Rust Supply-Chain Attack: arrayref, internment, and append-only-vec Poisoned by the proc-macro1 Build-Time Dropper - StepSecurity

Archived: 2026-11-04 02:03:38 UTC

Update, 2026-08-20: This incident is broader than a single crate. Two more crates owned by the same maintainer, internment 0.8.7 and append-only-vec 0.1.9, were published with the same build-time dropper dependency, and a second dropper crate (proc-macro-en) plus four more attacker-owned crates (aovine, arone, aronenao, tinymember) have been deleted from crates.io. All malicious versions are gone, the maliciously yanked releases have been restored, and the publishing accounts are locked. The sections below have been updated; the original timeline and dropper analysis are unchanged.

A compromised maintainer account and a same-day impersonator of one of Rust's best-known authors turned a routine `cargo update` *into silent remote code execution, hiding inside a 245-million-download crate at the root of the GUI and crypto ecosystems. Three crates from the same owner were poisoned in 23 minutes, and the malicious releases are already gone from crates.io, but the exposure window leaves an open question: who built during it? Here is the verified timeline, the indicators of compromise, and what your team should check right now. Our investigation is ongoing.*

`arrayref` is one of the Rust ecosystem's quiet load-bearing crates: a tiny array-conversion utility with roughly **245 million all-time downloads**, sitting underneath `tiny-skia`, `winit`, `blake3`, and large parts of the Solana and Ethereum tooling stacks. On August 20, 2026, a malicious version, `arrayref@0.3.10`, was published to crates.io from the account of its longtime owner. It injects a first-ever dependency, `proc-macro1@1.0.107`, a typosquat of `proc-macro2` published four minutes earlier by an account impersonating David Tolnay. That dependency's `build.rs` is a cross-platform dropper: it downloads a second-stage binary over TLS with certificate validation disabled, writes it to a temp directory, and executes it with a command-and-control address as its argument. Because Rust build scripts run at **compile time**, simply building any project whose lockfile resolved `arrayref 0.3.10` was enough to detonate the payload. The crate's code never needs to be called.

The compromise was not limited to `arrayref`**.** Within 23 minutes the same account published poisoned releases of two more of its crates, and the Rust Security Response Team removed a second dropper crate and four further attacker-owned crates in the same sweep. The full set of affected packages:

- **Compromised legitimate crates** (same maintainer account, clean source, malicious build-time dependency added): `arrayref 0.3.10`, `internment 0.8.7`, `append-only-vec 0.1.9`. All three are deleted from crates.io; the last safe versions are `0.3.9`, `0.8.6` and `0.1.8`.
- **Dropper crates** (attacker-published, carry the malicious `build.rs`): `proc-macro1` and `proc-macro-en`, all versions. RustSec's advisory records that `proc-macro-en` shipped the same build script as `proc-macro1`, so taking down one typosquat did not disarm the campaign.

- **Additional attacker-owned crates deleted in the same action:** `aovine`, `arone` and `aronenao`. `arone` (7 versions) and `aronenao` (11 versions) were last published on **2026-08-18**, two days before the `arrayref` push, and also executed malicious code through a build script.
- **Deleted by association:** `tinymember` (2 versions, 2026-08-18, 27 downloads). It contained no malicious code but was owned by the same account as `arone` and `aronenao`. No crates depended on it.

The maintainer of the three legitimate crates is **not** believed to be acting maliciously; the Rust Security Response Team assesses their machine or credentials as compromised and has locked the account as a precaution.

> If `cargo build`, `cargo update`, or CI resolved `arrayref 0.3.10`, `internment 0.8.7`, `append-only-vec 0.1.9`, or any version of `proc-macro1` or `proc-macro-en` between **07:11 and 09:25 UTC on 2026-08-20**, assume that machine is compromised.

There is not a single line of malicious code inside `arrayref` itself, and that is exactly what makes this attack dangerous. The poisoned release changes nothing but its dependency manifest. The malicious crate, meanwhile, ships the *genuine* `proc-macro2` source, so every build succeeds and every test passes. The only weapon is a build script.

This was not opportunistic. It was precision. The typosquat persona was created the same morning, staged with a clean decoy release five hours before weaponization. The malicious release went live four minutes after its dependency. And in the same minute, every clean `0.3.x` release of `arrayref` was yanked in a scripted burst, weaponizing Cargo's own "yanked version" warning as the delivery mechanism and herding developers toward the one remaining, poisoned release. The whole operation, from persona creation to registry takedown, fits inside a single working morning.

The attack at a glance:

- **3 crates**: legitimate packages poisoned from one compromised account (arrayref, internment, append-only-vec)
- **6 crates**: attacker-owned packages deleted from crates.io (proc-macro1, proc-macro-en, aovine, arone, aronenao, tinymember)
- **86 / 90 / 107 minutes**: per-crate exposure windows; **07:11–09:25 UTC** overall
- **264M**: combined all-time downloads of the three compromised crates
- **406**: dependent crate versions in the arrayref blast radius
- **5 hours**: decoy staged before weaponization

The incident was first reported to the RustSec advisory database (advisory-db#3161) and the Rust security team at 07:54 UTC, 39 minutes after the malicious publish. We verified every checkable element of that report against crates.io audit logs and the crates.io-index git history, and extended it with registry-side forensics and a runtime reproduction. This post is the full technical write-up.

Runtime Analysis with Harden-Runner

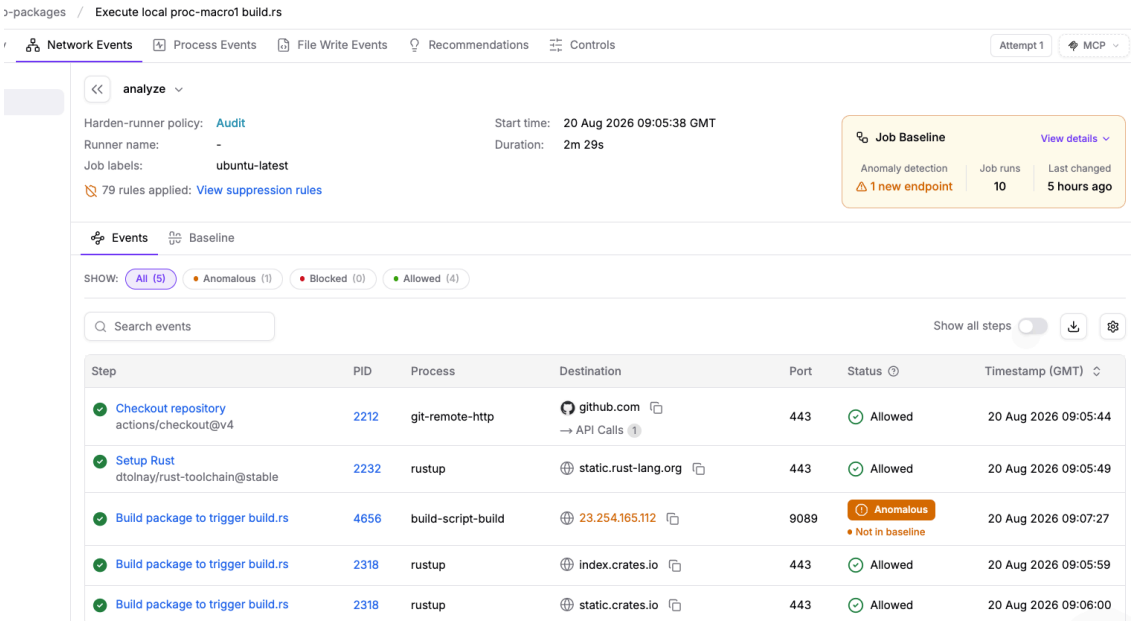
To confirm what actually happens when the poisoned release is built, we reproduced the attack in a GitHub Actions workflow (`arrayref-0310.yml` in `actions-security-demo/comp-packages`) monitored by StepSecurity

Harden-Runner. The workflow checks out a project carrying `arrayref 0.3.10`, sets up Rust, and runs a build step named, fittingly, "Build package to trigger build.rs".

Observed runtime behavior

In the first run, Harden-Runner watched the job in audit mode. The network events tell the whole story. Every connection a Rust build should make went out as expected and was marked Allowed: `github.com:443`, `index.crates.io:443`, `static.crates.io:443`, `docs.rs:443`, `static.rust-lang.org:443`. Then, at **09:07:27 UTC**, in the middle of the build step, the job opened a connection that no Rust build has any reason to make:

- **Step:** Build package to trigger build.rs (conclusion: success)
- **Event:** outbound connection
- **Destination:** `23.254.165.112:9089`
- **PID:** 4656
- **Status:** Anomalous, first-seen endpoint, never observed in any prior run
- **Timestamp:** 2026-08-20T09:07:27Z



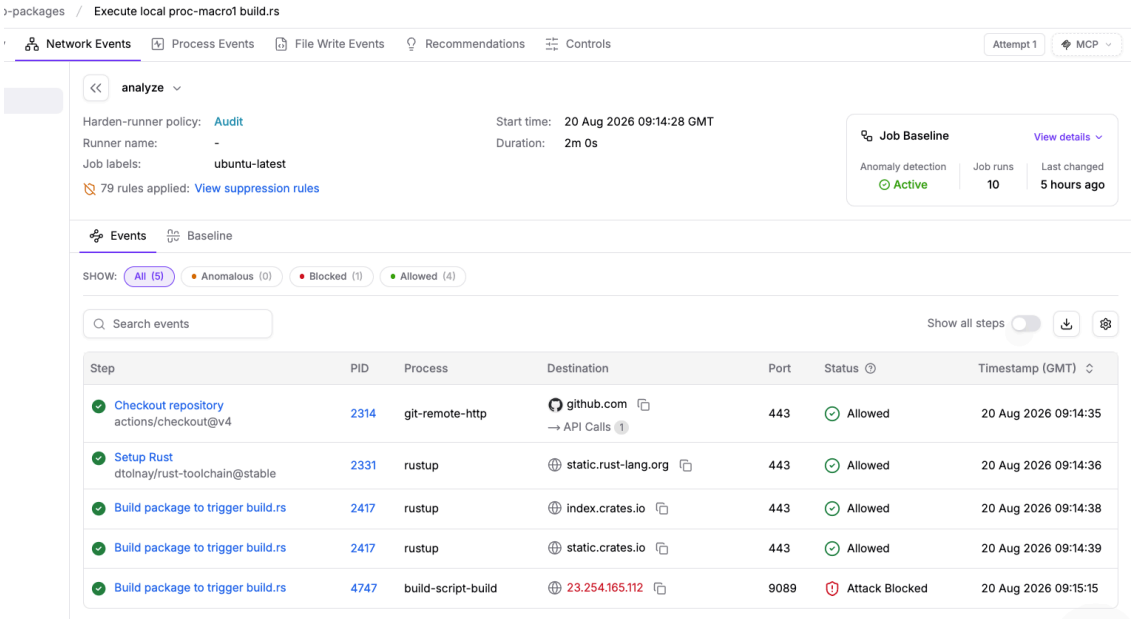
Harden-Runner network events for the audit run. Every legitimate Rust build destination is Allowed, and the build step's connection to 23.254.165.112:9089 is flagged as Anomalous, not in the baseline.

That is the dropper's payload fetch, captured live: the `proc-macro1` build script reaching out to the payload host over TLS with certificate validation disabled. Harden-Runner flagged it as **Anomalous** because the endpoint had never appeared in any previous run of the workflow. Two details make this run a perfect illustration of the attack's stealth. The step concluded **success**, and the overall job went green: from the Actions UI alone, nothing looks wrong. And the process tree shows the fetched binary continuing to run after the build script exits, matching the dropper's use of a detached spawn and `std::mem::forget(child)` to escape Cargo's job object.

Automatic blocking with the global block list

Once the incident was triaged, the payload host was added to StepSecurity's global block list. The second run repeats the exact same workflow. This time the dropper never gets its binary:

- **Step:** Build package to trigger build.rs
- **Event:** outbound connection
- **Destination:** `23.254.165.112:9089`
- **PID:** 4747
- **Status:** AttackBlocked
- **Policy:** GLOBAL_BLOCKLIST
- **Reason:** COMPROMISED_ARRAYREF_C2_IP
- **Timestamp:** 2026-08-20T09:15:15Z



The same workflow after the payload host was added to the global block list. The connection to 23.254.165.112:9089 is denied and labeled Attack Blocked, with no policy changes to the workflow.

The connection to `23.254.165.112:9089` was denied at 09:15:15 UTC, about 100 seconds into the job, with no changes to the workflow, no custom policy, and no manual rule. The global block list entry (`COMPROMISED_ARRAYREF_C2_IP`) matched automatically. With the fetch dead, the build script's silent-failure design simply swallowed the error and the build continued, but the stage-2 payload never reached the runner and the C2 channel was never established. This is the fail-closed property that matters for this attack class: even a release that is malicious for only 86 minutes gets caught by destination-based egress control, because the malware still has to call somewhere.

Both run views are public, so you can inspect every network event, file event, and process event yourself:

- Run 32352102796: runtime detection, connection flagged Anomalous
- Run 32352856981: same workflow, connection AttackBlocked by the global block list

Attack Timeline

The attack was pre-staged across roughly six hours on the day itself, with attacker-owned dropper crates seeded two days earlier and the typosquat decoyed before the malicious publish. All timestamps are UTC, 2026-08-20 unless stated:

- **2026-08-18**: `arone` (7 versions) and `aronenao` (11 versions) reach their final publishes. Both carry malicious build scripts, and both are owned by the same account as `tinymember`. The campaign's infrastructure predates the `arrayref` push by two days.
- **01:17:36**: GitHub account `dtolney` created, impersonating David Tolnay (`dtolnay`), author of the real `proc-macro2`. Zero repositories ever created.
- **01:25:58**: crates.io account `dtolney` (user id 438608) created, 8 minutes later.
- **01:55:34**: `proc-macro1@1.0.106` published: a **clean decoy**, a full copy of the legitimate `proc-macro2` source with no build script. Its sole purpose is to establish publishing history so the crate does not appear as a zero-history package during later inspection.
- **07:11:15**: `proc-macro1@1.0.107` published: malicious payload added. A `build.rs` dropper and three build dependencies (`base64`, `rustls`, `ureq`) are introduced.
- **07:15:00**: `arrayref@0.3.10` published by the compromised `droundy` owner account, injecting `proc-macro1 ^1.0.107` as a runtime dependency, the first real dependency in the crate's decade-long history.
- **07:15:24–07:15:40**: `arrayref` 0.3.9, 0.3.8, 0.3.7, 0.3.6, 0.3.5 yanked by the same account in a scripted burst (~4 s apart): the upgrade lure.
- **07:34:07**: `internment@0.8.7` published from the same compromised account, with the same build-time dropper dependency injected.
- **07:37:49**: `append-only-vec@0.1.9` published from the same account, completing three poisoned releases in 23 minutes.
- **07:54:11**: Reported to RustSec (advisory-db#3161) by researcher `jhobern`; security@rust-lang.org notified in parallel. At this point only the `arrayref` compromise is known.
- **08:03:09**: crates.io deletes `proc-macro1` entirely (index commit `c34a466f`). The crate had been live ~6 hours; the malicious 1.0.107 for ~52 minutes.
- **08:41:40**: crates.io removes `arrayref@0.3.10` from the index (commit `ce721745`). It had been live **86 minutes**.
- **09:04:11**: `internment@0.8.7` deleted, after **90 minutes** online.
- **~09:05**: The `droundy` GitHub account and the canonical `droundy/arrayref` repository return 404. Whether this was the attacker, GitHub Trust & Safety, or the owner is unknown at time of writing.
- **09:14**: `append-only-vec 0.1.9` is publicly flagged as malicious in the RustSec thread, after a reporter notices the crate was updated at the same time as `arrayref`.
- **09:25:24**: `append-only-vec@0.1.9` deleted, after **107 minutes** online, the longest exposure of the three.
- **Same day**: `proc-macro-en`, `aovine`, `arone`, `aronenao` and `tinymember` are deleted, the attacker accounts are locked, the owner's account is locked as a precaution, and the maliciously yanked `arrayref`, `internment` and `append-only-vec` releases are **un-yanked**, restoring the clean versions.

How the Attack Works

Step 1: The Typosquat Persona and the Decoy Release

Before touching `arrayref`, the attacker built a disposable identity. The crates.io account `dtolney`, one transposed letter away from `dtolnay`, the prolific author of `serde`, `syn`, and `proc-macro2`, was created at 01:25 UTC, eight minutes after its matching GitHub account. Its first and only act as a "maintainer" was to publish `proc-macro1@1.0.106`: a bit-for-bit clean copy of `proc-macro2`, versioned one patch ahead of the real crate's latest, with forged metadata: the authors field claims "David Tolnay" but points at `rchaitm@gmail.com` (not Tolnay's address), and the repository URL `https://github.com/dtolnay/proc-macro1` returns 404.

The decoy did nothing malicious. Its job was reputational: any scanner or reviewer checking the crate five hours later would find a published, building, innocent-looking crate with history, not a zero-day-old package.

Step 2: Weaponizing proc-macro1 (1.0.106 to 1.0.107)

At 07:11 UTC, version 1.0.107 landed. The library source stayed genuine `proc-macro2`; the entire weapon is a new `build.rs` plus the build dependencies it needs. Because the malicious `.crate` files were deleted from the registry (and purged from docs.rs and public mirrors) before they could be preserved, the cleanest forensic record of the change is the crates.io-index git history, the registry's own append-only metadata log.

The decoy `1.0.106` (pubtime 2026-08-20T01:55:34Z) declares a single normal dependency, `unicode-ident ^1.0`, plus dev-only dependencies (`quote`, `flate2`, `rayon`, `rustversion`, `tar`). Checksum: `b5c1b5b0763a8809a644a8f92224653f0aca623a98eecc714d27f74b80fbe436`.

The malicious `1.0.107` (pubtime 2026-08-20T07:11:15Z) adds exactly three **build-kind** dependencies: `base64 ^0.22`, `rustls ^0.23` (features: ring, std, tls12), and `ureq ^2` (features: tls). Dev dependencies are unchanged. Checksum: `61198155da51b838772eecf5bfaac6cbc4dcc388dccc56658fc28a8e831b34d4`.

A genuine proc-macro utility has no build-time dependencies at all. The 1.0.107 manifest adds exactly three, and they are a complete downloader toolkit: `base64` to reassemble obfuscated URLs, `rustls` for TLS, and `ureq` for HTTP. The registry metadata alone proves the build script's capabilities before a single line of it is read, and the registry checksums match the SHA-256 IOCs published by the original reporter exactly.

Step 3: Compromising Three Crates, the One-Line Injection Repeated

`arrayref` is owned on crates.io by a single account: `droundy` (David Roundy, a Rust developer since 2009), which also owns `internment` and `append-only-vec`. At 07:15:00 UTC that account published `arrayref@0.3.10`. The index diff against the previous release is surgical; this is the complete change to the dependency manifest of a crate that had not added a dependency in ten years:

The clean `0.3.9` (2024-09-14) lists only `quickcheck ^1.0` as a dev dependency (checksum `76a2e8124351fda1ef8aaaa3bbd7ebbcb486bbcd4225aca0aa0d84bb2db8fecb`). The malicious `0.3.10` (pubtime 2026-08-20T07:15:00Z) adds a single normal dependency, `proc-macro1 ^1.0.107`, the entire weapon (checksum `25ad700976873c76af785cb99b33c48db7df8b81f21d1e9e06b3676b9a9373ae`).

Nineteen minutes later the account repeated the move on `internment@0.8.7` (07:34:07 UTC), and three minutes after that on `append-only-vec@0.1.9` (07:37:49 UTC). Each poisoned release is the same shape: untouched library source, one added build-time dropper dependency. The compromise is of the **maintainer's account**, not of

any one crate, which is why remediation has to cover all three, and why the crates.io team locked the account rather than only removing the `arrayref` release.

There is a second-order effect worth noting: `internment` itself depends on `append-only-vec`, so a single `internment` resolve during the window could pull two poisoned crates into one dependency tree.

Phantom dependency. `arrayref` is a tiny no-dependency utility that provides macros for array references; it has no use for a proc-macro library, and `proc-macro1` is imported nowhere. A dependency that appears in the manifest of a zero-dependency crate, pointing at a same-day typosquat, is a high-confidence indicator of a compromised release all by itself.

Provenance red flags. Every prior `arrayref` release corresponds to development on the owner's GitHub repository. No commit, tag, or release for 0.3.10 exists, and at the time of writing the `droundy` GitHub account itself returns 404. The publish came from the owner's crates.io credentials (a long-lived API token or stolen session), not from the project's normal workflow. The Rust Security Response Team's assessment is that the owner's computer or credentials were compromised, not that the owner acted maliciously.

Step 4: The Yank-and-Upgrade Lure

This is the attack's most original move. Twenty-four seconds after publishing 0.3.10, the same account began yanking every other modern release: 0.3.9 at 07:15:24, then 0.3.8, 0.3.7, 0.3.6, and 0.3.5, done by 07:15:40. The crates.io audit log shows the cadence of a script, not a human:

- 2026-08-20T07:15:00.82Z: **publish** arrayref 0.3.10 by droundy
- 2026-08-20T07:15:24.21Z: **yank** arrayref 0.3.9 by droundy (+24s)
- 2026-08-20T07:15:26.80Z: **yank** arrayref 0.3.8 by droundy (+2.6s)
- 2026-08-20T07:15:30.70Z: **yank** arrayref 0.3.7 by droundy (+3.9s)
- 2026-08-20T07:15:36.29Z: **yank** arrayref 0.3.6 by droundy (+5.6s)
- 2026-08-20T07:15:40.18Z: **yank** arrayref 0.3.5 by droundy (+3.9s)

Yanking does not break existing builds; Cargo keeps using a yanked version already in a lockfile. What it does is make Cargo print, on every subsequent build: `warning: package \`arrayref v0.3.9\` in Cargo.lock is yanked in registry \`crates-io\`, consider updating to a version that is not yanked`.

The intended victim psychology is obvious: developers and CI maintainers see a scary new warning on a dependency with 152 million downloads, and the "responsible" fix, `cargo update -p arrayref`, resolves to the single non-yanked modern release: **0.3.10**. The original reporter states this is exactly how they were hit. The attacker turned the registry's own safety feature into the delivery channel. The same yank pattern was applied to the other two crates, and crates.io has since **reversed all of these yanks**, so the clean releases no longer carry a warning.

Step 5: A Second Dropper Crate, and Crates Staged Two Days Early

Removing `proc-macro1` did not end the campaign. crates.io also deleted `proc-macro-en`, which RustSec's advisory records as containing **the same build script** as `proc-macro1`, a spare dropper, ready to be swapped into a manifest if the first typosquat was burned. Four more attacker-owned crates went with them: `aovine`, `arone`,

`aronenao` and `tinymember`. `arone` and `aronenao` carried malicious build scripts of their own across 7 and 11 versions respectively, with their last publishes on **2026-08-18**; `tinymember` held no malicious code but shared their owner and was removed by association. Treat every version of all six attacker-owned crates as untrusted, not just the versions that happened to be referenced by a compromised release.

The Dropper: Static Analysis of build.rs

The following is quoted from the original report and corroborated by the registry metadata above. The `.crate` files themselves were deleted before they could be archived, but all three reported SHA-256 hashes match the registry's checksums, and the build-dependency set is only explainable by this downloader.

The payload infrastructure is split into base64 fragments so no string match on the raw source reveals a URL. The source defines `SRC_URL_PARTS` (`aHR0cHM6Ly8=` , `MjMuMjU0Lg==` , `MTY1Lg==` , `MTEyOg==` , `OTA4OS8=`) and `END_URL_PARTS` (`MjMuMjU0Lg==` , `MTY1Lg==` , `MTEyOg==` , `NDQz`). Decoded: SRC = `https://23.254.165.112:9089/` (payload host), END = `23.254.165.112:443` (C2, passed to the payload as argv[1]).

The fetch is made over TLS with certificate validation explicitly disabled, an `AcceptAll` `ServerCertVerifier` whose verify methods return success unconditionally (necessary, since the endpoint is a bare IP with no valid certificate). One of four artifacts, `rust-crate_0.1.0` through `rust-crate_0.4.0` , is selected by target platform, then executed with the C2 address as its first argument:

- **Unix:** written to `/tmp/rust-setup` , made executable, spawned detached with stdio nulled.
- **Windows:** dropped as `rust-setup.ps1` in the user's temp directory and launched through a `rust-setup-launch.vbs` wrapper under `wscript.exe` with no console, no window, no prompt.
- **Both:** the child handle is abandoned with `std::mem::forget(child)` , commented in-source as escaping Cargo's job object, so the build does not wait on the malware and the compiler exits cleanly. The build succeeds. Nothing looks wrong.

Execution Sequence: One cargo build Is Enough

The critical property of this attack is where it executes. Rust build scripts compile and run on the machine performing the build, with that user's full privileges, including access to SSH keys, cloud credentials, CI secrets, and signing keys. The sequence: `cargo build` (or `cargo update` first) resolves `arrayref ^0.3` to 0.3.10, which pulls `proc-macro1 ^1.0.107` ; Cargo compiles and runs the build script; the script base64-decodes the URL fragments, fetches the platform binary from `23.254.165.112:9089` over HTTPS with the AcceptAll verifier, drops `/tmp/rust-setup` , makes it executable, spawns it detached, and abandons the handle with `mem::forget(child)` . The build script exits 0 and the build continues normally. The payload now runs independently of the build, connecting to the C2 at `23.254.165.112:443` . The infection is invisible in normal compiler output. The same sequence applies to a tree that resolved `internment 0.8.7` or `append-only-vec 0.1.9` .

What the Stage-2 Payload Does on an Infected Host

The stage-2 binaries were not recoverable from public sources, but a developer who was infected on Linux reported the post-execution behavior in the public RustSec thread. On that host the payload established persistence: it created `$HOME/.config/AzureKits` and `$HOME/.config/ServiceKit`, dropped executables named `MonoService` and `MonoXpc`, registered a systemd service to restart itself, and connected to `23.254.167.216`, a different address in the same Hostwinds range as the build-time C2. A separate researcher in the same thread reported `23.254.167.107` in use for command and control. These are third-party reports we have not independently reproduced, but they are consistent with the dropper's design and worth hunting for now.

Blast Radius

All three compromised crates are low-level utilities that sit far down other people's dependency trees, and none of them is a leaf:

- `arrayref` **0.3.10**: 245.2M all-time downloads, 53.7M in the last 90 days. 406 dependent crate versions. Last safe version: **0.3.9**.
- `internment` **0.8.7**: 14.4M all-time downloads, 3.5M in the last 90 days. Last safe version: **0.8.6**.
- `append-only-vec` **0.1.9**: 4.5M all-time downloads, 1.3M in the last 90 days. Last safe version: **0.1.8**. Also pulled in transitively by `internment`.

For `arrayref`, the original report traced the GUI chain, and the crates.io reverse-dependency graph shows the exposure extends into hashing, blockchain infrastructure, and beyond. 406 crate versions depend on `arrayref` directly, including:

- `tiny-skia` / `tiny-skia-path` → `sctk-adwaita` (26M downloads) → `winit` (49.8M downloads) → `egui` / `eframe` / `iced`: most Rust GUI apps
- `blake3` and `blake2b_simd` / `blake2s_simd`: hashing
- `revm-precompile`: Ethereum tooling
- `spl-token` / `solana-runtime`: Solana tooling

Any project anywhere in these trees whose lockfile was created or refreshed during the exposure windows would have resolved the poisoned version and executed the dropper at build time. Builds with pre-existing lockfiles pinning clean versions kept compiling the clean, yanked-but-downloadable code, since yanking alone never breaks a locked build.

Indicators of Compromise

Network:

- `23.254.165.112:9089` (payload host, TLS with any certificate accepted)
- `23.254.165.112:443` (C2, passed to payload as argv[1])
- `23.254.167.107` (additional C2, reported by a third party in advisory-db#3161)
- `23.254.167.216` (stage-2 C2 observed on an infected Linux host, reported in advisory-db#3161)
- `hwsrv-798836.hostwindsdns.com` (Hostwinds VPS)

Both `23.254.165.0/24` and `23.254.167.0/24` are Hostwinds ranges; treat the ranges, not just the individual addresses, as the indicator.

Files (build-time dropper):

- `/tmp/rust-setup` (Unix)
- `rust-setup.ps1` and `rust-setup-launch.vbs` in the Windows temp directory

Files and persistence (stage 2, third-party reports):

- `$HOME/.config/AzureKits` and `$HOME/.config/ServiceKit` (directories created on an infected Linux host)
- Executables named `MonoService` and `MonoXpc`
- A systemd service registered to restart the payload

Payload artifacts: `rust-crate_0.1.0` / `rust-crate_0.2.0` / `rust-crate_0.3.0` / `rust-crate_0.4.0`

Compromised legitimate crates (deleted): `arrayref 0.3.10` , `internment 0.8.7` , `append-only-vec 0.1.9`

Attacker-owned crates (all versions deleted): `proc-macro1` (1.0.106 decoy & 1.0.107 weaponized), `proc-macro-en` , `aovine` , `arone` , `aronenao` , `tinymember`

Not affected: `proc-macro2` . The legitimate, foundational crate by David Tolnay is not compromised in any way; only the `proc-macro1` and `proc-macro-en` typosquats are malicious.

Accounts: `dtolney` (crates.io id 438608), impersonator of dtolnay, locked; the account owning `arone` / `aronenao` / `tinymember` , locked; `droundy` , legitimate owner of the three compromised crates, locked as a precaution and assessed as compromised rather than malicious

Email: `rchaitm@gmail.com` (forged "David Tolnay" author metadata)

SHA-256 (.crate files, registry-verified via index git history):

- `25ad700976873c76af785cb99b33c48db7df8b81f21d1e9e06b3676b9a9373ae` : arrayref-0.3.10.crate
- `61198155da51b838772eecf5bfaac6cbc4dcc388dccc56658fc28a8e831b34d4` : proc-macro1-1.0.107.crate
- `b5c1b5b0763a8809a644a8f92224653f0aca623a98eecc714d27f74b80fbe436` : proc-macro1-1.0.106.crate

Checksums for `internment 0.8.7` , `append-only-vec 0.1.9` and the `proc-macro-en` release are not recoverable: those versions were hard-deleted from crates.io, the sparse index and docs.rs before they could be archived. If you hold a cached `.crate` file for any of them, the Rust security response team would like to hear from you.

Am I Affected?

Check if you are affected in minutes

StepSecurity checks your repositories, CI/CD pipelines, and developer machines for compromised Rust crates like arrayref 0.3.10 and proc-macro1.

Code repositories

The decisive check is the lockfile, not the registry. Search every `Cargo.lock` in your organization for these entries:

- `arrayref` at version **0.3.10**: compromised
- `internment` at version **0.8.7**: compromised
- `append-only-vec` at version **0.1.9**: compromised
- `proc-macro1` , `proc-macro-en` , `aovine` , `arone` , `aronenao` , `tinymember` : **any** hit, at any version, means compromised

Note that `Cargo.toml` pins do NOT protect you: `arrayref = "0.3"` matches 0.3.10. Only the resolved lockfile version matters. A single command over your checkouts:

```
grep -rEn 'proc-macro1|proc-macro-en|aovine|arone|aronenao|tinymember|arrayref.*0\.3\.10|internment.*0\.8\.7|append-only-vec.*0\.1\.9' --include=Cargo.lock --include=Cargo.toml .
```

Automated tooling will not flag this for you. Because the malicious versions were *deleted* rather than yanked, and the RustSec advisories are still pending, `cargo audit` reports clean for a project that pinned a poisoned version. Worse, a machine with a warm `~/.cargo/registry` cache or a committed `vendor/` directory keeps building the payload offline even though the version no longer exists in the registry. Deletion from crates.io does not clean local caches.

Local and CI caches

Check the cargo cache directly for the deleted artifacts; this is the check the Rust security team recommends:

```
find ~/.cargo/registry/cache -type f \( -name 'arrayref-0.3.10.crate' -o -name 'internment-0.8.7.crate' -o -name 'append-only-vec-0.1.9.crate' -o -name 'proc-macro1-*.crate' -o -name 'proc-macro-en-*.crate' -o -name 'aovine-*.crate' -o -name 'arone-*.crate' -o -name 'aronenao-*.crate' -o -name 'tinymember-*.crate' \) -print
```

Run the same check against any cached cargo registry restored by CI (`actions/cache` , self-hosted runner home directories, container images baked with a warm cache), and purge matching entries before the next build.

CI/CD pipelines

Audit build logs from **07:11–09:25 UTC on 2026-08-20** for jobs that created or updated lockfiles (`cargo update` , `cargo add` , fresh `cargo build` without a committed lockfile, cache-busted runners). Check egress logs for connections to `23.254.165.112` , `23.254.167.107` and `23.254.167.216` . Ephemeral runners destroyed after the job still exfiltrated whatever secrets the job held, so review what each affected job could access.

Developer machines

On Unix, check whether the file `/tmp/rust-setup` exists. On Windows, check the user's temp directory for `rust-setup.ps1` and `rust-setup-launch.vbs`. On Linux, also check for the stage-2 persistence artifacts reported by an infected developer: `$HOME/.config/AzureKits`, `$HOME/.config/ServiceKit`, binaries named `MonoService` or `MonoXpc`, and any unexpected systemd service that restarts them.

- **Pin the clean releases.** If your tree needs these crates, pin `arrayref = "=0.3.9"`, `internment = "=0.8.6"`, and `append-only-vec = "=0.1.8"`. crates.io has reversed the attacker's yanks, so these versions are once again the normal, warning-free choice, but verify the resolved version in your lockfile rather than trusting the absence of a warning.
- **Treat exposed machines as compromised.** The payload ran with the building user's privileges. Rotate every credential reachable from the machine: SSH keys, cloud tokens, crates.io/API tokens, signing keys, and CI secrets.
- **Purge caches and vendored copies.** Delete matching `.crate` files from `~/.cargo/registry/cache`, drop CI cache entries, and rebuild any `vendor/` directory from clean sources; a deleted version still builds from a warm cache.
- **Rebuild artifacts produced during the window** from clean sources on clean infrastructure before shipping them.
- **Block the IOCs** at egress: `23.254.165.112`, `23.254.167.107` and `23.254.167.216` on all ports (ideally the surrounding Hostwinds ranges), and alert on historical connections to them.

Defense in depth

- **Cooldown policies:** the malicious releases were live for 86, 90 and 107 minutes. A policy that only admits crate versions older than N days would have made this attack a non-event for your organization.
- **Lockfile discipline:** commit `Cargo.lock` for applications, build with `--locked` in CI, and review lockfile diffs in PRs. A new entry named `proc-macro1` or `proc-macro-en` is impossible to miss in review.
- **Dependency provenance:** tools like `cargo vet` and `cargo audit`, plus alerting on "yanked" warnings treated as release signals rather than chores, all raise the cost of this attack class. Note the gap this incident exposed: deleted malicious versions produce no `cargo audit` finding at all.
- **Publishing hygiene:** this compromise almost certainly used a long-lived crates.io token, and one stolen token was enough to poison three crates in 23 minutes. Scoped, short-lived tokens and trusted-publishing-style OIDC flows remove that theft surface.

For StepSecurity Enterprise Customers

Threat Center Alert

StepSecurity Threat Center customers have been alerted to this campaign with the complete IOC set: the payload host and C2 addresses (`23.254.165.112`, plus the reported `23.254.167.107` and `23.254.167.216`), the dropped-file artifacts for Unix and Windows, the stage-2 persistence artifacts, all three compromised crate versions, the six attacker-owned crates, and the registry-verified SHA-256 hashes. Threat Center continuously tracks this incident as analysis of the second-stage payload progresses, and maps the indicators across your

organization's repositories and pipelines so you can see at a glance which projects carried `arrayref`, `internment` or `append-only-vec` in their dependency tree during the exposure windows.

Harden-Runner

The runtime section above shows both sides of Harden-Runner's value for this incident. In audit mode, it flagged the build-time connection to `23.254.165.112:9089` as an anomalous destination never seen in prior runs of the workflow, along with the file write to `/tmp/rust-setup` and the detached child process that outlives the build step. And once the indicator was added to the global block list, the same connection was blocked automatically (`AttackBlocked`, reason `COMPROMISED_ARRAYREF_C2_IP`) with no policy changes required from the customer. Because the block is on the destination rather than the package name, the same protection applies to the `internment` and `append-only-vec` releases and to the spare `proc-macro-en` dropper, which share the same infrastructure.

Interactive demo: walk through how Harden-Runner monitors and blocks outbound traffic from CI workflows

Harden-Runner's community tier is free for public repositories and is used by over 12,000 open-source projects; insights for community-tier projects are public by design, so anyone can verify detections, including the two runs linked in this post.

References

- Rust Security Response Team: Supply chain attack on arrayref: the official advisory, with per-crate publish and deletion timestamps and the full list of deleted crates
- rustsec/advisory-db#3161: original disclosure by researcher `jhobern` (primary source for the `build.rs` quotations), and the thread where `append-only-vec`, the additional C2 addresses, and the stage-2 persistence artifacts were reported
- rustsec/advisory-db#3162 and #3163: pending RustSec advisories for `arrayref`, `internment`, `append-only-vec`, `proc-macro1`, `proc-macro-en`, `arone`, `aronenao` and `tinymember`
- Harden-Runner run 32352102796: runtime detection of the payload-host connection (Anomalous)
- Harden-Runner run 32352856981: automatic block via the global block list (AttackBlocked, COMPROMISED_ARRAYREF_C2_IP)
- crates.io index and its git mirror: publication, yank, and deletion records cited throughout
- crates.io: internment and crates.io: append-only-vec: current registry state and download counts for the other two compromised crates

Last updated: 2026-08-20. The Rust Security Response Team credits the Research Team at Nextron Systems GmbH with initially discovering and reporting the malicious crate. The stage-2 binaries (`rust-crate_0.1.0` to `_0.4.0`) were not recoverable from public sources at the time of writing; the persistence behavior described above comes from third-party reports by infected developers, which we have not independently reproduced. If you captured the payload, or have build or egress telemetry from 07:00–09:30 UTC on 2026-08-20, please share it with the Rust security response team. This post will be updated as the investigation progresses.

Source: https://www.stepsecurity.io/blog/arrayref-rust-crate-supply-chain-attack#blast-radius