POLARIS INTELLIGENCE

Technical Due Diligence Report

XZ

03 April 2026

Ref: f0b0349e

CONFIDENTIAL

1. Executive Summary

Repository Classification: Library (Confidence: medium)
This repository is a C/C++ library. Package manifest detection is not applicable for C/C++ ecosystems. Dependency findings reflect the library's own dependencies, not a consuming application.

Copyleft licence contamination detected. Third-party GPL/AGPL-licensed code found that may impose open-source obligations.

Bus factor: Tier 3: Critical Concentration (capped from high — library archetype). Review recommended to assess key-person dependency.

1 known supply chain incident involving this package directly (1 sabotage). See Dependencies section for details.

The table below rates risk across 13 dimensions, from Clean (no findings) to Critical (potential deal impact). Together they form the technical risk profile of the target asset.

CATEGORY	RATING	SUMMARY
Secrets & Credentials	CLEAN	No credential exposure detected in scanned files
Dependency Vulnerabilities	CLEAN	No dependency surface — no third-party packages detected
Supply Chain Risk	CRITICAL RISK	1 critical this project incident
Licence & IP	CRITICAL RISK	Third-party AGPL/copyleft contamination detected
Developer Concentration	MEDIUM RISK	Tier 3: Critical Concentration (capped from high — library archetype)
Code Quality	CLEAN	Grade B (74.0/100)
Architecture	CLEAN	24 isolated modules (standalone utilities, examples, or platform-specific code)
Malware / Destructive Code	CLEAN	No suspicious patterns detected in scanned files
Test Coverage	LOW RISK	Moderate test coverage (9% test ratio, CI gates active) — below comprehensive threshold

CATEGORY	RATING	SUMMARY
Infrastructure & Deployment	CLEAN	Limited deployment infrastructure (expected for library) (capped from low — library archetype)
Technical Debt	CLEAN	Minimal tech debt (77 markers, 1.4/KLOC)
Governance & CI/CD	HIGH RISK	Significant governance deficiencies (Grade D, 3.6/10)
Engineering Maturity	LOW RISK	Fair maturity (6.4 releases/yr, 20% community health) — some process gaps

2. Transaction Impact Assessment

This section translates technical findings into their commercial implications for the transaction. Ratings range from Clean (no concern) to Critical (potential deal-breaker), with specific conditions that may need to be met before or after completion.

CATEGORY	ASSESSMENT	DETAIL
Security Exposure	CRITICAL RISK	1 critical this project incident
Operational Risk	MEDIUM RISK	Tier 3: Critical Concentration (capped from high — library archetype); Grade B (74.0/100)
IP & Licence Risk	CRITICAL RISK	Third-party AGPL/copyleft contamination detected
Integration Complexity	CLEAN	24 isolated modules (standalone utilities, examples, or platform-specific code)
Maintenance Burden	CLEAN	Grade B, 2 quality issues, 77 tech debt markers

Remediation Effort Estimate

1 critical area requiring immediate action; 3 high-severity areas for short-term remediation; 2 medium-severity items for the integration roadmap; operational concerns in governance requiring ongoing practice change.

Estimates assume a senior developer familiar with the technology stack. Actual effort may vary based on codebase familiarity and organisational context.

3. Scope & Methodology

Repository: https://github.com/tukaani-project/xz

Analysis date: 03 April 2026

Codebase size: 285 files, 71,787 lines

LANGUAGE	LINES
C	44,481
C/C++ Header	19,412
Text	5,175
Shell	1,932
YAML	787

Methodology

This report was produced by Polaris Intelligence automated analysis pipeline. The following scanners were applied:

1. **GitHub Enrichment** — project metadata, release cadence, community health
2. **Secret Scanner** — regex pattern matching + context classification
3. **Dependency Scanner** — manifest parsing + OSV vulnerability cross-reference + exploitability analysis
4. **Supply Chain Intelligence** — cross-reference against known supply chain incidents
5. **Licence Auditor** — declared licence + source header contradiction detection
6. **Bus Factor Analysis** — 5-tier developer concentration taxonomy (24-month window)
7. **Code Quality Scorer** — cyclomatic complexity, duplication, security anti-patterns
8. **Architecture Mapper** — import graph, circular dependencies, module coupling
9. **Engineering Maturity** — release discipline, community governance, project signals
10. **Malware Heuristic** — destructive actions, crypto mining, exfiltration, obfuscation
11. **Governance & CI/CD** — OpenSSF Scorecard, branch protection, dependency management

Note: *File counts may vary between sections because each scanner operates on a different subset of files (e.g. quality analysis covers source code files only, while the scope total includes configuration, documentation, and data files).*

4. Secrets & Credentials CLEAN

Hardcoded credentials — API keys, database passwords, tokens — are the most common cause of data breaches. Their presence indicates both an immediate security exposure and a gap in the target's engineering practices that transfers with the acquisition.

No hardcoded secrets or credentials detected.

5. Dependencies & Supply Chain CRITICAL RISK

Modern software relies on hundreds of third-party packages. Known vulnerabilities in these dependencies are publicly catalogued and actively exploited. Unpatched critical CVEs represent a quantifiable security liability that transfers to the acquirer.

0 dependencies analysed across 0 manifests.

Note: *The section rating of* CRITICAL RISK *is driven by supply chain intelligence findings (see below), not by dependency vulnerabilities.*

No known vulnerabilities detected.

Note: *OpenSSF Scorecard reports 4 vulnerabilities via OSV. Polaris found 0 after filtering to versions declared in the manifest. This discrepancy typically arises because OSV includes advisories for version ranges that do not match the pinned versions in this repository.*

Supply Chain Intelligence: This Project

This project was the subject of a known supply chain incident. *This is not a dependency issue — the scanned repository itself has a documented incident history. Acquirers should assess reputational impact and residual technical risk.*

CRITICAL RISK **xz** (system) — Sabotage

Sophisticated multi-year backdoor planted in xz/liblzma compression library by a trusted contributor (Jia Tan). Backdoor targeted OpenSSH via systemd, enabling remote code execution. Discovered by Andres Freund before widespread deployment.

Date: 2024-03-29 | Incident affected versions: 5.6.0, 5.6.1 | CVE-2024-3094

Malware scan reconciliation: *The malware heuristic scanner reports* ***clean*** *because the malicious code from the documented incident has since been removed. The supply chain rating reflects the historical incident record, not the current code state. Both findings are correct and complementary.*

6. Developer Concentration (Bus Factor) MEDIUM RISK

"Bus factor" measures how many developers would need to leave before critical knowledge is lost. High concentration in one developer creates key-person dependency — a material operational risk that can delay integration and increase post-acquisition costs.

Tier 3: Critical Concentration — Bus factor score: **96.4%** [risk capped to MEDIUM for library archetype]

Despite 24 contributors (7 active), a single developer has authored the majority of commits in 96.4% of core files. Effective bus factor is 1. Code review practices and pair programming should be introduced to distribute knowledge more evenly.

Top Contributors

DEVELOPER	COMMITS	OWNED FILES	CORE %	STATUS
Lasse Collin	614	234	96%	Active
Sam James	17	1	1%	Departed
Christoph Junghans	3	0	0%	Departed
Collin Funk	2	1	1%	Active
Radek Zikmund	2	0	0%	Active
Dexter Castor Döpping	2	1	1%	Departed
Tobias Stoeckmann	2	0	0%	Departed
Christian Weisgerber	2	0	0%	Departed
Salman Muin Kayser Chishti	1	1	0%	Active
Nobuhiro Iwamatsu	1	0	0%	Active

Departed Developer Risk

DEVELOPER	MONTHS INACTIVE	CORE FILES OWNED	RISK
Simon Josefsson	7	2	MEDIUM RISK
Sam James	11	1	MEDIUM RISK
Dexter Castor Döpping	15	1	MEDIUM RISK
Firas Khalil Khana	19	1	MEDIUM RISK
Xi Ruoyao	21	1	MEDIUM RISK
RainRat	22	1	MEDIUM RISK

7. Licence & Intellectual Property CRITICAL RISK

Copyleft licences (GPL, AGPL) require derivative works to be released under the same open-source terms. If copyleft code is embedded in a proprietary product, the acquirer may face an obligation to open-source their own code — or costly remediation to replace the affected components.

Declared Licences

SPDX ID	RISK	SOURCE	FILE
GPL-3.0	HIGH RISK	licence_file	`COPYING`

Project Licensing Posture

The following reflects the project's own chosen licence. This is not contamination — it describes the terms under which this software is distributed.

LICENCE	POSTURE	INVESTOR IMPLICATIONS
GPL-3.0	Project is licensed under GPL-3.0 — Strong copyleft — derivative works must be distributed under same licence terms	This project is distributed under GPL-3.0 (strong copyleft). Derivative works must be distributed under the same licence. Acquirers should verify…

Licence Risk Findings

SEVERITY	TYPE	LICENCE	DETAIL	RECOMMENDATION
MEDIUM RISK	Contradiction	LGPL-2.1	9 files have LGPL-2.1 headers but project declares GPL-3.0 — e.g. lib/getopt-pfx-core.h, lib/getopt-cdefs.h…	LGPL (LGPL-2.1) allows linking without copyleft obligation, but modifications to LGPL-covered files must be shared…
HIGH RISK	Contradiction	GPL-2.0	1 files have GPL-2.0 headers but project declares GPL-3.0 — extra/scanlzma/scanlzma.c	GPL (GPL-2.0) contamination means derivative works must be GPL-licensed. Options: (1) replace with a permissive…

8. Code Quality & Technical Debt CLEAN

Code quality directly predicts the cost and speed of post-acquisition development. A low grade signals elevated technical debt — higher bug rates, slower feature delivery, and more expensive onboarding for new developers joining after the transaction.

Quality grade: **B (74.0/100)** — Acceptable

Grade Scale

GRADE	MEANING
A	Excellent maintainability
B	Good engineering quality
C	Moderate technical debt
D	High technical debt
F	Severe structural risk

METRIC	VALUE
Total files	234
Code lines	30,522
Functions	826
Types/Structs	184
Avg function length	36.7 lines
Avg complexity	5.4

Technical Debt Indicators

The following indicators are informational. At this grade level, they represent normal characteristics of a healthy codebase rather than actionable concerns. Duplication above automated thresholds is common in test files and generated code.

- 20 files exceed 500 lines
- 20 functions with high cyclomatic complexity

Complexity Hotspots

SEVERITY	FILE	FUNCTION	COMPLEXITY
HIGH RISK	`src/xz/args.c`	parse_real	81
HIGH RISK	`src/liblzma/common/stream_decoder_mt.c`	stream_decode_mt	76

SEVERITY	FILE	FUNCTION	COMPLEXITY
HIGH RISK	`src/xz/coder.c`	coder_set_compression_settings	66
HIGH RISK	`src/liblzma/lzma/lzma_decoder.c`	lzma_decode	61
HIGH RISK	`src/liblzma/common/common.c`	lzma_code	57
HIGH RISK	`lib/getopt.c`	_getopt_internal_r	51
HIGH RISK	`src/liblzma/common/string_conversion.c`	anonymous	48
HIGH RISK	`src/liblzma/lzm..._encoder_optimum_normal.c` [F1]	helper2	48
HIGH RISK	`src/liblzma/common/file_info.c`	file_info_decode	44
HIGH RISK	`lib/getopt.c`	process_long_option	42

Large Files (>500 lines)

FILE	TOTAL LINES	CODE LINES
`src/liblzma/common/stream_decoder_mt.c`	2005	911
`src/xz/file_io.c`	1486	859
`src/xz/coder.c`	1477	857
`src/liblzma/common/string_conversion.c`	1364	802
`src/xz/list.c`	1358	891
`src/xz/message.c`	1327	852
`src/liblzma/common/index.c`	1290	759
`src/liblzma/common/stream_encoder_mt.c`	1279	737
`src/liblzma/lzma/lzma_decoder.c`	1264	688
`src/liblzma/api/lzma/container.h`	994	97

No security anti-patterns detected.

9. Architectural Assessment CLEAN

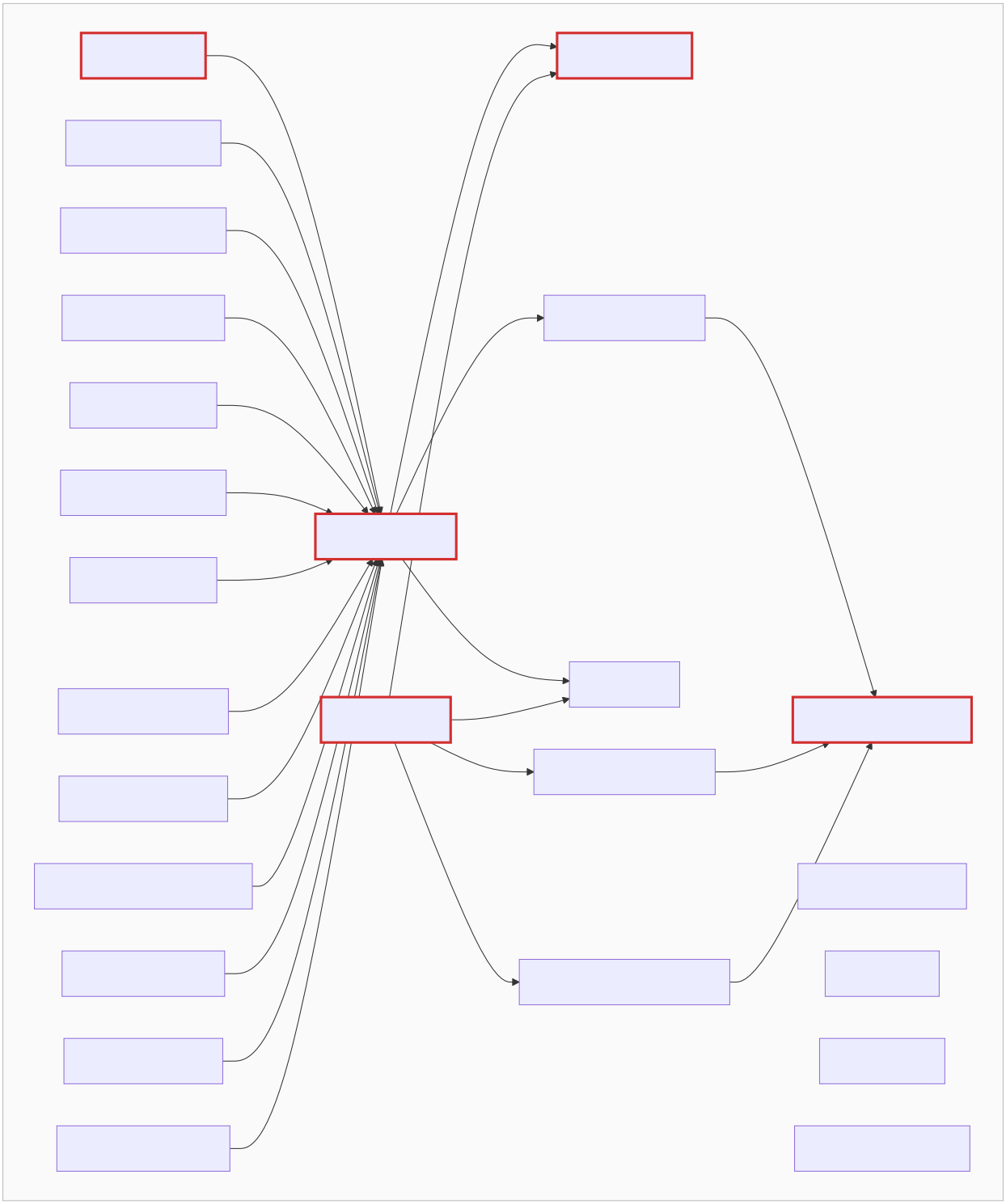
225 production modules, 308 internal dependencies, 1 external imports.

Change impact zones: 5 modules have high fan-in (many dependents). Top modules: `src/liblzma/common/common.h, src/liblzma/check/check.h, src/xz/private.h`. Changes to these modules carry elevated regression risk.

24 isolated modules detected (no internal imports or dependents). These are typically standalone utilities, examples, platform-specific implementations, or generated code.

Module Dependency Map

Modules are grouped by architectural layer and coloured accordingly. Arrows show import dependencies. Numbers in parentheses indicate how many other modules depend on each file. Thick red borders highlight high fan-in modules (change risk).



Layer Distribution

Modules are grouped by their role in the application: business logic (core functionality), presentation (user-facing), data (storage), and test. A well-structured codebase separates these concerns clearly.

LAYER	MODULES
other	128
infrastructure	97

Entry Points (24)

LOCATION	PATTERN
`src/xzdec/xzdec.c:384`	`int main(`
`src/lzmainfo/lzmainfo.c:194`	`int main(`
`src/xz/main.c:156`	`int main(`
`src/liblzma/rangecoder/price_tablegen.c:87`	`int main(`
`src/liblzma/lzma/fastpos_tablegen.c:20`	`int main(`
`src/liblzma/check/crc64_tablegen.c:83`	`int main(`
`src/liblzma/check/crc_clmul_consts_gen.c:89`	`int main(`
`src/liblzma/check/crc32_tablegen.c:108`	`int main(`
`lib/getopt.c:760`	`int main (`
`lib/getopt1.c:77`	`int main (`

High Fan-In Modules (Change Risk)

“Fan-in” measures how many other parts of the codebase depend on a given file. High fan-in files are widely relied upon — changes to them carry elevated risk because they can affect many dependent components.

MODULE	DEPENDENTS	RISK
`src/liblzma/common/common.h`	42	HIGH RISK
`src/liblzma/check/check.h`	15	MEDIUM RISK
`src/xz/private.h`	13	MEDIUM RISK
`src/common/sysdefs.h`	13	MEDIUM RISK
`src/common/tuklib_common.h`	11	MEDIUM RISK

Isolated Modules (24)

These files neither use nor are used by any other file in the codebase. They may be unused code, standalone utilities, or components loaded indirectly. Each should be reviewed to confirm it is genuinely needed.

- `debug/testfilegen-arm64.c`
- `doc/examples/01_compress_easy.c`
- `doc/examples/02_decompress.c`
- `doc/examples/03_compress_custom.c`
- `doc/examples/04_compress_easy_mt.c`
- `doc/examples/11_file_info.c`
- `dos/config.h`
- `extra/scanlzma/scanlzma.c`
- `lib/getopt-cdefs.h`
- `lib/getopt-core.h`
- `lib/getopt-ext.h`
- `lib/getopt-pfx-core.h`
- `lib/getopt-pfx-ext.h`
- `lib/getopt.in.h`
- `src/liblzma/check/crc32_arm64.h`

9b. Architecture Topology

This section applies graph-theoretic analysis to the module dependency network identified in the Architectural Assessment. Community detection reveals natural subsystem boundaries; centrality analysis identifies critical bridge modules whose failure or refactoring would disproportionately affect the codebase.

METRIC	VALUE
Modules analysed	225
Internal dependencies	308
Subsystems detected	10
Modularity score	0.668 (strong modular structure)
Unclustered modules	24

Subsystem Overview

Communities are groups of modules that are more tightly connected to each other than to the rest of the codebase. Each represents a natural subsystem boundary.

SUBSYSTEM	MODULES
src/liblzma/ (infrastructure, 40 modules)	40
src/xz/ (other, 29 modules)	29
src/ (other, 26 modules)	26
src/liblzma/ (other, 19 modules)	19
src/common/ (infrastructure, 23 modules)	23
src/liblzma/ (other, 17 modules)	17
src/liblzma/ (other, 10 modules)	10
lib/ (infrastructure, 3 modules)	3
src/liblzma/ (infrastructure, 17 modules)	17
src/liblzma/ (other, 17 modules)	17

Bridge Modules (Bottleneck Risk)

Bridge modules sit on critical paths between subsystems. High betweenness centrality means many inter-module communication paths pass through this module — changes here carry elevated risk of cascading failures.

MODULE	CENTRALITY	BRIDGES	FAN-IN / OUT	RISK
`src/liblzma/common/common.h`	0.0448	7	42 / 4	LOW RISK
`src/liblzma/api/lzma.h`	0.0376	3	9 / 14	LOW RISK

God Modules (Excessive Coupling)

“God modules” have direct dependencies spanning many subsystems. They violate separation of concerns and make the codebase harder to modify safely — changes risk unintended side effects across multiple subsystems.

MODULE	SUBSYSTEMS	CROSS-EDGES	TOTAL DEGREE	RISK
`src/liblzma/common/common.h`	7	26	46	HIGH RISK

Structural Gaps

Subsystem pairs with very few connecting dependencies that may indicate missing integration.

SUBSYSTEM A	SUBSYSTEM B	CONNECTING EDGES
src/liblzma/ (infrastructure, 40 modules)	src/liblzma/ (other, 10 modules)	0
src/liblzma/ (infrastructure, 40 modules)	lib/ (infrastructure, 3 modules)	0
src/xz/ (other, 29 modules)	src/liblzma/ (other, 19 modules)	0
src/xz/ (other, 29 modules)	src/liblzma/ (other, 17 modules)	0
src/xz/ (other, 29 modules)	src/liblzma/ (other, 10 modules)	0
src/xz/ (other, 29 modules)	lib/ (infrastructure, 3 modules)	0
src/xz/ (other, 29 modules)	src/liblzma/ (infrastructure, 17 modules)	0
src/xz/ (other, 29 modules)	src/liblzma/ (other, 17 modules)	0
src/ (other, 26 modules)	src/liblzma/ (other, 19 modules)	0
src/ (other, 26 modules)	src/liblzma/ (other, 17 modules)	0

Topology Findings

HIGH RISK **God Module:** Module 'src/liblzma/common/common.h' connects 7 subsystems with 26 cross-boundary edges. Changes here risk cascading across the codebase.

10. Test Coverage LOW RISK

Automated tests act as safety nets for the codebase. When developers make changes, tests verify nothing else has broken. Strong test coverage means the acquirer's team can modify and extend the code with confidence; weak or absent testing means changes carry a higher risk of introducing undetected problems.

Some testing in place, but gaps remain — coverage is partial or CI enforcement is missing, limiting the safety net for code changes.

METRIC	VALUE
Test files	21
Source files	246
Test-to-source ratio	8.5%
Test functions / cases	0

This is a low ratio. Most of the codebase has no automated tests, meaning changes cannot be verified automatically.

CI configuration detected *(.github/workflows/freebsd.yml, .github/workflows/openbsd.yml, .github/workflows/solaris.yml, .github/workflows/netbsd.yml, .github/workflows/dragonflybsd.yml, .github/workflows/msys2.yml, .github/workflows/msvc.yml, .github/workflows/ci.yml, .github/workflows/coverity.yml, .github/workflows/cifuzz.yml, .github/workflows/haiku.yml), but test enforcement could not be fully confirmed. The acquirer should verify that tests run automatically on all pull requests.*

11. Infrastructure & Deployment CLEAN

This section assesses whether the deployment process — how the software is built, packaged, and released — is documented in code or relies on undocumented manual steps. Codified deployment means a new team can operate the software independently; undocumented deployment creates dependency on the original developers and increases transition risk.

Limited build automation detected. Package publication may rely on manual steps.

Infrastructure Found

CATEGORY	TOOLS / CONFIGURATION
CI/CD Automation	github-actions (11)

> **As a library distributed via package registry, containerisation, orchestration, and deployment infrastructure are not expected. CI/CD for build and publish is the primary infrastructure concern.**

12. Technical Debt CLEAN

Technical debt represents shortcuts, deferred work, and known problems that the development team has acknowledged but not yet fixed. Every codebase carries some debt; what matters is the volume and severity. High technical debt increases the cost of post-acquisition development and the risk that changes introduce new problems.

Minimal technical debt — the codebase is well-maintained with few acknowledged shortcuts or deferred work items.

Developers have left **77** notes in the code flagging work that needs to be done (TODO items, known bugs, temporary workarounds). That is **1.4 markers per 1,000 lines of code**. This density is typical for a well-maintained project.

Debt Markers by Type

TYPE	COUNT
TEMP	28
FIXME	25
TODO	13
HACK	8
WORKAROUND	3

13 markers are planned work items (TODO) while 36 flag known problems (FIXME, HACK, BUG). A high proportion of FIXME/HACK markers is more concerning than TODOs, as they indicate acknowledged broken or fragile code.

13. Governance & CI/CD Security HIGH RISK

Governance measures the security and maturity of development processes — CI pipeline hardening, release signing, code review enforcement, and dependency management. Weak governance increases post-acquisition remediation costs and ongoing operational risk.

Overall governance: D (3.6/10) [OpenSSF Scorecard + local analysis]

OpenSSF Scorecard aggregate: 5.4/10

CI Pipeline Security — Grade F

CHECK	SCORE	SOURCE	DETAIL
Token-Permissions	N/A	scorecard	No tokens found
Pinned-Dependencies	N/A	scorecard	no dependencies found
Dangerous-Workflow	N/A	scorecard	no workflows found
CI-Tests	N/A	scorecard	no pull request found
SAST	0/10	scorecard	no SAST tool detected

CI pipelines have significant security weaknesses requiring remediation. No static analysis (SAST) detected — consider CodeQL, Semgrep, or similar.

Release Engineering — Grade A

CHECK	SCORE	SOURCE	DETAIL
Signed-Releases	8/10	scorecard	5 out of the last 5 releases have a total of 5 signed artifacts.
Packaging	N/A	scorecard	packaging workflow not detected
Maintained	10/10	scorecard	30 commits and 7 issue activity found in the last 90 days -- score normalized to 10
release_frequency	6/10	enrichment	6.4 releases/year, 100% semver compliant

Release engineering practices are mature.

Governance Posture — Grade B

CHECK	SCORE	SOURCE	DETAIL
Security-Policy	10/10	scorecard	security policy file detected
Contributors	10/10	scorecard	project has 8 contributing companies or organizations
CII-Best-Practices	0/10	scorecard	no effort to earn an OpenSSF best practices badge detected
License	9/10	scorecard	license file detected

Governance posture is adequate with some gaps.

Branch Protection & Code Review — Grade F

CHECK	SCORE	SOURCE	DETAIL
Branch-Protection	0/10	scorecard	branch protection not enabled on development/release branches
Code-Review	0/10	scorecard	Found 0/30 approved changesets -- score normalized to 0

Weak or absent branch protection — code can be pushed without review. Significant proportion of changes merged without peer review.

Dependency Management — Grade C

CHECK	SCORE	SOURCE	DETAIL
Dependency-Update-Tool	0/10	scorecard	no update tool detected
Vulnerabilities	6/10	scorecard	Scorecard reports 4 via OSV; Polaris found 0 after filtering to versions in the manifest.
Fuzzing	10/10	scorecard	project is fuzzed

Dependency management has gaps — automated updates or vulnerability tracking missing.

14. Engineering Maturity LOW RISK

Engineering maturity measures the project's operational health beyond source code quality — release discipline, community governance, and project signals that indicate long-term viability.

Overall maturity: **B** — minor maturity gaps only (risk rating: LOW)

Release Cadence

METRIC	VALUE
Releases per year	6.4
Days since last release	2
Semver compliance	100.0%
Grade	A

Community Health

DOCUMENT	STATUS
SECURITY.md	Present
CONTRIBUTING.md	Missing
CODE_OF_CONDUCT	Missing
Issue template	Missing

DOCUMENT	STATUS
PR template	Missing
Health score	20%
Grade	F

Project Signals

METRIC	VALUE
Stars	1,479
Forks	225
Open issues	18
Contributors	64
Repository created on GitHub	2022-10-18
Grade	A

Note: *GitHub API reports 64 contributors while git history analysis (Bus Factor section) identified 24. This discrepancy arises because GitHub counts all commit authors across the full history, while git analysis may use a limited clone depth or different author-deduplication rules.*

15. Malware & Destructive Action Scan CLEAN

This scan searches for code patterns commonly associated with protestware, supply-chain attacks, and sabotage — filesystem wipers, obfuscated payloads, unauthorised network calls, and install-hook abuse. Findings are heuristic and warrant manual review rather than automatic condemnation.

Files scanned: 156

No suspicious patterns detected.

16. Risk Summary & Recommendations

Recommendations are prioritised by their potential impact on the transaction. Immediate and Urgent items should be addressed as conditions precedent; Medium items can be scheduled into the post-acquisition integration roadmap.

PRIORITY	CATEGORY	RECOMMENDED ACTION
URGENT	Licence Contamination	Engage legal counsel to assess copyleft exposure. Consider replacing GPL/AGPL dependencies with permissive alternatives.
URGENT	Supply Chain Risk	This project was the subject of 1 known supply chain incident. Assess reputational impact, verify remediation completeness, and review residual risk.
MEDIUM RISK	Governance & CI/CD	Governance grade D — review branch protection, code review enforcement, and CI pipeline hardening.

Appendix A: Raw Data

Module Dependencies (top 30)

MODULE	FAN-IN	LAYER
`src/liblzma/common/common.h`	42	infrastructure
`src/liblzma/check/check.h`	15	other
`src/common/sysdefs.h`	13	infrastructure
`src/xz/private.h`	13	other
`src/common/tuklib_common.h`	11	infrastructure
`src/liblzma/simple/simple_private.h`	9	other
`src/liblzma/api/lzma.h`	9	other
`src/liblzma/common/index.h`	8	infrastructure
`src/liblzma/common/easy_preset.h`	6	infrastructure
`src/liblzma/lzma/lzma_decoder.h`	6	other
`src/liblzma/common/filter_encoder.h`	6	infrastructure
`src/common/tuklib_integer.h`	5	infrastructure
`src/liblzma/lz/lz_decoder.h`	5	other
`src/liblzma/lzma/lzma_encoder.h`	5	other
`src/common/tuklib_progname.h`	5	infrastructure
`src/liblzma/lzma/fastpos.h`	5	other
`src/liblzma/lz/lz_encoder.h`	4	other
`src/liblzma/delta/delta_common.h`	4	other
`src/liblzma/common/block_encoder.h`	4	infrastructure
`src/liblzma/rangecoder/range_common.h`	4	other
`src/liblzma/common/stream_flags_common.h`	4	infrastructure

MODULE	FAN-IN	LAYER
`src/common/tuklib_mbstr_nonprint.h`	4	infrastructure
`src/liblzma/common/filter_decoder.h`	4	infrastructure
`src/liblzma/check/crc_common.h`	4	other
`src/liblzma/lzma/lzma2_encoder.h`	4	other
`src/liblzma/common/memcmplen.h`	4	infrastructure
`src/liblzma/common/block_decoder.h`	4	infrastructure
`src/common/tuklib_exit.h`	4	infrastructure
`src/liblzma/common/stream_decoder.h`	4	infrastructure
`src/common/tuklib_mbstr.h`	4	infrastructure

Appendix B: File Reference

Full file paths for truncated references in the report.

REF	FULL PATH
F1	`src/liblzma/lzma/lzma_encoder_optimum_normal.c`