

Trust but Verify: The xz-utils Backdoor (CVE-2024-3094) and the Foundations of Open-Source Software Supply-Chain Trust

Bin Cao

ORCID: 0009-0005-0304-8805

School of Information Science and Technology and Guangdong Provincial Key Laboratory of Aerospace Communication and Networking Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China

Yishuai Chen

ORCID: 0000-0002-0105-783X

School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China

Correspondence: Yishuai Chen · contact details held in the journal record

Publication history. First published in the offline edition of Cybersecurity Advances, Volume 1, Issue 1 (December 2024). This publisher-authorized digital archival edition was released on 12 October 2026. The scholarly content is reproduced from the supplied offline-edition manuscript; review-stage notices were removed and journal typography, pagination, metadata, and accessibility were normalized.

© 2024 the author(s). Published by AlphaTech Scholarly Press under the Creative Commons Attribution 4.0 International License (CC BY 4.0). No DOI is displayed unless and until registration is verified.

Background: In March 2024, a multi-year social-engineering operation culminated in the injection of a sophisticated backdoor into xz/ utils versions 5.6.0 and 5.6.1 (CVE-2024-3094, CVSS 10.0), reaching OpenSSH on rolling-release GNU/Linux distributions before a fortuitous performance anomaly led to its discovery and containment. **Objective:** This paper treats the xz-utils incident not as an anomaly to be patched but as a diagnostic event that exposes the load-bearing assumptions of trust in open-source software supply chains — and evaluates, stage by stage, whether the security instruments now in circulation would have prevented, detected, or contained it. **Methods:** We conduct a qualitative single-case study built exclusively from primary sources (the disclosure thread, the CVE record, distribution advisories, and the CISA alert), reconstruct the attack chain as a sequence of six trust transitions, and perform a counterfactual control analysis mapping six classes of supply-chain controls (maintainer-support programs, provenance attestation, reproducible builds, SBOM and binary analysis, runtime isolation, and coordinated response) against each stage. **Results:** No single instrument spans the full chain. Provenance attestation and reproducible builds would have raised the cost of the build-time activation stage dramatically; maintainer-support and criticality programs address the social-engineering stage that every technical control leaves untouched; runtime isolation (SELinux) actually prevented exploitation on some systems; and the coordinated response within hours demonstrated that recovery-side trust mechanisms are the field's most mature layer. We further show that SBOMs — the most-cited 2024 remedy — would not have detected this attack, and argue that trust in supply chains must be built on *verifiable process evidence* rather than identity or reputation. **Conclusions:** The xz-utils incident defines the research and policy agenda for supply-chain trust: fund maintainers of critical-but-invisible infrastructure, instrument trust transitions with verifiable process guarantees, and design for the adversary who behaves like an ideal contributor.

Keywords: software supply chain security; open-source software; xz-utils; CVE-2024-3094; provenance attestation; reproducible builds; SBOM; digital trust

Contribution statement. This paper makes four contributions: (1) a source-verified reconstruction of the CVE-2024-3094 attack chain as six sequential trust transitions, suitable for reuse in threat modeling; (2) a counterfactual control analysis that maps six control classes against the six attack stages and identifies precisely where the current instrument stack has no meaningful coverage; (3) a correction of the post-incident policy discourse, showing that SBOM adoption alone would not have detected or prevented this attack; and (4) a stakeholder-differentiated set of recommendations (maintainers, distributors, consumers, funders) grounded in the 2024 policy corpus (SLSA v1.0, NIST SSDF, EU Cyber Resilience Act, CISA Secure-by-Design).

1. Introduction

Open-source software is the substrate of digital trust: it composes operating systems, TLS stacks, compression libraries, and firmware that together mediate nearly every secure transaction on the Internet. Yet the trust relations that produce and sustain this

substrate are thin — maintained by individuals whose incentives are reputational rather than financial, integrated by distributions on release-velocity timescales, and consumed by organizations that rarely inspect what they compile and run. The software supply-chain security literature has catalogued this asymmetry for years [5][6][7], but the field's warnings acquired a concrete, near-catastrophic demonstration on 29 March 2024, when Andres Freund disclosed that xz/utils — a compression utility present in effectively every GNU/Linux system — had been carrying a carefully engineered remote-code-execution backdoor in versions 5.6.0 and 5.6.1 [1].

The mechanics of the incident are worth stating plainly, because their sophistication reframes the threat model. An attacker operating for roughly two and a half years under the identity "Jia Tan" (account created 2021) first contributed ordinary patches to the under-resourced project, then cultivated operational credibility — new release infrastructure, mailing-list presence, and persistent advocacy for co-maintainer status — while the long-serving sole maintainer, managing the project through a period of personal difficulty, gradually ceded responsibility [1]. In early 2024 the attacker committed a malformed test corpus whose decode failure was engineered to trigger injection of an object file at build time on glibc-based GNU/Linux systems; the resulting liblzma object hooked the symbol resolution of programs that linked it — including OpenSSH's sshd via systemd's libsystemd — enabling unauthenticated remote code execution on affected x86-64 systems [1][3][4]. The backdoor reached rolling-release distributions (Fedora Rawhide, openSUSE Tumbleweed, Debian sid/experimental, and others) but not the enterprise or stable channels; it was discovered because Freund, investigating a curious half-second CPU overhead in ssh logins while testing PostgreSQL under Valgrind, followed a defect in memory profiling to its impossible origin [1]. Within hours of disclosure, distributions reverted, the CVE was assigned with a CVSS score of 10.0, and CISA issued an alert [3][4]. The incident ended as a narrow miss — a weaponized, patient, apparently state-grade intrusion deactivated days before reaching millions of stable systems.

The field's immediate reaction was procedural (rotate keys, patch, audit), but the durable lesson is structural: **every individual hop in the attack chain was a legitimate-seeming trust transition** — a helpful patch, a release-infrastructure improvement, a co-maintainer appointment, a test-corpus fix, a build-system modernization. No signature check, no code review, and no vulnerability scanner in routine use would have flagged any single transition as malicious. The attack did not defeat the supply chain's controls; it *became* the supply chain's normal operation.

1.1 Research questions

This paper asks what the incident teaches about the *foundations* of software supply-chain trust. **RQ1:** What is the verified sequence of trust transitions by which the attacker reached OpenSSH (the attack chain)? **RQ2:** Which existing controls — technical and organizational — would have prevented, detected, or contained each stage of the chain? **RQ3:** Where does the current control stack have no meaningful coverage, and what does that imply for research and policy? **RQ4:** What differentiated responsibilities follow for maintainers, distributors, consumers, and funders of critical open-source infrastructure?

1.2 Contributions

We answer these questions with four contributions. First, a source-verified reconstruction of the attack chain as six trust transitions (Section 4, Figures 1–2) (RQ1). Second, a counterfactual control analysis across six control classes and six attack stages (Section 5, Figure 3, Table 2), identifying the coverage frontier — the stages where no deployed instrument provides meaningful protection (RQ2, RQ3). Third, an argument — against the dominant post-incident discourse — that SBOM adoption, while valuable for other purposes, would not have detected this attack, because the backdoor was hidden inside a compressed *test fixture* and activated at build time on consumers' machines (Section 5.3) (RQ3). Fourth, stakeholder-differentiated recommendations aligned with the 2024 policy corpus (Section 6, Table 3) (RQ4).

2. Background and Related Work

2.1 Open-source supply-chain attack taxonomy

The supply-chain attack literature provides the taxonomy into which CVE-2024-3094 falls. Ohm et al.'s review of open-source supply-chain attacks classifies attacks by the lifecycle stage they compromise [6]; Ladisa et al.'s systematization enumerates the attack vectors and linked defenses for open-source ecosystems, ranking vectors by feasibility [5]. Within these taxonomies, the xz-utils incident is a *source-code integrity attack executed through maintainer compromise* — the most severe class, since it operates upstream of every technical checkpoint, in the social layer where authority to modify the project is granted. Zimmermann et al.'s study of the npm ecosystem demonstrated empirically that the effective security of a package depends on the security behavior of its maintainers and that "trusted" identity is a poor proxy for trustworthiness [7]. The xz incident is the maximal expression of that finding: the attacker did not impersonate a maintainer — they *became* one.

2.2 Precedents and the criticality asymmetry

The incident joins a lineage of wake-up calls whose pattern is instructive. OpenSSL's Heartbleed (2014) revealed that infrastructure used by nearly every Internet user was maintained by a handful of barely-funded volunteers [19]; the 2018 event-stream npm incident demonstrated social-engineering takeover of an abandoned-but-widely-installed package [6][7]. The repeated lesson — formalized in the OpenSSF Mobilization Plan [14] and in criticality analyses such as the Census II study [17] — is an asymmetry: the most *consumed* open-source components are often the least *resourced*. xz-utils, present in effectively every GNU/Linux installation yet maintained by one volunteer, sits at the extreme of that distribution.

2.3 The 2021–2024 instrument stack

Between 2021 and 2024 the policy apparatus for supply-chain trust expanded substantially, and all of it predates the incident. U.S. Executive Order 14028 (2021) and the NTIA minimum elements made SBOMs a procurement expectation [10][11]. NIST's Secure Software Development Framework (SSDF v1.1) codified producer-side secure development practices [12]. The OpenSSF's SLSA framework reached v1.0 in 2023, defining graded levels of build-platform provenance guarantee [9]; in-toto provides cryptographic attestation of the steps of a software supply chain [8]; sigstore provides free, short-lived code-signing infrastructure [20]; reproducible-builds techniques enable independent verification that binaries correspond to claimed sources [13]. On the consumer side, OpenSSF Scorecard automates project security posture assessment [18]. On the regulatory side, the EU Cyber Resilience Act (Regulation (EU) 2024/2847), adopted in October 2024, imposes product-level security obligations that will reach open-source suppliers [15], and CISA's Secure-by-Design initiative presses producers toward memory safety and transparency [16]. The question this paper answers is empirical and uncomfortable: *against the xz attack chain specifically, which of these instruments bite, and where?*

2.4 Incident-response precedent

Finally, the response itself is documented and worth studying as the field's most mature trust mechanism: disclosure to the oss-security list with full technical detail [1], immediate distribution advisories and downgrades [2], a CVE record and CISA alert within a day [3][4], and broad ecosystem communication. The contrast between the fragile social trust that admitted the attacker and the robust operational trust that evicted them frames the paper's conclusion.

3. Methodology

We apply a qualitative single-case design with three components.

Primary-source reconstruction (RQ1). The attack chain was reconstructed exclusively from primary sources: the disclosure thread on the oss-security mailing list, including the discoverer's own technical account [1]; the CVE record [3]; distribution advisories and the Red Hat alert [2]; and the CISA alert [4]. Claims not attested by a primary source (e.g., speculative attributions of the attacker to any state actor) are excluded from the reconstruction and flagged as open questions. The reconstruction is normalized into six trust transitions (Section 4.2, Figure 2), each specified with its mechanism, its trust premise, and its evidence.

Counterfactual control analysis (RQ2, RQ3). Six control classes were selected from the 2021–2024 instrument stack (Section 2.3): maintainer-support and criticality programs [14][17]; provenance attestation (SLSA, in-toto, sigstore) [8][9][20]; reproducible builds [13]; SBOM and consumer-side binary analysis [10][11]; runtime isolation and least privilege [16]; and coordinated response and transparency [1][2][4]. For each attack stage, each control was assigned a qualitative coverage level — *strong* (would very likely have prevented, detected, or contained the stage), *partial* (raises cost or narrows exposure under conditions), or *none* (no meaningful effect) — based on documented mechanism behavior, not aspiration. The resulting matrix (Figure 3, Table 2) is an author assessment; we state its subjectivity in Section 7 and mitigate it by tying every assignment to a specific mechanism argument.

Stakeholder-differentiated recommendation synthesis (RQ4). Recommendations were derived by taking each cell of the coverage matrix with level *none* or *partial* and identifying the stakeholder best positioned to act (maintainer, distributor, consumer, funder), then aligning the recommendation with the 2024 policy corpus (SSDF, SLSA, CRA, Secure-by-Design) [9][12][15][16].

4. Case Analysis: The Attack Chain as Trust Transitions (RQ1)

4.1 Chronology

Figure 1 presents the verified chronology, 2021–2024. The attacker's account was created in 2021 under a different identity, and contributions of ordinary quality accumulated through 2022–2023 alongside escalating pressure on the sole maintainer to add co-maintainers and modernize release infrastructure [1].

In early 2024 the attacker held co-maintainer status; in February 2024, xz 5.6.0 and 5.6.1 shipped with the payload; by late March the code had been absorbed by rolling-release distributions; on 29 March 2024 the disclosure ended the operation [1][2][3][4].

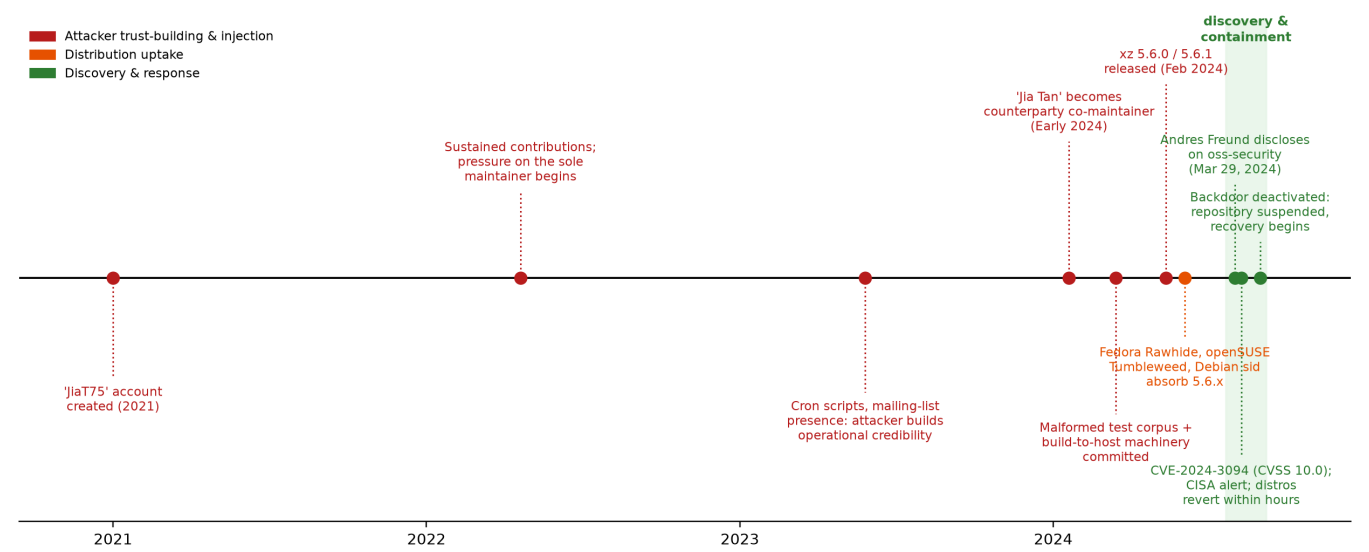


Figure 1. Verified chronology of the xz-utils compromise, 2021–2024 [1][2][3][4].

4.2 The six trust transitions

Figure 2 decomposes the chain into the same six transitions (T1–T6) defined below and consolidated in Table 1, each of which exploited a legitimate trust mechanism.

T1 — Trust acquisition through contribution. Ordinary, useful contributions over years built the committer's standing. Trust premise: sustained technical contribution implies aligned intent. This premise is the field's baseline social contract and is untouchable by technical control [5][7].

T2 — Trust-role escalation. Persistent pressure and demonstrated utility led to co-maintainer authority over the project's build and release machinery. Trust premise: demonstrated competence warrants authority. The maintainer's documented workload and personal circumstances made this transition seem not merely reasonable but humane [1][14].

T3 — Artifact injection inside test fixtures. A malformed test file (bad-3-corrupt_lzma2.xz), presented as a test corpus addition, embedded obfuscated payload data. Trust premise: test files are inert. Reviewers accordingly apply weaker scrutiny to test data than to source code — an assumption the attacker engineered around [1].

T4 — Build-time activation. A modification to the build machinery (build-to-host.m4) caused extraction of an object file from the test fixture at *configure* time on glibc-based GNU/Linux systems, grafting hostile code into liblzma at the consumer's own build. Trust premise: the build system is plumbing. The activation was conditional, architecture-specific, and invisible in the released source diff — it targeted exactly the class of systems (rolling distros with direct builds) most likely to ship it and least likely to have deep review capacity [1].

T5 — Linkage transfer into sshd. On affected distributions, the packaged systemd linked liblzma into OpenSSH's sshd; the backdoor hooked ifunc-based symbol resolution to intercept authentication routines. Trust premise: dependency closure is intentional and reviewed. The attack reached the most security-critical daemon on the system through two unreviewed transitive hops [1][3].

T6 — Dormancy and exposure. The payload reached end-user-facing rolling releases but not enterprise/stable channels; exploitation was blocked on systems where SELinux denied the required memory-protection operations — a runtime control the attacker had not fully engineered around [1]. The chain was severed on 29 March 2024 by disclosure, not by any deployed preventive control.

Transition	Mechanism	Trust premise exploited	Primary source
------------	-----------	-------------------------	----------------

T1 Trust acquisition	Multi-year ordinary contributions under "JiaT75"/"Jia Tan"	Sustained contribution implies aligned intent	[1]
T2 Role escalation	Pressure + utility led to co-maintainer authority over release/build machinery	Demonstrated competence warrants authority	[1]
T3 Artifact injection	Malformed test corpus (bad-3-corrupt_lzma2.xz) embedding obfuscated payload	Test data is inert; weak review depth	[1]
T4 Build-time activation	Modified build-to-host.m4 extracts object file at configure time on glibc GNU/Linux	Build system is trusted plumbing	[1]
T5 Linkage transfer	libsystemd pulls liblzma into sshd; ifunc resolution hooks authentication routines	Dependency closure is reviewed and intentional	[1][3]
T6 Dormancy & exposure	Payload limited to x86-64 glibc rolling-release systems; blocked by SELinux execmem denial on some hosts	Runtime isolation assumed absent	[1]

Table 1. The six trust transitions: mechanism, exploited trust premise, and evidence [1][3].

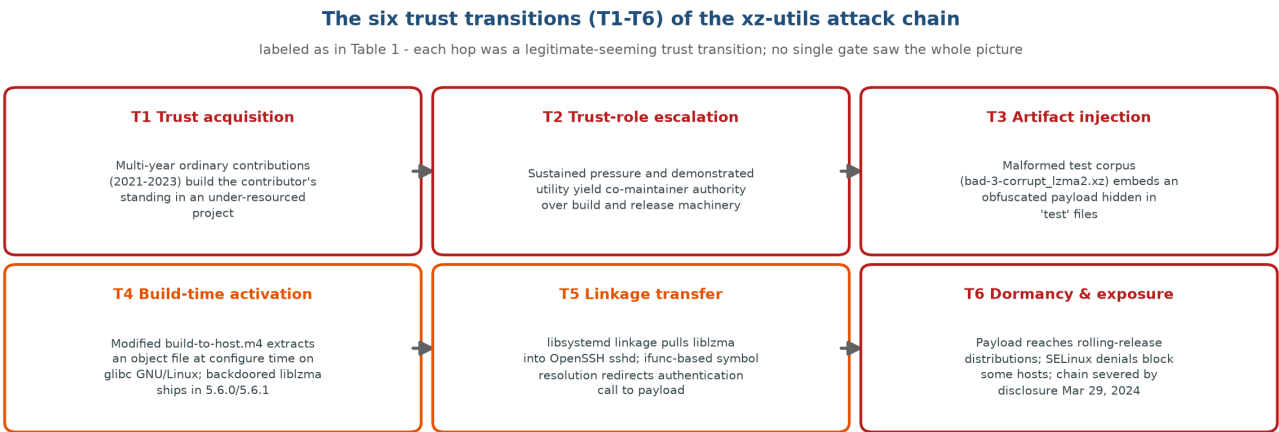


Figure 2. The trust path: the six trust transitions T1-T6 of Table 1, from social engineering to sshd exploitation and containment [1][3].

4.3 What made the attack exceptional

Three properties distinguish CVE-2024-3094 from prior supply-chain incidents [5][6]. First, *patience*: a multi-year, multi-identity social-engineering campaign aimed at one of the most widely deployed codebases in existence. Second, *build-time indirection*: the payload lived outside the reviewable source surface, in test data activated by build machinery on the consumer's machine — defeating source-based review by construction. Third, *precision targeting*: the activation conditions and linkage path were tuned to reach sshd on exactly the systems likely to adopt the compromised release quickly, while leaving stable enterprise channels untouched — a launch strategy that came within days of full deployment.

5. Counterfactual Control Analysis (RQ2, RQ3)

5.1 The coverage matrix

Figure 3 summarizes the analysis: six attack stages against six control classes, with qualitative coverage levels. The stage cut S1-S6 differs deliberately from the trust-transition cut T1-T6 of Section 4.2, because the two decompositions serve different questions: S1 merges T1 and T2, which together constitute the human trust-building layer against which no technical control operates; S2 and S3 correspond to T3 and T4 respectively; S4 (distribution uptake) and S5 (runtime exploitation) decompose T6's exposure phase into the two points at which different control classes act; and S6 adds the post-disclosure recovery stage that the attacker-side chain does not contain but the control analysis requires. Table 2 records the mechanism argument for each assignment. The matrix's headline finding is the diagonal shape of coverage: technical controls cluster around the middle of the chain (build-time activation, distribution uptake), while

its two ends — the social engineering that acquired control, and the human forensics that followed discovery — remain largely uncovered by any deployed instrument.

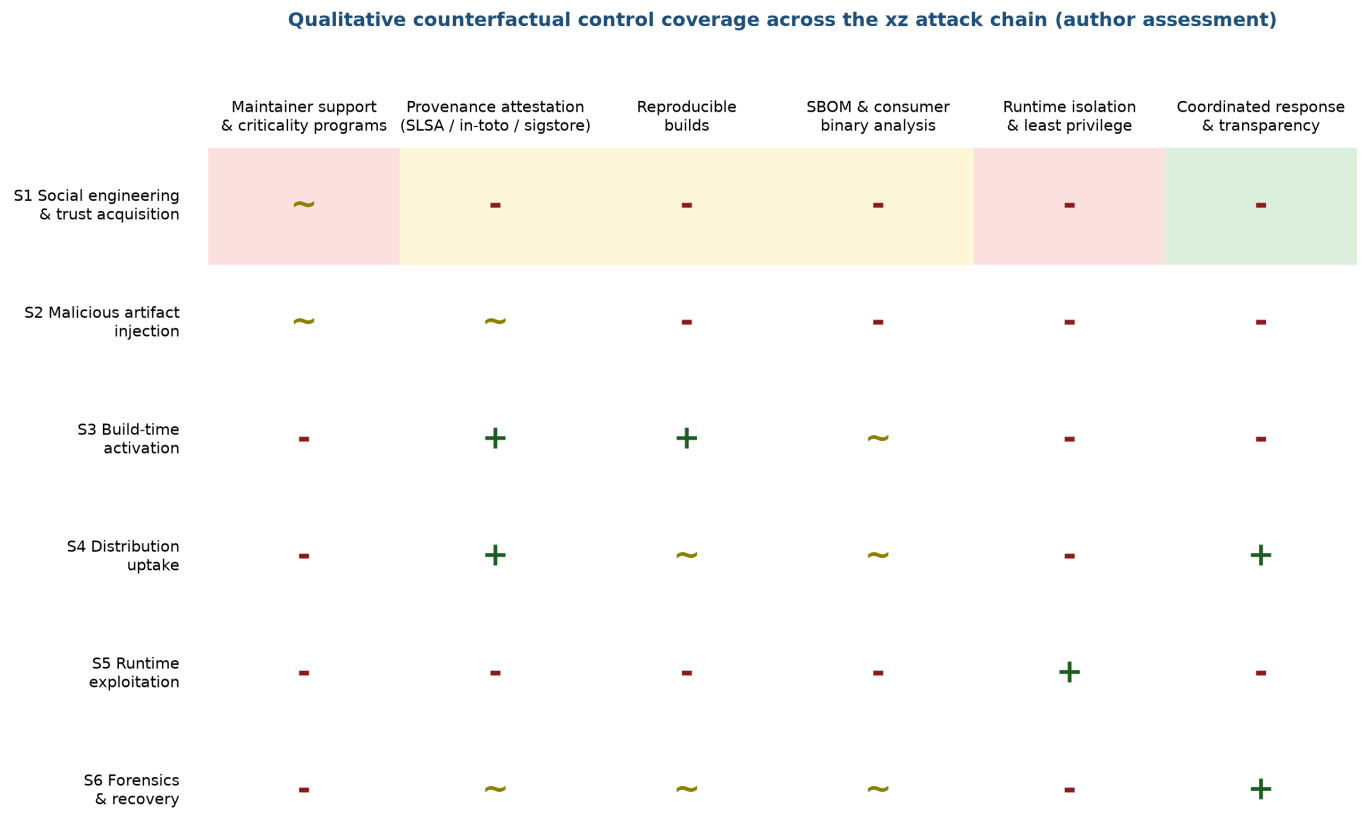


Figure 3. Qualitative counterfactual control coverage: control classes against the six-stage xz attack chain S1-S6 (author assessment; stage cut mapped to the trust transitions T1-T6 of Table 1 in Section 5.1; mechanism arguments in Table 2).

Stage	Control	Level	Mechanism argument
S1	Maintainer support & criticality programs	~	Reduces takeover-prone conditions (funding, co-maintainership, project health) but cannot distinguish patient attacker from genuine volunteer [14][17]
S1	All technical controls	-	Social-layer trust acquisition is upstream of all artifact and build controls [5][7]
S2	Provenance attestation	~	Binds artifacts to process, not maintainer intent; legitimate write access passes [8][9]
S2	Scorecard / project-health signals	~	Sudden role/infrastructure changes are detectable signals, noisy in practice [18]
S2	SBOM / binary analysis (release)	-	Payload dormant in compressed test fixture; invisible to release-level inventory [1][10]
S3	Reproducible builds	+	Independent rebuild exposes divergence between released source and produced binary [13]
S3	SLSA L3+ provenance	+	Build platform attests actual build inputs/outputs; test-fixture extraction becomes visible [9]
S4	Provenance verification at distribution gate	+	Unverifiable or non-attested critical releases quarantined pending rebuild [9][13]
S4	Coordinated response	+	Advisory-and-revert cycle executed within hours once triggered [2][4]
S5	Runtime isolation (SELinux, sandboxing)	+	execmem/permission denials blocked exploitation on configured hosts [1][16]

S6	Response & transparency process	+	Full technical disclosure, CVE with CVSS 10.0, CISA alert, rollbacks within a day [1][3][4]
S6	Provenance attestation	~	Bounds the period of malicious releases; blind to the multi-year human operation [8][9]

Table 2. Mechanism arguments for the counterfactual control coverage assignments.

5.2 Stage-by-stage findings

S1 (Social engineering): partial at best. Maintainer-support and criticality programs — funding, security training, co-maintainer rotation policies, project-health monitoring such as OpenSSF Scorecard signals [14][17][18] — reduce the *conditions* under which a takeover succeeds (an exhausted sole maintainer of hyper-critical code) but cannot distinguish a patient attacker from a genuine volunteer. No 2024 instrument provides verification here; this is the frontier.

S2 (Injection): partial. Provenance attestation binds artifacts to their production process but cannot certify the *intent* of a maintainer with legitimate write access [8][9][20]. Scorecard-style anomaly signals (sudden release-infrastructure changes, unfamiliar committers gaining privileged roles) offer weak, noisy detection [18]. SBOM and binary analysis of the *release* would not see the payload at all, because it was dormant in compressed test data (Section 5.3).

S3 (Build-time activation): strong. This is where the instrument stack bites hardest. A reproducible-builds regime would have exposed the divergence between the released source and the produced binary: the build-to-host machinery would produce non-reproducible artifacts, and any independent rebuild would surface the injected object [13]. SLSA Level 3+ provenance would likewise require the build platform to attest what it actually built and from what inputs, making the test-fixture-to-object-file extraction visible and attestable [9]. This stage is, in other words, *solved in principle* — the gap is adoption: xz was built by distributors and end consumers on heterogeneous hosts with no attestation pipeline.

S4 (Distribution uptake): strong for attestation, partial elsewhere. Verified provenance and reproducible builds at the *distribution* layer would have quarantined 5.6.x pending verification [9][13]; the rapid, coordinated advisory-and-revert cycle that followed disclosure (within hours) demonstrates that the response layer works when triggered [2][4] — but it was triggered by luck, not by controls.

S5 (Runtime exploitation): strong for isolation. On systems where SELinux denied the memory-protection operations the payload required, exploitation failed [1]. Runtime isolation, sandboxing of long-lived daemons, and least-privilege defaults (CISA Secure-by-Design themes [16]) are the last line of defense — and in this incident they were the only preventive control that actually stopped anything.

S6 (Forensics and recovery): strong for process, weak for attribution. The disclosure-quality of the oss-security thread, the immediate CVE assignment (CVSS 10.0), CISA's alert, and distribution rollbacks [1][2][3][4] constitute a functioning recovery layer. What no instrument provided: attribution, or a way to audit the two and a half years of trust-building. Provenance attestation helps here only partially — it can bound the period in which malicious releases existed, not the period in which the human operation ran.

5.3 The SBOM caveat

The most-cited post-incident remedy — "generate SBOMs" — deserves explicit correction. An SBOM inventories the components of a software product [10][11]; xz *would appear* in SBOMs of affected distributions, as it always had. The backdoor was not a rogue dependency or a renamed package; it was inside a legitimate component's legitimate release, hidden in compressed test fixtures and activated at build time. An SBOM therefore neither prevented, detected, nor would even have flagged this attack. SBOMs retain genuine value for vulnerability management (rapid identification of who ships xz 5.6.x) [10] — and indeed, their use in the response was real — but presenting SBOM adoption as the lesson of CVE-2024-3094 mistakes the incident's class. The relevant lessons are process-evidence instruments (reproducible builds, attestation) and human-layer controls.

6. Recommendations (RQ4)

Table 3 states stakeholder-differentiated recommendations. Their logic follows the coverage matrix: invest most where coverage is *none* or *partial*, and convert the *strong* cells from luck-dependent to default-on.

Maintainers and projects should adopt a trust-transition policy: any change to release infrastructure, build machinery, or maintainer roles requires a second opinion — mirroring the two-person integrity rule that cryptographic infrastructure already

applies. Projects should treat test corpora and build scripts as security-relevant code with equivalent review depth. Maintainers of critical components should be *paid*: the OpenSSF Mobilization Plan's funding mechanisms exist and remain underused [14].

Distributors should make SLSA L3+ provenance verification and reproducible-build checks a gate for absorbing new upstream releases of critical packages [9][13], and should sandbox their build farms so that build-time compromise cannot silently propagate. The xz incident showed rolling-release channels are the soft entry point; they need the strongest gates.

Consumers should instrument runtime isolation for long-lived network daemons (the SELinux lesson [1][16]), maintain SBOMs for vulnerability management rather than attack detection [10], and — per the EU Cyber Resilience Act's trajectory [15] — begin requiring process-evidence from suppliers rather than point-in-time audit attestations.

Funders and policymakers should direct resources at the uncovered frontier: programs that monitor the *health and resilience* of hyper-critical, under-resourced projects (criticality × bus-factor ranking per Census II [17]), research into detecting social-engineering campaigns in maintainer communities, and sustained support for reproducible-builds infrastructure across the long tail of packages [13][14].

Stakeholder	Recommendation	Policy alignment
Maintainers / projects	Two-person rule for build machinery, release infrastructure, and maintainer-role changes; security-relevant review depth for test corpora	SLSA [9]; SSDF [12]
Maintainers / projects	Publish project-health signals (bus factor, release-process attestation)	OpenSSF Scorecard [18]
Distributors	Gate absorption of new critical releases on provenance verification and reproducible-build checks; strongest gates on rolling channels	SLSA v1.0 [9]; reproducible builds [13]
Distributors	Harden build farms against build-time compromise propagation	SSDF [12]
Consumers	Enforce runtime isolation and least privilege for long-lived daemons	CISA Secure-by-Design [16]
Consumers	Use SBOMs for vulnerability management, not attack detection; require process evidence from suppliers	NTIA [10]; EU CRA [15]
Funders / policymakers	Fund maintainership of hyper-critical, under-resourced components (criticality x bus-factor ranking)	OpenSSF Mobilization Plan [14]; Census II [17]
Funders / policymakers	Research instrumentation of the social layer: maintainer-workload, role-change review, contribution-pattern anomaly detection	OpenSSF [14]

Table 3. Stakeholder-differentiated recommendations aligned with the 2024 policy corpus [9][10][12][13][14][15][16][17][18].

7. Discussion

7.1 Trust transitions, not trust levels

The deepest lesson of the incident is architectural: trust in open-source supply chains is not a property of artifacts but of *transitions* — from contributor to committer, from committer to release authority, from source to binary, from project to distribution, from distribution to daemon. Each transition in the xz chain was individually unremarkable; their composition was catastrophic. Controls that certify states (signed releases, SBOMs, scorecard badges) leave transitions undefended; controls that certify *processes* (reproducible builds, in-toto attestations, SLSA provenance) defend exactly the transitions where compromise materializes [8][9] [13]. Digital trust for supply chains should therefore be specified, measured, and audited at the level of trust transitions.

7.2 The human layer is the perimeter

The attack succeeded for two and a half years in the only layer with no telemetry: human trust. The field's instrumentation of code (scanners, SAST, SBOM) and of artifacts (signatures, provenance) is mature by comparison. Extending instrumentation to the social layer — maintainer-workload metrics, role-change review, contribution-pattern anomaly detection (the same family of techniques as Manuscript 1's compliance monitoring, applied to repositories) — is the open research frontier that this incident makes impossible to defer. Equally, the sustainable fix is material: fund the humans. Heartbleed demonstrated the cost of unpaid critical infrastructure in

2014 [19]; the OpenSSF Mobilization Plan specified the response in 2022 [14]; xz demonstrated in 2024 that the response had not reached the project it was designed for.

7.3 The luck problem

The discovery depended on an investigator with rare domain depth noticing a half-second performance anomaly — an outcome no control produced. The design goal for the field should be that next time, discovery is *structural*: a reproducible-builds mismatch, an attestation gap, a sandbox violation — each of which converts "someone happened to notice" into "the pipeline refuses to pass." The response layer's demonstrated speed [2][4] then becomes the backstop rather than the whole story.

8. Limitations and Threats to Validity

Construct validity. The six-stage decomposition and the coverage levels are analytical constructs; a different decomposition could shift individual cell assignments, and the levels are qualitative judgments by the authors, mitigated — but not eliminated — by grounding each in a specific mechanism argument (Table 2). **Internal validity.** The reconstruction excludes unverified claims by design; however, some public accounts of the incident (e.g., attribution speculation, full extent of targeted architectures) remain unsettled, and our analysis inherits that uncertainty. **External validity.** One incident, however rich, cannot establish which controls generalize; the counterfactual analysis is a hypothesis-generating mapping, not an empirical effectiveness study. The recommendation priorities assume GNU/Linux distributor-consumer topology and may vary across ecosystems (npm, PyPI, container registries). **Reliability.** Primary-source dependence is a strength for facts but a constraint on scope: the attacker's internal decision-making is necessarily inferred from observed actions. **Temporal scope.** The analysis reflects instruments and advisories available through the fourth quarter of 2024; subsequent enforcement details of the EU Cyber Resilience Act and successor U.S. policy are out of scope.

9. Conclusion

The xz-utils backdoor was not defeated by the supply-chain security stack — it was discovered by a curious performance anomaly, days before reaching stable systems at scale. Reading the incident as six legitimate-seeming trust transitions yields a precise map of where the field's instruments work (build-time verification, runtime isolation, coordinated response), where they merely help (artifact attestation, distribution gating), and where they are absent (the human layer that admitted the attacker). The foundations of software supply-chain trust, this incident teaches, rest on three pillars that must be built deliberately: *funded and resilient maintainership* for the hyper-critical infrastructure nobody pays for; *verifiable process evidence* — reproducible builds and provenance attestation by default — at every trust transition; and *isolation by design* so that the last line of defense is an architecture rather than an accident. None of these existed, fully formed, for xz-utils in March 2024. Building them is the work the incident has assigned.

Declarations

Funding. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

CRedit contribution statement. Bin Cao: conceptualization, investigation, case reconstruction, writing — original draft, writing — review & editing. Yishuai Chen: conceptualization, validation, counterfactual analysis, writing — review & editing, supervision.

Competing interests. The authors declare no competing interests.

Ethics and dual-use. This paper documents publicly disclosed information only. It does not reproduce the backdoor's technical implementation details beyond what is necessary for defensive analysis, contains no exploit material or live credentials, and is consistent with the coordinated-disclosure record already published in [1][3][4].

Data and code availability. No new datasets were generated. All factual claims are traceable to the cited primary sources [1][2][3][4]; the coverage matrix assignments are qualitative assessments documented in Table 2.

AI assistance declaration. The authors used AI-based tools to assist with literature discovery, source verification, figure preparation, and manuscript drafting; all content was reviewed, verified against primary sources, and approved by the authors, who take full responsibility for the work.

References

[1] A. Freund, "xz backdoor: timely disclosure post," oss-security mailing list (Openwall), Mar. 29, 2024. [Online]. Available: <https://www.openwall.com/lists/oss-security/2024/03/29/4>

- [2] Red Hat, "Urgent Security Alert: Fedora Linux and the xz compression library," Red Hat Security Blog, Mar. 29, 2024. [Online]. Available: <https://www.redhat.com/en/blog/urgent-security-alert-fedora-41-and-xz>
- [3] National Institute of Standards and Technology, "CVE-2024-3094," National Vulnerability Database, 2024. [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2024-3094>
- [4] Cybersecurity and Infrastructure Security Agency, "CISA Statement on Malware Impacting XZ Utils Data Compression Library," Mar. 29, 2024. [Online]. Available: <https://www.cisa.gov/news-events/alerts/2024/03/29/statement-malware-impacting-xz-utils-data-compression-library>
- [5] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in Proc. 2023 IEEE Symposium on Security and Privacy (S&P), 2023. doi: 10.1109/SP46215.2023.10179308.
- [6] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: a review of open source software supply chain attacks," in Proc. 17th Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020), LNCS 12230, 2020. doi: 10.1007/978-3-030-52668-9_3.
- [7] M. Zimmermann, C. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: a study of security threats in the npm ecosystem," in Proc. 28th USENIX Security Symposium, 2019. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity19/presentation/zimmermann>
- [8] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: providing farm-to-table guarantees for bits and bytes," in Proc. 28th USENIX Security Symposium (USENIX Security 19), pp. 1393-1410, 2019. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias>
- [9] Open Source Security Foundation (OpenSSF), "SLSA Specification v1.0," 2023. [Online]. Available: <https://slsa.dev/spec/v1.0/>
- [10] National Telecommunications and Information Administration, "The Minimum Elements for an Software Bill of Materials (SBOM)," NTIA, Jul. 2021. [Online]. Available: <https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom>
- [11] Executive Order 14028, "Improving the Nation's Cybersecurity," 86 FR 26633, May 12, 2021.
- [12] National Institute of Standards and Technology, "Secure Software Development Framework (SSDF) v1.1," NIST SP 800-218, Feb. 2022. doi: 10.6028/NIST.SP.800-218.
- [13] C. Lamb and S. Zacchiroli, "Reproducible builds: increasing the integrity of software supply chains," IEEE Software, vol. 39, no. 2, pp. 62-70, Mar./Apr. 2022. doi: 10.1109/MS.2020.2996796 (early access 2020).
- [14] Open Source Security Foundation (OpenSSF), "The Open Source Software Security Mobilization Plan," 2022. [Online]. Available: <https://openssf.org/getinvolved/mobilization-plan/>
- [15] Regulation (EU) 2024/2847 of the European Parliament and of the Council laying down measures for a high common level of cybersecurity across the Union (Cyber Resilience Act), adopted 23 October 2024, published in the Official Journal of the European Union, 20 November 2024.
- [16] Cybersecurity and Infrastructure Security Agency, "Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Security-by-Design and -Default," Apr. 2023. [Online]. Available: <https://www.cisa.gov/resources-tools/resources/shifting-balance-cybersecurity-risk-principles-and-approaches-security-design-and>
- [17] F. Nagle, J. Dana, J. Hoffman, S. Randazzo, and Y. Zhou, "Census II of Free and Open Source Software - Application Libraries," Linux Foundation and the Laboratory for Innovation Science at Harvard, Mar. 2022. doi: 10.70828/KHEH5209. [Online]. Available: <https://www.linuxfoundation.org/research/census-ii-of-free-and-open-source-software-application-libraries>
- [18] Open Source Security Foundation (OpenSSF), "Scorecard: Security health metrics for open source projects," 2023. [Online]. Available: <https://securityscorecards.dev>
- [19] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of Heartbleed," in Proc. ACM Internet Measurement Conference (IMC 2014), 2014. doi: 10.1145/2663716.2663755.
- [20] Open Source Security Foundation (OpenSSF), "sigstore: software signing transparency," 2024. [Online]. Available: <https://www.sigstore.dev>