06-14

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Beginning June 11, 2026, attackers systematically claimed ownership of abandoned Arch User Repository (AUR) packages and modified their build scripts to distribute a Rust-written credential stealer and an optional eBPF rootkit, in a campaign Sonatype named "Atomic Arch" and tracked as Sonatype-2026-003775 (CVSS v3.1: 8.7) [1][2].
- The first wave compromised approximately 408 packages by injecting `npm install atomic-lockfile`; a second wave launched June 12 substituted `bun install js-digest`, expanding the total to more than 1,500 affected packages across the AUR ecosystem [3][4].
- The eBPF rootkit component hooks the `getdents64()` system call and maintains three BPF maps (`hidden_pids`, `hidden_names`, `hidden_inodes`) at `/sys/fs/bpf/` to conceal its presence from standard user-space tools including `ls`, `ps`, and `find` – rendering conventional malware scanning unreliable on infected hosts [5].
- Targeted data spans browser credentials, SSH keys, GitHub and npm tokens, HashiCorp Vault secrets, cloud provider credentials, Slack/Discord/Teams session tokens, Docker credentials, OpenAI bearer tokens, and cryptocurrency wallet data – the assets that underpin CI/CD pipeline access and developer workstation trust [1][3].
- Organizations running Arch Linux in development or production environments should immediately audit AUR package installation history for the June 9–12 window, rotate all credentials on any affected hosts, and treat hosts where root execution occurred as fully compromised, requiring reinstallation from trusted media [4][6].

Background

The Arch User Repository is a community-driven package repository that supplements the officially maintained Arch Linux package set. Unlike packages in the official Arch repositories, AUR packages are not vetted or signed by the Arch Linux security team. Instead, they are built from community-authored

PKGBUILD scripts — shell scripts that define how a package is downloaded, compiled, and installed. AUR helper tools such as `yay` and `paru` automate this process, fetching PKGBUILDs and executing them on the user's behalf during installation [7].

A longstanding feature of the AUR governance model is the concept of orphaned packages: when an original package maintainer becomes inactive, the package is marked as orphaned and any registered AUR user may formally claim ownership through a standard adoption request. This mechanism exists to prevent useful packages from falling into disrepair, but it creates an exploitable trust asymmetry. A package with years of installation history and community trust carries implied legitimacy that transfers to whoever controls it next, regardless of the new maintainer's intent [8].

Researchers have periodically flagged this orphan-adoption vector as a structural risk. A 2018 incident involving a malicious maintainer of an AUR PDF-viewer package demonstrated the concept in isolation. The Atomic Arch campaign represents what researchers have characterized as a qualitative escalation: rather than targeting individual packages opportunistically, the threat actor operationalized the orphan-adoption workflow at scale, systematically identifying and claiming hundreds of dormant packages with established user bases before modifying their build logic.

The Orphaned Package Attack Surface

Sonatype engineer Eyad Hasan, who identified the campaign on June 11, 2026, characterized the attacker's approach succinctly: the attackers were "not building trust from scratch — they were acquiring projects that had already earned it" [5]. The attack workflow proceeds in several steps. An attacker identifies packages that have been abandoned by their original maintainer — packages may be orphaned for entirely benign reasons, such as a maintainer changing jobs or losing interest. The attacker files an adoption request, which AUR's governance process accepts without cryptographic continuity requirements binding the new maintainer to the original project [8]. Once the adoption is approved, the attacker modifies the PKGBUILD and any `.install` hook scripts. When a user runs their AUR helper to install or update the package, the malicious build script executes with the user's full privileges.

In the Atomic Arch campaign, attackers added a single injected line that invoked the npm package manager to install the malicious `atomic-lockfile` package before or during the build process. They additionally spoofed git commit metadata to make the modification appear as if it came from previously trusted committers, reducing the likelihood that a casual reviewer of the commit history would notice the change [1][3].

Security Analysis

Malware Architecture: A Two-Stage Developer Credential Harvester

The `atomic-lockfile` npm package serves as a loader that retrieves and executes a Rust-compiled ELF binary named `deps` on the target system. This binary constitutes the primary infostealer payload, with SHA-256 hash `6144d433f8a0316869877b5f834c801251bbb936e5f1577c5680878c7443c98b` documented by Sonatype [1][5]. The choice of Rust may reflect operational preference for static linking, which produces self-contained executables with minimal runtime dependencies and can complicate detection signatures compared to interpreted payloads.

The credential stealer's target list is calibrated specifically for developer workstations and CI/CD environments. It harvests credentials from Chromium-based browsers (Chrome, Edge, and Brave), cookie databases and session tokens from Electron applications including Slack, Discord, Microsoft Teams, and Telegram, GitHub and npm authentication tokens, HashiCorp Vault access tokens, SSH private keys and `known_hosts` files, Docker and Podman credential stores, VPN configuration profiles and credentials, shell history files, and OpenAI and ChatGPT bearer tokens [1][3][4]. The explicit inclusion of CI/CD-adjacent credentials — GitHub tokens, npm credentials, Vault secrets — indicates the campaign was designed not merely for individual victim compromise but for lateral movement into software supply chains accessible from developer workstations.

Exfiltration routes operate in parallel to complicate network-layer blocking. The stealer routes data to a Tor onion command-and-control server through a local loopback proxy and supplements this with HTTP upload to `temp.sh`, a public file-sharing service. Using a legitimate file-sharing endpoint for exfiltration is a deliberate evasion technique: traffic to `temp.sh` blends with normal developer activity and may bypass proxies or firewalls that do not inspect content [1].

Persistence is established through systemd service units configured with `Restart=always`. On systems where the malware executes with root privileges, the service is installed under `/etc/systemd/system/` and the associated binary under `/var/lib/`. On systems running as a non-root user, the unit is installed in `~/.config/systemd/user/`. In both cases, the malware survives reboots and restarts automatically if killed [1][5].

The eBPF Rootkit: Kernel-Level Concealment

On hosts where the malware obtains root execution, a second component deploys an eBPF-based rootkit that operates at the Linux kernel interface rather than in user space. Extended Berkeley Packet Filter (eBPF) is a powerful kernel subsystem originally designed for network packet filtering and performance observability; it has increasingly been adopted by security tools for runtime threat detection. The Atomic Arch rootkit repurposes eBPF's kernel hook capabilities offensively [2][5][6].

The rootkit attaches to the `getdents64()` system call, which is the kernel function that directory-listing operations ultimately invoke. By intercepting this call, the rootkit can filter specific entries from directory listings before they reach user space. Three BPF maps pinned at `/sys/fs/bpf/` govern what is hidden: `hidden_pids` suppresses specific process identifiers from `ps` and similar tools, `hidden_names` removes specific filenames from directory listings, and `hidden_inodes` hides by inode number to catch cases where a file might be accessed through an alternative path [5][6]. The practical result is that a system administrator running `ls /etc/systemd/system/` on an infected host may see no suspicious service file even when one is present, and `ps aux` will not show the malware process. Community analysis also indicates the rootkit may block debugger attachment to protected processes [5][6], which would frustrate live forensic analysis.

This use of eBPF for offense is significant for defenders. Many endpoint detection and response (EDR) products rely on eBPF themselves for behavioral monitoring. An attacker-controlled eBPF program loaded at higher privilege could potentially interfere with the integrity of monitoring data, though the extent of this capability in the Atomic Arch rootkit was not fully characterized in public analysis at time of writing [2].

Scale and Second Wave

The Atomic Arch campaign proceeded in two waves. The first wave, discovered June 11, 2026, involved approximately 408 packages using the `atomic-lockfile` delivery mechanism. Within 24 hours, attackers had pivoted to a second delivery vector — the `js-digest` npm package installed via the `bun` runtime — expanding the total number of compromised packages to more than 1,500 according to community tracking at the time of initial reporting [3][4]. Community detection efforts subsequently identified approximately 1,600 affected packages as analysis progressed; both figures are cited in this document and reflect the evolving count. The speed of the second wave suggests the attackers had pre-staged alternative delivery infrastructure in anticipation that the first vector would be detected and blocked.

Confirmed compromised packages from public reporting include `alvr` (a virtual reality streaming application) and `premake-git` (a build configuration tool), as well as references to packages related to `monero-wallet-gui` that may carry an additional cryptominer payload [1][5]. The breadth of affected packages spans multiple categories — development tools, multimedia utilities, gaming applications, and system utilities — reflecting a strategy of maximum coverage rather than targeted vertical selection. Compromised packages span attack window dates from early June 2026 onward, with the modification window for some packages potentially beginning before June 11 [4].

Structural Vulnerabilities in the AUR Trust Model

The Atomic Arch campaign exploits three structural properties of the AUR ecosystem that individually are defensible design choices but in combination create a meaningful attack surface. First, AUR packages are community-maintained without cryptographic binding between a package's historical identity and its current maintainer — ownership transfers are procedural rather than cryptographic. Second, the AUR build process necessarily executes arbitrary code (PKGBUILD scripts and `.install` hooks) on the user's system, by design; the security model depends entirely on the trustworthiness of maintainers. Third, the volume of packages in AUR exceeds what any security team can audit continuously, creating persistent blind spots where malicious modifications may go undetected for days or longer [7][8].

Recommendations

Immediate Actions

Organizations should begin incident response activities now for any Linux systems running Arch Linux or Arch-derived distributions (including Manjaro, EndeavourOS, and Garuda Linux) that use AUR packages.

The community-maintained detection tool at `github.com/lenucksi/aur-malware-check` provides scripted detection across three dimensions: it cross-references installed AUR packages against the list of known-compromised packages, scans `pacman.log` for installations within the June 9–12 attack window, and checks for rootkit artifacts including the presence of BPF map files at `/sys/fs/bpf/hidden_*` and suspicious systemd service units with `Restart=always` [6]. The tool returns exit codes (0 = clean, 1 = warnings, 2 = infected) suitable for scripted fleet-wide scanning.

Any host where the detection tool returns a positive result, or where an AUR helper log confirms installation of a package updated between June 9 and June 12, 2026, should be treated as a credential compromise event requiring immediate rotation of all stored secrets. This includes GitHub personal access tokens and deploy keys, npm authentication tokens, AWS/GCP/Azure access keys and credentials, SSH private keys, HashiCorp Vault tokens, Docker Hub and container registry credentials, browser-stored passwords and session cookies, and service account tokens for CI/CD platforms [1][3][4]. Given the breadth of targeted credential types, teams should assume that any secret accessible on an affected workstation has been exfiltrated.

Hosts on which the malware executed with root privileges present a higher-severity scenario. The eBPF rootkit's ability to conceal processes and files means standard forensic tools may not produce accurate results on a running system. The recommended remediation is full host reinstallation from trusted media rather than in-place cleaning; security tooling that depends on kernel-level observation may return false negatives on a rootkitted host [2][4]. Before wiping, analysts should consider collecting memory images for forensic preservation and documenting which credentials were potentially accessible.

Short-Term Mitigations

Teams should immediately implement a policy requiring manual review of PKGBUILD scripts before any AUR package installation or update. AUR helpers such as `yay` and `paru` support review prompts that can be made mandatory by configuration; making this a default rather than an opt-in behavior meaningfully narrows the window in which a malicious PKGBUILD modification can execute undetected [7][8]. The review should specifically look for unexpected calls to `npm`, `bun`, `pip`, `curl`, `wget`, or any network fetch instruction not required by the package's documented build process.

For enterprises with Arch Linux deployments, restricting AUR usage to an explicit allowlist of packages reviewed and mirrored internally eliminates the orphan-adoption vector entirely. Teams that cannot practically eliminate AUR should prioritize packages with active maintainers, frequent upstream commits, and community discussion visible in their AUR page comments — orphaned packages and those with very recent maintainer transfers should receive heightened scrutiny.

Network-layer controls offer a complementary detection layer. Because the malware's Tor-based C2 resolves onion addresses entirely within the Tor network rather than via external DNS resolvers, DNS query monitoring will not surface onion address lookups. The reliable network-layer detection method is analysis of TCP connections to known Tor guard node IP addresses, which are publicly maintained in threat intelligence feeds (such as the Tor Project's consensus data and the dan.me.uk/torlist database). HTTP proxy inspection for uploads to `temp.sh` and similar temporary file-sharing services remains a

valid complementary control; traffic to these services can be flagged or blocked at the proxy layer. While a determined attacker can adapt exfiltration endpoints, these controls increase detection probability and reduce the operational window before discovery [1].

Strategic Considerations

The Atomic Arch campaign illustrates a broader pattern: as organizations harden their officially supported software distribution channels through code signing, reproducible builds, and mandatory provenance attestation, attackers shift toward community-governed repositories that operate on social trust rather than cryptographic guarantees. The AUR is not unique in this regard — PyPI, npm, and RubyGems have each experienced documented supply chain incidents involving orphaned or typosquatted packages [7][8]. The structural lesson is that developer toolchains represent a high-value, structurally porous supply chain layer that deserves security scrutiny proportional to the access it commands.

Organizations should inventory all package managers in use across developer workstations and build environments, map which package sources lack cryptographic maintainer binding, and apply commensurate controls. The presence of AUR, a local npm registry pointed at the public internet without lockfile verification, or a Python environment pulling from PyPI without hash pinning all represent analogous trust anchors that an adversary can target with similar techniques.

The eBPF dimension of this attack also warrants strategic attention. eBPF has become a foundational technology for cloud-native observability and security tooling — Cilium, Falco, Tetragon, and many EDR products rely on it. The dual-use nature of the technology means that organizations running eBPF-based security tools should understand what happens to their detection fidelity on a system where an attacker-controlled eBPF program is already loaded. Verifying that security tooling operates correctly in the presence of adversarial eBPF programs is an emerging validation requirement; to date, limited public guidance addresses this systematically.

CSA Resource Alignment

The Atomic Arch campaign maps directly to several domains covered by CSA frameworks, offering concrete implementation anchors for organizations responding to this class of threat.

CSA's **Cloud Controls Matrix (CCM)** addresses software supply chain risk in its Application & Interface Security (AIS) domain, specifically controls requiring that third-party software components be assessed before deployment and that integrity verification be applied to build artifacts [9]. The AUR orphan-

adoption vector would be mitigated by controls requiring cryptographic signing and provenance verification for all package sources — a gap the AUR model does not currently close. The CCM's Supply Chain Management, Transparency & Accountability (STA) domain additionally calls for continuous monitoring of third-party dependencies for compromise, which aligns with the need for automated scanning of AUR package installation history against known-bad package lists.

The **MAESTRO** framework (Multi-Agent Environment, Security, Threat, Risk, and Outcome) for agentic AI threat modeling is relevant to the extent that the campaign explicitly targeted OpenAI bearer tokens and CI/CD secrets, demonstrating that AI development toolchains are within scope for this class of credential-harvesting attack. Agentic systems that rely on secrets stored locally — including the OpenAI bearer tokens this campaign targeted — are at risk when the workstation running their development toolchain is compromised. MAESTRO's emphasis on securing the operational environment of AI agents extends to ensuring that the workstations and build pipelines that produce and operate those agents are not themselves compromised supply chain vectors [10].

CSA's **Zero Trust** guidance is directly applicable to the post-compromise response and to structural hardening. Zero Trust principles hold that no credential should be trusted indefinitely and that access should be continuously re-evaluated. For organizations where developer workstations have AUR packages installed, a Zero Trust approach to credential hygiene — short-lived tokens, just-in-time access, hardware-backed credential stores — limits the blast radius of a successful credential exfiltration even when detection fails [11]. Hardware security keys for GitHub and cloud provider authentication significantly raise the bar for re-authentication after a token expires, reducing the attacker's ability to maintain persistent access once stolen tokens age out. They do not, however, invalidate already-exfiltrated tokens that are still live — credential rotation remains the primary mitigation for active token compromise.

The **CSA STAR** (Security Trust Assurance and Risk) program provides a registry framework through which cloud providers and software vendors can communicate their security posture. Organizations procuring developer tooling or cloud services should request STAR-level documentation that addresses software supply chain controls, including whether upstream package sources are audited, whether build artifacts are reproducible and signed, and how orphaned or unmaintained dependencies are handled in vendor products.

References

[1] Swati Khandelwal. "Over 400 Arch Linux AUR Packages Compromised in Supply Chain Attack." The Hacker News, June 2026.

[2] Guru Baran. "400+ Arch Linux AUR Packages Compromised in a Supply Chain Attack Deploying Infostealers." CyberSecurityNews.com, June 2026.

[3] Bill Toulas. "Over 400 Arch Linux packages compromised to push rootkit, infostealer." BleepingComputer, June 2026.

[4] Eastern Herald. "Atomic Arch Supply Chain Campaign Hijacks 400+ Linux Packages to Deploy Rootkit and Credential Stealer." Eastern Herald, June 13, 2026.

[5] Latest Hacking News. "AUR Supply Chain Attack: 400+ Arch Packages Backdoored." LatestHackingNews.com, June 13, 2026.

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

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

[8] Threat-Modeling.com. "Arch Linux AUR Supply Chain Compromise: 400+ Packages Distributing Rootkit and Infostealer." Threat-Modeling.com, June 2026.

[9] Cloud Security Alliance. "Cloud Controls Matrix (CCM) v4." Cloud Security Alliance, 2021.

[10] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance, February 2025.

[11] Cloud Security Alliance. "Zero Trust Advancement Center." Cloud Security Alliance. Accessed June 2026.