HDF5 Audit Report

Gerd Heber[1] and Elena Pourmal[2]

HDF5 is a foundational component of the scientific software ecosystem, yet its safety, security, and privacy properties warrant analysis across component boundaries. This report presents a targeted, ecosystem-level technical audit of the core library and file format, command-line tools, extension mechanisms (VOL connectors, filter plugins, and virtual file drivers), selected language bindings and applications, the upstream and downstream software supply chain, representative independent format implementations, and long-term data-accessibility concerns.

Three cross-cutting themes recur; this thematic organization is not a risk ranking. First, safety failures require no adversary: storage exhaustion or abrupt termination during writes can leave files incomplete, corrupted, or unreopenable, while reader/writer disagreement and unavailable non-core filters can make data unreadable. These scenarios require tested operational containment, explicit recovery behavior, conformance fixtures, and preservation planning; the wider safety surface remains an evidence gap rather than an assumed Low risk. Second, the selected HDF5 CVE history contains recurring failures to validate file-derived sizes, counts, offsets, and related state before low-level C operations. Patching or containment is necessary treatment for affected scope but is insufficient to eliminate those classes. We therefore propose a prevention program built on an explicit invariant registry for each on-disk structure, a two-phase parser boundary separating decode from object construction, centralized checked arithmetic, structure-aware fuzzing, and file-open security profiles that make hostile input a first-class concern. Third, privacy exposure can arise during normal, authorized operation: the format's self-describing metadata, structural layout, workflow artifacts, and extension mechanisms can disclose sensitive information even when raw

[1]The HDF Group, gheber@hdfgroup.org
[2]Lifeboat, LLC, elena.pourmal@lifeboat.llc

data are protected. Encryption may be required at some disclosure boundaries, but it is one partial control alongside minimization, recipient restriction, and lifecycle governance.

A related source-level observation found candidate deserializer invariants derived from in-file bytes but enforced only inside assertions, which release builds may disable. The report assigns this perimeter a 30-day catalog and debug/release differential-test target; each confirmed adverse path is routed by its consequence without waiting for the longer parser migration.

Throughout, we frame the recurring failure as trust-boundary confusion and give risk-informed mitigations together with a distinct implementation-planning framework for library maintainers, extension authors, downstream projects, and end users. The resulting priority order is bounded and scenario-specific: equivalent exposed services that follow attacker-selected external resources require Immediate containment; confirmed affected High and upper-bound-High scopes require Near-term treatment; Moderate findings receive Planned treatment; and unresolved safety, extension-privacy, and supply-chain observations require evidence rather than an assumed Low rating. Longer-horizon structural work prevents recurrence but does not defer those response clocks. Although the subject is HDF5, the analysis is intended to generalize to comparable open-source scientific data ecosystems.

Acknowledgments: This material is based upon work supported by the U.S. National Science Foundation under Federal Award No. 2534078. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Revision History

Version Number	Date	Comments
v1	July 4, 2026	Initial draft released for internal review.
v2	August 31, 2026	Public release.

Executive Summary

What this report is

This is a targeted, ecosystem-level technical assessment of safety, security, and privacy risk in the HDF5 software and data ecosystem, with an evidence cutoff of August 27, 2026. It is not a certification, attestation, penetration test, or exhaustive source review, and it expresses no assurance opinion. The full engagement type, scope, and limitations are stated in Sections 1.1–1.11.

Ratings apply to specific adverse-event scenarios at specific boundaries, never to HDF5 as a technology. The rating method is defined once, in Section 1.8, and Appendix A is authoritative for every formal rating in this revision.

Appendix G provides plain-language definitions of the HDF5, security, privacy, risk, testing, supply-chain, and interoperability terms and identifier conventions used throughout the report.

Four ecosystem-level judgments

1. **Historically affected core and official-tool paths repeatedly failed to validate file-derived values** — sizes, counts, offsets, message state, layout state, filter and datatype parameters — before using them in trusted C operations. Local fixes can treat a known variant; eliminating the class requires invariant-driven parsing and continuing regression evidence.
2. **Applications can turn documented HDF5 capabilities into attack mechanisms** when they process untrusted files while holding filesystem, plugin-loading, deserialization, or resource authority that the file's originator does not have. The defect is in the authority boundary, not necessarily in the parser.
3. **Reader acceptance, format conformance, and safe deployment are three separate decisions.** Permissive historical acceptance is not proof that a file is consistent, and stricter rejection is not automatically a regression.

4. **Privacy harm can arise during correct, authorized operation**, because protecting the payload does not protect metadata, structure, derived information, caches, logs, or workflow artifacts.

Registered findings at a glance

Table 1 is a navigation aid. The controlling records — scenario statements, scope, evidence, and rating bases — are in Appendix A; the proposed owners come from Appendix C and are role labels, not present assignments. Counts here describe bounded scenarios only. They are not prevalence estimates and must not be aggregated into a score.

Table 1 – Registered findings, ratings, and response routes at a glance

ID	Scenario shorthand	Dim.	Tier	Status	Response route	Owner
APP-SEC-001	Affected dataset service followed attacker-selected external raw-storage paths with worker authority and returned local files.	Sec.	Critical	Rated	Immediate (24 h from identification)	APP
CORE-SEC-001	Malformed on-disk metadata reaches memory-unsafe core-library paths in historically affected releases.	Sec.	High	Provisional	Near-term (30 d)	CORE
APP-SEC-002	Affected Keras legacy-HDF5 model loading executes embedded code despite `safe_mode=True`.	Sec.	High	Rated	Near-term (30 d)	APP
APP-SEC-003	Affected Keras model loading follows attacker-controlled external dataset references and discloses local files.	Sec.	High	Rated	Near-term (30 d)	APP
APP-SEC-005	TensorFlow 2.17.0 loads HDF5 plugins from an uncontrolled search location, enabling local privilege escalation.	Sec.	High	Rated	Near-term (30 d)	APP
CORE-TOOL-001	A malicious file triggers the reported `h5dump` use-after-free in HDF5 1.14.1-2 and earlier.	Sec.	Mod.–High	Provisional	Near-term (30 d)	CORE
CORE-PRIV-001	Sensitive plaintext metadata is disclosed when a file is shared under payload-only protection.	Priv.	Mod.–High	Provisional	Near-term (30 d)	PRIV

Table 1 – Registered findings, ratings, and response routes at a glance, continued

ID	Scenario shorthand	Dim.	Tier	Status	Response route	Owner
CORE-SAFE-001	Storage exhaustion during write or close leaves an incomplete or corrupted file after the operation reports failure; the library defines no recovery or poisoned-file protocol.	Safety	Mod.–High	Provisional	Near-term (containment)	APP/CORE
CORE-SAFE-002	Abrupt process or host termination during write leaves a file corrupted or not reopenable.	Safety	Mod.–High	Provisional	Near-term (containment)	APP/CORE
LONG-SAF-001	An archived dataset becomes unreadable because a required non-core filter is gone. Decadal horizon.	Safety	Moderate	Provisional	Planned (90 d)	INTEROP
CORE-SEC-002	File-controlled work, allocation, traversal, or error paths terminate or exhaust an affected reader.	Sec.	Moderate	Provisional	Planned (90 d)	CORE
APP-SEC-004	Affected Keras weight loading allocates from an extreme file-declared shape and exhausts memory.	Sec.	Moderate	Rated	Planned (90 d)	APP
IND-SAF-001	HDF5 2.0/2.1 exact-width validation rejects a sufficient but nonminimal chunk encoding permitted by the specification.	Safety	Moderate	Rated	Planned (90 d)	INTEROP
IND-SAF-002	Older permissive readers accept contradictory PureHDF 2.1.2 chunk metadata that HDF5 2.0 rejects after upgrade.	Safety	Moderate	Rated	Planned (90 d)	INTEROP
CORE-SEC-003	Candidate deserializer invariants derived from in-file bytes are enforced only inside `assert(...)` and may be absent from `-DNDEBUG` builds. One development-tree instance has a source-reported adverse result; supported- release scope and population remain unverified.	Sec.	*None*	Backlog	Evidence action (RM-1E, 30 d target)	CORE
FMT-SAF-001	The specification and reference library disagree on one open group-info rule; the affected population and adverse consequence are not yet bounded.	Safety	*None*	Backlog	Administrative adjudication and evidence action (90 d target)	INTEROP
EXT-PRIV-001	Extensions can move data and metadata across storage, cache, logging, network, and executable-plugin boundaries.	Priv.	*None*	Backlog	Evidence action (RM-2E)	PRIV
SUP-SAFE-001	Downstream builds and workflows can disagree about ABI, linkage, filters, plugins, VFDs, parallel features, or file locking.	Safety	*None*	Backlog	Evidence action (RM-2F)	REL

Fourteen formal findings are registered: seven Rated and seven Provisional. Their tiers comprise one Critical, four High, four Moderate–High ranges, and five Moderate. Four observations remain in the evidence backlog; backlog means insufficient evidence to rate, not Low risk. `FMT-SAF-001` carries an administrative adjudication but no inherent-risk urgency. The registered write-side and lifecycle findings ensure that non-adversarial data survival is considered alongside hostile-input risk; this is not a prevalence claim.

Where the priority actually falls

Response urgency comes from Table 31 and nowhere else. Chapter order, recommendation numbering, CVE counts, publicity, effort, and dependencies do not create or revise it.

- **The single Immediate route belongs to downstream service operators, not to the reference-library maintainers.** It is conditional: it applies to any service that still automatically follows file-selected external resources while holding access to sensitive local files. The affected July 2026 deployment is reported remediated, and this report assigns no current residual tier to it.
- **For The HDF Group specifically, there is no Immediate action in this revision.** The near-term core obligations are the deployment census (`RM-1A`), the affected-version treatment of `CORE-SEC-001` and `CORE-TOOL-001` (`RM-1B`), and the 30-day assert-site catalog (`RM-1E`).
- **Deployment and storage operators own the write-side containment clock.** Within 30 days of identifying applicable scope, `RM-1F` requires tested capacity monitoring, checkpoint and recovery discipline, and a validated independent copy; `RM-1A` supplies scope evidence but is not itself containment.
- **Two response clocks are distinct.** A finding's clock starts when an affected deployment or equivalent exposure is *identified*. The census and assert-catalog clocks start when the roadmap is *adopted*. Adoption must set

due dates for both inventories, so that a response clock cannot stay latent through unbounded non-discovery.

If only a few things can be funded

Section C.5 identifies a capacity-constrained subset that preserves response clocks. It is a funding aid, not a risk rank or quantitative efficiency claim.

What this report does not establish

No proposed mitigation is shown adopted or effective; every package remains *Proposed / implementation not verified*, with no residual-risk acceptance. Population, fix-coverage, and deployment-control evidence remain incomplete, and findings rely on document, artifact, or architecture review unless independent reproduction is expressly stated. Silence is not evidence of absence.

Contents

Executive Summary **4**

1. Introduction **13**
1.1. Purpose and Engagement Type 13
1.2. Objectives and Intended Audience 13
1.3. Scope 14
1.4. Assessment Dimensions 15
1.5. Assessment Surfaces 16
1.6. Temporal Boundary and Evidence Basis 16
1.7. Methodology 18
1.8. Report-wide Risk Rating and Mitigation Planning 19
1.9. Identifier Scheme and Relationship to SSP SIG Artifacts 25
1.10. Reconciliation with the SSP SIG Risk Registries 25
1.11. Limitations and Assurance 26
1.12. Privacy Context 27

2. HDF5 Core Library and File Format **28**
2.1. HDF5 Safety Observations and Evidence Gaps 28
2.2. Malformed Input as a Security Boundary 32
2.3. HDF5 Privacy: Assets and Exposure 36
2.3.1. Introduction 36
2.3.2. HDF5 Assets and their Exposure 38
2.4. HDF5 Tools 42
2.4.1. Privacy exposure through HDF5 command-line tools . . 42
2.5. Audit Findings and Mitigation Planning 45
2.5.1. Security and Safety 46
2.5.2. Prior Art and the Remaining Work 47
2.5.3. Core Security Control Program 49
2.5.4. Safety: Durability and Recovery 53
2.5.5. Privacy 56

3. HDF5 Library Extensions **63**
3.1. Assessment Boundary 63
3.2. Virtual Object Layer Connectors 63
3.3. Filter Plugins 66
3.4. Virtual File Drivers 67
3.5. Control Proposals and Routing 69

4. HDF5 Wrappers and Language Bindings **72**
4.1. Boundary Model 72
4.2. Python-Facing Cases 72
4.3. Coverage and Mitigation Routing 73

5. Applications using HDF5 **75**
5.1. Boundary Case Study: External Raw Storage and the Hugging Face Incident 76
5.2. Audit Findings and Mitigation Planning 78

6. HDF5 in the Software Supply Chain **81**
6.1. Upstream Dependencies 81
6.2. Downstream Dependencies 82
6.3. Audit Findings and Mitigation Planning 85

7. Independent HDF5 Implementations **88**
7.1. Overview 88
7.2. Two Distinct 2026 Chunk-Layout Cases 90
7.2.1. Sufficient but Nonminimal Encoding 90
7.2.2. Contradictory PureHDF Metadata 90
7.3. When a Normative Ambiguity Remains Unresolved 91
7.4. Audit Findings and Mitigation Planning 94

8. Long-term Data Accessibility and Interpretability **96**
8.1. How Extension Dependencies Affect Preservation 96
8.2. Audit Findings and Mitigation Planning 97

9. Conclusion **99**
9.1. Overall Assessment Conclusion 99
9.2. Reconciled Mitigation Priorities 100
9.3. Significance of the 2026 Developments 101
9.4. Assurance, Closure, and Residual Risk 102

References **104**

A. Findings Register **111**
A.1. Correspondence with the SSP SIG Registries 113
A.2. Evidence and Scoping Backlog 114
A.3. Response Routing 117
A.4. Evidence Actions Without Assigned Urgency 119
A.5. Core Library, Tool, and Format Findings 119
A.6. Application-Boundary Findings 124
A.7. Independent-Implementation and Interoperability Findings . . . 127
A.8. Format and Specification Evidence Record 128
A.9. Long-Term Accessibility Findings 129

B. Recommendation Register **131**
B.1. Correspondence with SSP SIG Policy Controls 135
B.2. Recommendations Not Tied to a Formal Finding 136

C. Phased Mitigation Roadmap **137**
C.1. Accountability Roles 137
C.2. Phase Gates and Clocks 138
C.3. Licensing Constraint on Reuse of Existing Work 139
C.4. Work-Package Register 140
C.5. Minimum Viable Subset Under Constrained Capacity 143
C.6. Acceptance and Closure Evidence 144
C.7. Finding and Evidence-Action Coverage 147
C.8. Change Control and Residual Risk 149

D. CVE History Summary **151**

E. Software Supply Chain Details **163**

F. Resources and Supporting Tooling **173**

G. Glossary **174**
G.1. Abbreviations 174
G.2. Report identifiers and notation 180
G.3. Quick distinctions 182
G.3.1. Risk and response terms 182
G.3.2. Format interpretation terms 183
G.4. Terms and concepts 184
G.4.1. A–C 184
G.4.2. D–H 193
G.4.3. I–P 202
G.4.4. Q–Z 211

1. Introduction

1.1. Purpose and Engagement Type

This report presents a targeted, evidence-based technical assessment of safety, security, and privacy risks in the HDF5 software and data ecosystem. Its purpose is to identify systemic risk patterns, explain the trust boundaries at which those risks arise, and recommend mitigations for the organizations and individuals that develop, distribute, deploy, or use HDF5.

The term *audit* denotes a documented technical examination. This engagement is not a compliance certification, attestation engagement, penetration test, or exhaustive line-by-line source-code review. It expresses no independent assurance opinion and does not certify that any HDF5 release, application, file, or deployment is safe, secure, privacy-preserving, conforming, or free from defects. Authorship and sponsorship are disclosed in the front matter.

1.2. Objectives and Intended Audience

The assessment maps HDF5 assets, capabilities, trust boundaries, and failure modes; distinguishes non-adversarial safety failures, adversarial security failures, and privacy harms during authorized operation; groups evidence by recurring root cause; separates conformance, reader acceptance, and deployment safety; and assigns mitigations to actors able to enforce them. It considers the format, core library and tools, extensions, bindings, applications, supply chain, independent implementations, and data lifecycle.

The primary audience is The HDF Group's maintainers, security responders, release engineers, and technical governance bodies. Additional audiences include extension and plugin authors, binding and independent-implementation maintainers, downstream application and distribution maintainers, operators of data-ingestion services, research-infrastructure teams, data stewards, and security and privacy reviewers.

Recommendations are assigned, where possible, to the layer able to enforce them. A control failure at an application or deployment boundary is not classified as a core-library defect merely because an HDF5 mechanism was involved.

1.3. Scope

The assessment uses three coverage levels:

Primary assessment objects. The HDF5 on-disk format; the core C library and command-line tools; VOL, filter, and VFD extension classes; and HDF5 build, release, and dependency configuration. The report states findings or evidence actions for these objects where the record supports them.

Boundary case studies. Selected incidents and interoperability disputes show how HDF5 capabilities interact with application authority, deployment policy, or independent writers. The referenced projects are not audited as products.

Contextual references. Other components appear only to explain ecosystem reach, dependencies, compatibility, or potential exposure; their inclusion is not an assessment.

Cases were selected purposively for untrusted-input or ambient-authority exposure, recurring failure classes, interoperability or archival consequence, and cross-cutting controls. They are not a statistical sample; "ecosystem-wide" describes breadth of surfaces, not exhaustive coverage or prevalence.

Out of scope. Outside the engagement are:

- HSDS implementation and operations (it appears only in ecosystem context);
- product-wide assessment of HDFView, Keras, TensorFlow, Hugging Face, PureHDF, or other cited downstream or independent projects;

- AI/ML behavior unrelated to an HDF5 integration boundary, and detailed MPI, collective-I/O, concurrency, performance, or parallel-filesystem analysis;
- exhaustive testing of releases, platforms, configurations, extensions, bindings, applications, and packages, or penetration testing and exploit development; and
- organization-specific infrastructure assessment or any legal, regulatory, contractual, research-ethics, or functional-safety certification.

References to these systems are limited to the expressly discussed HDF5 boundary and do not create findings about the product as a whole.

1.4. Assessment Dimensions

Safety. Safety means operational robustness and protection of data integrity, availability, recoverability, interoperability, and long-term accessibility in the absence of a malicious actor. Relevant conditions include malformed, truncated, inconsistent, or partially written files; resource exhaustion; I/O failure; incompatible implementations; and foreseeable misuse. Safety in this report does not mean regulated functional or physical safety.

Security. Security concerns the possibility that an adversary can compromise confidentiality, integrity, or availability, or cause a process to exercise authority contrary to the operator's intent. The principal threat scenario is an HDF5 file, metadata field, external reference, plugin requirement, archive, or model artifact controlled by an untrusted party and processed by an application possessing filesystem, network, compute, plugin-loading, or credential authority.

The assessment includes memory-safety defects, denial of service, resource exhaustion, external-resource resolution, dynamic code loading, downstream deserialization, and supply-chain compromise. A security consequence does not require a defect in the HDF5 format or parser: an intentional HDF5 capability can become

an attack mechanism when a downstream application interprets file-controlled instructions using authority unavailable to the file's originator.

Privacy. Privacy concerns adverse consequences arising from the disclosure, retention, inference, aggregation, correlation, or secondary use of information. Such consequences may arise during authorized and otherwise correct operation, without a security breach. The assessment includes scientific data, descriptive and structural metadata, object names and attributes, provenance, workflow artifacts, caches, temporary copies, external resources, and information that becomes sensitive when datasets are combined.

The report evaluates privacy-exposure mechanisms and possible technical or governance controls. It does not determine compliance with privacy statutes, regulations, contracts, consent requirements, institutional policies, or research-ethics obligations.

1.5. Assessment Surfaces

Figure 1 maps the principal components and interfaces considered when identifying HDF5 risk. Inclusion in the figure provides ecosystem context and does not override the scope levels and limitations defined in Section 1.3. In particular, HSDS is shown for context but remains outside the engagement.

1.6. Temporal Boundary and Evidence Basis

The evidence cutoff for this revision is August 27, 2026. Public evidence available on or before that date is eligible for consideration. A retrieval, archive-capture, or citation-access date after the cutoff may document an artifact or event that existed on or before August 27, provided that the cited record identifies that earlier artifact or event and the assessment does not rely on a later substantive change. A measurement, reproduction, fix, issue update, decision, or other substantive evidence first generated after the cutoff is a subsequent-event update and is outside this revision unless expressly incorporated and labelled as such.

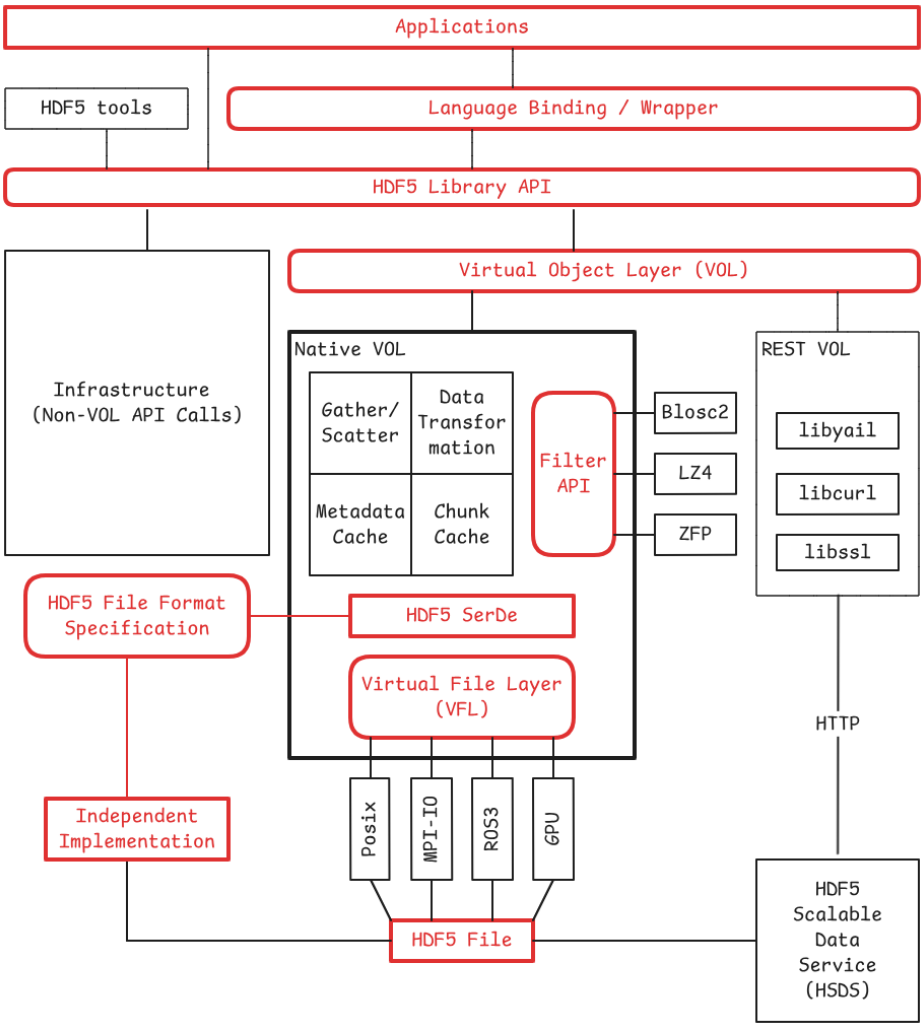

Figure 1 – HDF5 assessment surfaces.

The historical analysis has no fixed lower date and includes earlier releases where necessary to examine cited vulnerabilities, compatibility behavior, and implementation changes. Because this is a historical, ecosystem-level assessment, no single software baseline applies. A conclusion concerning one release, commit, build configuration, platform, or third-party version is not presumed to apply to others. Version-specific conclusions are bounded by the artifact identifiers cited with them. Where an exact artifact is not identified, a statement should be read as an architectural or historical class-level observation, not as a release-specific conclusion.

The principal evidence consists of published HDF5 specifications and API documentation; selected source code and build configurations; security policies, advisories, and CVE records; issue and pull-request discussions; cited commits and regression artifacts; and authoritative documentation or incident reports from representative downstream and independent projects.

1.7. Methodology

The qualitative method combines: (1) surface, asset, and trust-boundary mapping; (2) examination of specifications, source and build artifacts, vulnerability records, fixes, tests, and issue records; (3) grouping by the earliest evidenced invariant or authority-boundary failure; (4) separate analysis of normative conformance, writer behavior, reader validation, and compatibility policy; and (5) scenario rating under Section 1.8, followed by separate implementation planning. Acceptance by one reader is not proof of conformance, and rejection by another is not proof of non-conformance.

For incidents involving multiple components, the analysis distinguishes the HDF5 mechanism, the authority supplied by the host application, the failed control or policy boundary, and the resulting impact. The presence of HDF5 in an exploit chain is not, without additional evidence, classified as a parser or file-format defect.

Evidence is labelled, where practicable, as:

Documented observation: supported by a cited primary artifact, with source-reported evidence distinguished from independent reproduction;

Reasoned inference: a conclusion whose material assumptions and uncertainty are identified; or

Recommendation: a proposed control awaiting implementation and acceptance evidence.

Unless a finding expressly records an independent reproduction or test, it should be understood as the result of document, artifact, or architecture examination rather than independent dynamic verification.

1.8. Report-wide Risk Rating and Mitigation Planning

This subsection defines the only risk-rating method used in this report. A formal rating applies to a specific adverse-event scenario, not to a technology, chapter, CVE count, root-cause family, recommendation, or implementation task. Each rated finding states, directly or by reference, the assessment object and affected version or surface; the initiating actor or non-adversarial event; the condition, exposure, and material preconditions; the assets or people affected; the credible consequence; and, when time materially affects the rating, the assessment horizon. Materially different versions, deployments, preconditions, or consequences are rated separately.

A risk scenario can be summarized as follows:

> Given *[actor or non-adversarial event]*, *[assessment object and scope]*, and *[exposure and preconditions]*, *[condition or event]* may cause *[harm]* to *[assets, people, operations, or other organizations]* *[during the material assessment horizon, if any]*.

Inherent risk. Inherent risk is the risk presented by the stated scenario in its defined assessed population and deployment. It includes only controls that are mandatory and intrinsic to the identified implementation, version, and scoped

default configuration; it gives no credit for optional, compensating, organization-specific, or proposed controls. A feature, configuration, privilege, or user action required to reach the condition is a scenario precondition and informs exposure; it is not an implementation-effort factor. A disabled-by-default or opt-in state constrains exposure only when that state is part of the stated scope. Deployments in which the feature is enabled are rated separately when the difference is material.

Where the effectiveness and coverage of existing controls are supported by evidence, a finding may also state *current residual risk*. A *target residual risk* may be projected for a recommendation, but it is not treated as achieved until the control passes its acceptance criteria. Neither current nor target residual risk replaces the inherent-risk rating. When a residual rating is stated, it uses the same criteria and identifies the controls credited in the assessment.

Inherent risk combines the impact and exposure levels in Table 2 using Table 3. Exposure is the rubric's reachability and recurrence axis: it expresses how readily and commonly the scenario arises under the stated conditions, not a numerical probability. The axes and tiers are ordinal; their labels are not numeric scores and are not multiplied.

Table 2 – Report-wide impact and exposure criteria

Level	Name	Decision rule
I1	Limited impact	Transient or localized effect with routine recovery; no material incorrect result, durable data loss, sensitive disclosure, or harm to individuals.
I2	Material impact	Meaningful workflow interruption; recoverable data loss or corruption; a persistent but bounded compatibility or scientific-validity problem; or limited disclosure, inference, or privacy harm affecting a defined population.
I3	Serious impact	Major data loss, corruption, or outage; process compromise or unauthorized resource access; consequential scientific invalidity; or substantial sensitive- information disclosure or privacy harm.

Table 2 – Report-wide impact and exposure criteria, continued

Level	Name	Decision rule
I4	Severe or systemic impact	Widespread, cross-tenant, or privileged compromise; irreversible loss of unique or critical scientific records; large-scale disclosure of highly sensitive information; or ecosystem- or mission-level disruption.
E1	Exceptional exposure	The scenario requires an unusual configuration and several restrictive preconditions, affects a narrow surface, and is exceptional within the stated assessed population.
E2	Constrained exposure	The scenario is realistic but limited to an opt-in feature, restricted actor capability, uncommon workflow, or multiple material preconditions.
E3	Broad exposure	The scenario arises in a common workflow or deployment, has few material preconditions, commonly crosses a trust boundary, or recurs within the stated assessed population.
E4	Routine exposure	The scenario arises during default or normal operation, is broadly reachable and repeatable, or is straightforward to trigger or automate. Malicious intent is not required for safety or privacy harm.

Table 3 – Report-wide inherent-risk matrix

Impact	E1	E2	E3	E4
I4 Severe/systemic	High	High	Critical	Critical
I3 Serious	Moderate	High	High	Critical
I2 Material	Low	Moderate	Moderate	High
I1 Limited	Low	Low	Moderate	Moderate

The following decision rules apply:

1. Use the greatest credible impact supported by evidence, not an unconstrained theoretical worst case.
2. Judge exposure from the stated reachability, preconditions, deployment prevalence, and recurrence. A confirmed incident establishes feasibility but

does not by itself establish ecosystem-wide exposure. Absence of adequate prevalence or recurrence evidence changes confidence or rating status; it does not establish E1.

3. Rate the scenario at the boundary where authority and control reside. Participation in an exploit chain does not assign the full impact to every component in that chain.
4. Split findings when affected versions, deployment profiles, or consequences would produce materially different ratings.
5. Do not add or average safety, security, and privacy impacts. Record each applicable dimension and identify the substantiated impact that controls the overall rating.
6. Do not translate CVSS scores, CVE counts, publicity, privacy-exposure category codes, or recommendation numbers into this rating.
7. State a time horizon when it materially affects impact or exposure. Do not directly order ratings that use materially different horizons.

Evidence confidence and rating status. Confidence records uncertainty in the evidence and is separate from both exposure and risk:

High confidence. Direct, scope-specific primary evidence supports the condition and material rating assumptions through reproduction or mutually consistent specification, source, fix, and test artifacts.

Moderate confidence. Credible primary evidence supports the scenario, but an important element such as reproduction, prevalence, version boundaries, or causal attribution remains incomplete.

Low confidence. The scenario or rating depends materially on incomplete, disputed, secondary, or uncorroborated evidence.

Confidence is not multiplied into, added to, or subtracted from inherent risk. If uncertainty could change the tier, the report gives the plausible range and the controlling assumptions rather than silently lowering the rating.

A rating record also states one of the following statuses:

Rated. The impact and exposure levels and the resulting inherent-risk tier are supported by the cited evidence.

Provisional. The impact and exposure levels can be estimated, but material uncertainty remains and is expressly stated.

NE—not evaluated. At least one required rating input — impact or exposure — lacks sufficient evidence. An NE finding is not implicitly Low risk.

N/A—not applicable. The item is not a risk-bearing finding, such as a recommendation, implementation task, or program theme.

Narrative findings that do not state impact, exposure, tier, status, and confidence carry no formal report-wide rating.

Risk and implementation planning are separate. The risk record and the recommendation record are separate. The following planning fields never change the inherent-risk rating:

Table 4 – Treatment-planning fields kept separate from inherent risk

Field	Recording rule
Response urgency	Records *Immediate*, *Near-term*, *Planned*, or *Monitor*, with the rationale and target horizon. It is determined from current residual risk, time to harm, active exploitation or recurrence, and stated risk tolerance—not from dependencies or effort.
Implementation dependencies	Lists concrete prerequisite recommendation or decision identifiers. Use *None known* when none are identified and *TBD* when dependency analysis has not been performed; do not encode dependencies as severity.
Effort	Records *Not estimated*, *Small* (localized change), *Medium* (multi-module or coordinated change), or *Large* (architectural, cross-project, compatibility, or migration work). Estimate work independently of elapsed duration or delivery horizon. Effort is not subtracted from risk.

Table 4 – Treatment-planning fields kept separate from inherent risk, continued

Field	Recording rule
Implementation sequence	Records a delivery wave or order together with its concrete prerequisite or enabling rationale. Recommendation identifiers alone do not establish sequence, and delivery order is not a risk rating.
Owner and target horizon	Names the actor with authority to implement or accept the recommendation and the intended completion date or release.
Acceptance evidence	States the observable test, artifact, or operational evidence required to mark the recommendation complete and reassess residual risk.
Residual-risk decision	Records the verified remaining rating and the accountable decision to accept, further reduce, avoid, or transfer it.

Unless a stricter stated risk tolerance applies, the default response-urgency rule maps Critical current residual risk to *Immediate*, High to *Near-term*, Moderate to *Planned*, and Low to *Monitor*. A finding whose rating is expressed as a tier range is routed at the upper bound of that range; the report refers to such a case as *upper-bound-High* when the range is Moderate–High. Routing at the upper bound does not collapse the range, which remains the stated rating. Active exploitation or recurrence, imminent harm, or a credible irreversible consequence can raise urgency. When current residual risk has not been assessed, inherent risk is the conservative provisional baseline. Dependencies and effort may shape the delivery plan or require an interim containment action, but they never lower response urgency. Any departure from this mapping requires a recorded rationale and accountable decision.

A Critical scenario remains Critical even when its mitigation is high effort. An enabling action may appear earlier because it is a prerequisite; the action does not thereby acquire a risk rating. Immediate containment and a later architectural treatment may therefore address the same finding on different delivery horizons.

Every formal finding record should contain a stable identifier, scenario, dimension, affected scope, evidence, I/E rationale, an inherent-risk tier when evaluated or explicit no-tier/NE notation, rating status, confidence, and linked recommendation identifiers. Every recommendation record should separately identify the findings

addressed, enforcing actor or owner, urgency, dependencies, effort, sequence and target horizon, acceptance evidence, status, and residual-risk decision. Numbered recommendation identifiers in later sections identify items only; they do not establish risk, urgency, effort, dependency, or delivery order.

The populated report-wide register is Appendix A. That appendix is authoritative for formal ratings in this revision: chapter narrative, recommendation numbering, CVE metadata, and descriptive uses of words such as "high" or "important" do not create an additional rating unless they map to a registered finding. Appendix C separately groups the recommendations into proposed work packages with accountable roles, dependencies, effort, phase gates, acceptance evidence, and residual-risk decision requirements. Those planning fields do not revise the register's inherent ratings or represent stakeholder-approved commitments until adopted and assigned.

1.9. Identifier Scheme and Relationship to SSP SIG Artifacts

This report and the HDF5 Safety, Security & Privacy (SSP) SIG repositories are separate programme artifacts with separate namespaces. Report-local findings use component/dimension forms such as `CORE-SEC-nnn`; recommendations use forms such as `CORE-Sn`; and roadmap packages use `RM-nX`. Outside this document they should be prefixed with `SHINES-2026-001:`.

SSP SIG policy controls use `HDF5-SHINES-FAMILY-nn`; its threat, hazard, and exposure registries use `ATK-nnn`, `HAZ-nnn`, and `EXP-nnn` [48, 51, 50, 49]. Those records are not findings or recommendations of this report. The OpenSSF coverage matrix scores SSP controls, not this report's findings [52]. Appendices A and B contain editorial crosswalks; a mapping claims neither adoption, control satisfaction, nor agreement between ratings.

1.10. Reconciliation with the SSP SIG Risk Registries

The two rating methods are incompatible. SSP SIG multiplies 1–5 severity by 1–5 likelihood and bands the resulting score; this report resolves ordinal I1–

I4 impact and E1–E4 exposure through Table 3. Scores and tiers cannot be converted or merged: SSP entries rate ecosystem mechanisms, whereas this report rates bounded adverse-event scenarios and withholds a tier when population or consequence evidence is insufficient. Table 5 records the material reconciliations; the full record-level crosswalk remains in Appendix A.

Table 5 – Material reconciliations with SSP SIG ratings

SSP record	This report	Reconciliation
ATK-001, ATK-009	CORE-SEC-001 High, Provisional; CORE-SEC-002 Moderate	The registry rates recurrence or service denial at mechanism level. This report bounds affected version/path prevalence and the evidenced recoverable-workflow impact; a deployment-specific mission outage would be a separate scenario.
ATK-011, ATK-013	APP-SEC-005 High; SUP-SAFE-001 backlog	Plugin execution is rated where the evidenced search-path authority resides. The supply-chain record concerns feature mismatch, not the registry's distinct trojanized-package scenario.
HAZ-005	CORE-SAFE-001 Moderate–High, Provisional, Low confidence	The scenario is bounded to finite storage filling during active write without a known-good recovery control. RM-2D measures failure sequences and recoverability; it does not gate the provisional rating.
EXP-001, EXP-002, EXP-004–EXP-007	CORE-PRIV-001 Moderate–High; EXT-PRIV-001 backlog	This report rates a bounded plaintext-metadata disclosure, not the absence of whole-file encryption. Extension mechanisms remain unrated until a deployment, sensitive population, recipient, and harm are established.

Where this report is silent and a registry entry exists, the registry entry stands as the SSP SIG's own assessment. Silence here is not disagreement, and Section 1.11 applies.

1.11. Limitations and Assurance

This assessment is time-bounded and based predominantly on public evidence. It is not an exhaustive review of the HDF5 source tree, every file-format feature, every configuration, or every ecosystem component. Representative sampling supports conclusions about observed mechanisms and recurring classes; it does not establish their frequency throughout the ecosystem.

Public CVE and issue records may be incomplete, duplicated, disputed, or subsequently corrected. Deployment-specific exposure and impact depend on application privileges, isolation, resource limits, enabled features, data sensitivity, and

provenance. Proposed mitigations are evaluated analytically unless an implementation and verification procedure is expressly cited.

This report provides no formal assurance or certification opinion. Its conclusions are bounded by the assessment objects, methods, evidence, and cutoff described above. The absence of a reported finding does not establish the absence of defects, vulnerabilities, safety failures, or privacy risk.

Inherent-risk ratings express the authors' evidence-based judgment under Section 1.8; an unrated finding is not assumed to be Low risk. Numbered recommendation identifiers do not express risk, urgency, effort, dependency, or sequence. Recommendations become auditable commitments only when paired with an owner, affected scope, target horizon, implementation status, acceptance criteria, and residual-risk decision.

1.12. Privacy Context

NIST treats privacy as broader than confidentiality: inference, aggregation, secondary use, context change, and surveillance can create harm even when a system is secure and functioning as intended [73, 23]. HDF5's self-describing metadata, object structure, extension architecture, provenance, caches, and workflow artifacts can therefore reveal information beyond the stored payload.

Access control, anonymization, and aggregation do not by themselves prevent re-identification or inference from metadata and correlations. The risk can change as files move between organizations and repositories, are repurposed, or are combined with external data and machine-learning analysis. This report therefore examines privacy exposure throughout the HDF5 data lifecycle; it does not claim that the format or library supplies a general privacy guarantee.

2. HDF5 Core Library and File Format

2.1. HDF5 Safety Observations and Evidence Gaps

Safety in this chapter means robustness and data integrity in the absence of a malicious actor, as defined in Section 1.4. This revision examines four condition families and registers five formal findings: two write-side findings, one filter-dependent archival finding, and two reader/writer compatibility findings. A separate format-governance observation remains in the evidence backlog because its adverse consequence and exposure are not yet bounded.

The first three condition families are write-side and lifecycle risks and were the least developed part of an earlier revision of this report; the discussion below states each as a distinct trigger, because conflating them — as an earlier draft did by merging storage exhaustion and abrupt termination into one observation — obscures that they need different containment and different structural treatment.

Storage exhaustion during write or close.
Storage fills during a write, flush, or close, and the operation reports failure through the ordinary HDF5 error path. The library offers no structured VFD-level notification at the failure point, defined recovery procedure, or supported way to mark the file suspect and abandon it safely. The controlling evidence is a maintainer statement that there is no HDF5 recovery story for the condition and no VFD callback for it, together with four candidate treatments and a documented cache workaround used in its place [20]. Registered as `CORE-SAFE-001`, **Moderate–High**. An earlier revision recorded this as NE for want of a bounded recovery path; the recovery path is in absent at the cited architectural boundary. That establishes a control gap; it does not establish how often storage exhaustion occurs or what durable state follows it.

Corruption from abrupt termination.
A writer is killed, crashes, or loses power with metadata updates in flight, and in-memory and durable state diverge with no journal or write-ahead log to reconcile them. Community reports recur across sixteen years, and The HDF Group's own 2024 discussion states that a crash "can lead to data loss

or corruption of the file" and describes restarting crash-proofing work [18, 16, 17, 11, 12, 8, 15, 9]. Registered as `CORE-SAFE-002`, **Moderate–High**. This condition had no record of any kind in an earlier revision.

Extension-dependent loss of interpretability.
An archived dataset requires a non-core filter that is later unavailable. Registered as `LONG-SAF-001`, **Moderate** at a stated ten-year-or-more horizon, and developed in Section 8. Because the horizon is material, decision rule 7 forbids ordering this rating directly against ratings that state no horizon. VFD and VOL archival dependencies remain a reasoned extension of the inquiry, not part of this rated scenario.

Reader/writer disagreement.
The two 2026 chunk-layout cases in Section 7, registered as `IND-SAF-001` and `IND-SAF-002`. They share an unreadability outcome with `LONG-SAF-001`, but not its mechanism: these are specification and metadata-conformance failures, whereas the archival finding concerns an unavailable runtime dependency.

The shape of the gap this closes. It is worth naming why safety was the thinnest dimension. The rest of this report is organised around *read-side hostile input* — the rubric's principal threat scenario, the CVE analysis, the invariant registry, the parser boundary, the fuzzing programme, and the security profiles all answer the question "a malicious file has arrived." The three findings above answer a different question, "my file did not survive," and no structural recommendation in the security track addresses it. That asymmetry, rather than any judgment that the write path is sound, is why these conditions were previously unregistered or unrated.

Coverage limitation, stated against an existing hazard analysis. Safety is the least developed dimension in this revision. The extent of the gap can be stated precisely rather than vaguely, because the SSP SIG has already published a sixteen-entry safety hazard registry built on a control-system hazard model with unsafe control actions, derived safety constraints, and per-entry ratings [50, 45].

Table 6 maps that registry to this report. **Of sixteen hazards, this revision registers a corresponding formal finding for four, an unrated evidence record**

for two, and nothing at all for ten. An earlier revision covered one. That is the honest measure of this chapter's safety coverage. The registry entries are not reproduced or re-rated here — their method is incompatible with Section 1.8, as Section 1.10 explains — and where this report is silent, the registry entry stands as the SSP SIG's own assessment. Silence here is not disagreement and is emphatically not a Low-risk conclusion.

Table 6 – SSP SIG safety hazard registry mapped to this report's coverage

ID	Hazard	Their rating	Coverage in this report
HAZ-001	Crash during metadata update	20, Very High	**Covered.** CORE-SAFE-002, Moderate–High, Near-term containment with structural treatment under CORE-SAFE-S2. Registered in this revision; absent from the previous one.
HAZ-002	Concurrent access without safe coordination	15, High	**Not assessed.** No concurrency scenario was developed.
HAZ-003	Extension boundary violates core invariants	15, High	**Partially covered as an unrated record.** The privacy aspect is EXT-PRIV-001; the *safety* aspect — an extension breaking an invariant the core relies on — has no counterpart here.
HAZ-004	Metadata and artifacts leak sensitive information	12, High	**Covered.** CORE-PRIV-001, Moderate–High, Near-term. This is the one hazard with a corresponding rated finding.
HAZ-005	Out-of-space during write or close	15, High	**Covered.** CORE-SAFE-001, Moderate–High, Near-term containment with structural treatment under CORE-SAFE-S1. The registry's accident chain and safety constraints SC-1 to SC-3 are adopted as the RM-2D hypotheses.
HAZ-006	False sense of durability from flush	12, High	**Not separately assessed**, but within the evidence base for CORE-SAFE-002: the community record includes reports that flushing itself can leave a file corrupted. A distinct durability-signaling finding is still warranted.
HAZ-007	SWMR deployment misfit	12, High	**Not assessed.**

Table 6 – SSP SIG safety hazard registry mapped to this report's coverage, continued

ID	Hazard	Their rating	Coverage in this report
HAZ-008	File-locking pitfalls or disabling via environment variable	12, High	**Not assessed.** File locking appears only as one item in the SUP-SAFE-001 mismatch list, which is itself unrated.
HAZ-009	Thread-safe build uses a global lock	12, High	**Not assessed.**
HAZ-010	Chunk size too small	12, High	**Not assessed as a safety hazard.** Pathological chunk shapes appear here only as an adversarial resource-consumption mechanism under CORE-SEC-002; the non-adversarial operational hazard is absent.
HAZ-011	Deleting data does not shrink files by default	12, High	**Not assessed.** Note the privacy adjacency: the registry separately rates data remanence after deletion as EXP-003.
HAZ-012	Recovery tooling limits	12, High	**Not separately assessed.** Recovery capability is a controlling assumption of the CORE-SAFE-002 impact range — whether the file reopens after h5clear is what separates I2 from I3 — so the limits of that tooling bear directly on the rating. The evidence-gated repair planner in H5Lens is relevant prior art (Section 2.5.2).
HAZ-013	High-level API thread-safety gaps	9, Moderate	**Not assessed.**
HAZ-014	Portability hazard from missing filter plugins	9, Moderate	**Covered.** LONG-SAF-001 rates the missing-filter archival scenario at a stated decadal horizon. VFD and VOL archival dependencies are identified as unmeasured extensions of the inquiry, not as part of the rating.
HAZ-015	Misuse of metadata-cache flush controls	8, Moderate	**Not assessed.**
HAZ-016	Chunk cache mis-sizing	6, Moderate	**Not assessed.**

Consequence for the reader and for the roadmap. Three things follow. First, a reader must not treat this report's small number of safety findings as evidence that

HDF5's safety surface is small; it is evidence that this engagement examined it least. Second, the natural next revision of this report should adopt the hazard registry as its safety enumeration and work through it scenario by scenario, rather than re-deriving a list. Third, `RM-2D` is currently scoped to storage exhaustion alone; the registry shows at least `HAZ-001`, `HAZ-006`, `HAZ-012`, and `HAZ-015` share its evidence apparatus — fault injection at write, flush, and close boundaries, followed by durable-state and recovery inspection — and it has accordingly been widened to cover the whole cluster, including the abrupt-termination trigger behind `CORE-SAFE-002`. Ten hazards remain with no record here; that number, not the four now covered, is the honest measure of what a safety-focused revision would still have to do.

2.2. Malformed Input as a Security Boundary

Three mechanisms recur across the security evidence for the core library and official tools: arithmetic on file-derived quantities that is not checked for overflow or zero denominators; file-format parsing that constructs internal objects before validating the bytes that describe them; and executable extension code reached through plugin loading, which is treated in Section 3. The first two are examined here.

A CVE history summary is included in Appendix D. The list mixes real HDF5-library issues, removed-tool issues, duplicate/rejected/not-HDF5 entries, and entries with no commit URL. A recurring class represented in the selected history is failure to validate file-derived structure sizes, offsets, counts, message lengths, continuation chunks, datatype metadata, or filter parameters before using them in low-level C memory operations. The 2025 entries include heap overflows, use-after-free, null dereference, memory leak, and resource-consumption defects across object headers, file-space metadata, type conversion, filters, and metadata cache code. This qualitative clustering does not estimate ecosystem frequency.

The fix commentary confirms that several CVEs were symptoms of corrupted metadata not being rejected early. (Examples: the file-space page-size fix added a sanity check immediately after reading `page_size` from the file; the object-

header-continuation fix rejects zero continuation chunk sizes; the self-referential continuation-message fix checks expected versus actual deserialized object-header chunks; and the cache-image fix added buffer-size arguments, overflow checks, input validation, and cleanup.)

An August 2026 source-review observation exposes a related but distinct control gap. HDF5 Pickles issue #87 records candidate deserializer invariants whose truth depends on in-file bytes but whose checks are masked inside `assert(...)` expressions; those expressions provide no runtime validation when assertions are disabled with `-DNDEBUG`. The open catalog task names version-2 B-trees, dataset chunk records, extensible arrays, fractal heaps, free-space managers, shared object-header messages, and metadata-cache images, with a cursory estimate of 50–100 instances [57]. A fixed August 25 develop snapshot corroborates the mechanism: its build configuration notes that CMake adds `NDEBUG` for Release builds, while representative source sites assert a decoded filtered- chunk address and byte count before returning success, assert extensible-array parameters before geometry and allocation calculations, and assert a reverse-filter output size before copying the expected fractal-heap block size [86, 81, 82, 83]. This static sample confirms a source-level assurance gap, but the 50–100 estimate is not a verified site count, and the available evidence does not establish that every candidate is a sole guard, release-reachable in a supported version, or capable of a particular adverse outcome. This report therefore tracks the condition as evidence record `CORE-SEC-003`, not as a separately rated formal finding.

One member of that population has been carried further, and it is worth stating precisely because it illustrates both the mechanism and the remaining evidence boundary. A version-2 B-tree header declaring a zero record size reaches an integer division while the leaf record capacity is derived; the sole guard is an assertion in `H5B2hdr.c`, absent under `-DNDEBUG`. The cited case record reports `SIGFPE` from a locally built `h5dump-H` and the public `H5Fopen`/`H5Gopen2`/`H5Gget_info` path using a release-configured HDF5 2.3.0 development source tree [33]. It expressly records no 32-bit measurement and does not establish an affected supported release or version range. The case therefore corroborates an adverse `-DNDEBUG` path but does not satisfy this report's supported-version condition for promotion into a formal finding. It remains evidence under `CORE-SEC-003`.

A second candidate, the cache-image LRU rank relation, is reported without a measured release-build consequence [31]. This bounded reproduction is a reason to complete the supported-release catalog, not a substitute for it. Regardless of the final count, a predicate whose validity depends on file bytes must have an always-active runtime check; an assertion may duplicate that check for diagnostics but must not enforce it alone.

The observed root-cause categories are more compact than a CVE-by-CVE list and are summarized in Table 7. They were derived independently in this assessment, and they are consistent with the SSP SIG security threat model, which decomposes the same evidence into thirteen mechanism entries [47, 51]; that decomposition is the finer-grained one and is the better basis for the invariant tranches proposed in `CORE-S1`.

Table 8 maps each category to the standard weakness vocabulary, using the SSP SIG's top-CWE list for HDF5 contexts [53]. The mapping is worth stating explicitly because it makes the report's central security claim checkable against an external taxonomy rather than only against its own categories: nearly every entry resolves to `CWE-20` and its children as the *cause*, with the memory-safety CWEs appearing as *consequences*. Per-CVE tags are in Appendix D.

Table 7 – Observed root-cause categories in the selected HDF5 CVE history

Root-cause category	What it means for HDF5
unchecked file-derived sizes / counts / offsets	A recurring evidenced class. HDF5 metadata controls object layout, heap blocks, chunks, dataspaces, filters, links, and type descriptors. When those fields are malformed, C code later trusts them.
inconsistent internal bookkeeping	Attribute tables, cache images, continuation chunks, and free lists often needed separate "allocated capacity" versus "actual count" tracking.
decode/encode asymmetry	Several issues happen because decode did not reject bad messages, then encode/flush later assumed valid native structures.
arithmetic precondition failures	Divide-by-zero and multiplication-overflow checks were missing in chunking, layout, b-tree, filter, and dataspace paths.
unsafe cleanup/lifetime paths	Use-after-free, double-free, leaks, and null dereferences show that malformed input often breaks unwinding logic as much as the primary decode logic.

Table 7 – Observed root-cause categories in the selected HDF5 CVE history, continued

Root-cause category	What it means for HDF5
unbounded traversal, recursion, loop-progress, and resource-limit failures	File-controlled graphs, dimensions, and metadata can drive recursion, repeated work, or allocation without a structural progress rule or explicit budget.
legacy tool attack surface	`gif2h5`, HDF to GIF conversion, `h5dump`, and `h5repack` appear repeatedly. Removal of `gif2h5` was a structural mitigation, not just a patch.
triage quality, duplicate/variant clustering, reachability, and fix-completeness metadata	Several entries have `commit_url:null`, "not filed against HDF5," "cannot reproduce," "not applicable," or tentative confidence. Those should not be counted as reviewed HDF5 fixes without qualification.

Table 8 – Observed root-cause categories mapped to standard weaknesses

Root-cause category	Cause (CWE)	Consequence (CWE) and note
unchecked file-derived sizes, counts, offsets	`CWE-20`, `CWE-1284`, `CWE-1285`	`CWE-787`, `CWE-125`. The single largest group. A quantity or an offset taken from the file is used before it is validated.
inconsistent internal bookkeeping	`CWE-20`, `CWE-1284`	`CWE-787`, `CWE-908`. Allocated capacity and actual count are conflated, so a validated bound does not describe the buffer actually held.
decode/encode asymmetry	`CWE-20`, `CWE-1286`	`CWE-787`. Decode accepts a structure that encode or flush later assumes is well-formed; the syntactic check that was never made surfaces on the write path.
arithmetic precondition failures	`CWE-20`, `CWE-1284`	`CWE-369`, `CWE-190`. Divisors and multiplications derived from file bytes without a zero or overflow precondition.
unsafe cleanup and lifetime paths	`CWE-20`	`CWE-416`, `CWE-415`, `CWE-476`, `CWE-401`. Distinctive in that the unwinding path, not the decode path, is what malformed input breaks.
unbounded traversal, recursion, and resource limits	`CWE-20`, `CWE-1284`	`CWE-400`, `CWE-674`. File-controlled graphs and counts drive work with no structural progress rule or explicit budget.

Table 8 – Observed root-cause categories mapped to standard weaknesses, continued

Root-cause category	Cause (CWE)	Consequence (CWE) and note
legacy tool attack surface	CWE-20	CWE-787, CWE-125. Not a distinct weakness class; a distinct *exposure* class, since these are the programs users run first on untrusted files.
type-confusion on message and datatype flags	CWE-20, CWE-1287	CWE-787. A message or datatype is cast to a structure its declared type does not support — validation of a specified *type*, distinct from validating a quantity or an index.
triage and fix-completeness metadata	*n/a*	*n/a*. A records-quality category, not a weakness. It affects what can be concluded from the history, not what the library does.

Read together, the two tables state the report's core security claim in standard terms: the memory-safety weaknesses that dominate the CVE history are consequences, and the cause they share is CWE-20. That is why CORE-S1 through CORE-S3 target validation and checked arithmetic rather than the individual crash classes, and why closing a CWE-787 instance does not close the class that produced it.

2.3. HDF5 Privacy: Assets and Exposure

2.3.1. Introduction

HDF5 design features create predictable privacy-exposure mechanisms, but realized harm, exposure, and any lifecycle trend are deployment-specific. The file format and library are one source of exposure surfaces; scientific workflows, software ecosystems, interoperability mechanisms, and changing data contexts can add others.

Before describing the assets that require protection in HDF5 and the mechanisms through which privacy exposure may occur, it is important to emphasize several fundamental characteristics of HDF5 and the audit approach adopted in this document.

- *HDF5 privacy primarily concerns accidental or unintended exposure of sensitive information rather than malicious attacks*. Malicious attacks and unauthorized access are primarily addressed through cybersecurity mechanisms and are outside the scope of this section.
- *HDF5, by design, does not provide privacy protections*. When privacy is a concern, users and organizations are responsible for protecting data throughout the entire data lifecycle. In this document, the term data refers not only to individual HDF5 objects within a single file, but also to collections of HDF5 files and external data accessible through the HDF5 library, connectors, and related software components.
- *HDF5 neither implements nor enforces privacy standards*. Furthermore, even when privacy standards and technical safeguards are in place, sensitive information may still be exposed through legitimate data processing, contextual interpretation, pattern discovery, inference, aggregation, and other analytical activities.
- *HDF5 does not manage authentication, authorization, or filesystem protection*. However, privacy risks extend beyond traditional security concerns. Sensitive information may be exposed during legitimate data use by authorized users, even within highly secure computing environments.
- *Additional lifecycle stages can add privacy-exposure surfaces*. Sensitive information stored in HDF5 may be exposed during data creation, processing, storage, sharing, transfer, reuse, or archival; classification, minimization, access restrictions, and lifecycle controls can increase or reduce the realized deployment risk.
- *HDF5 extensibility mechanisms and associated software ecosystems may alter the privacy characteristics of data*. Features such as filters, Virtual Object Layer (VOL) connectors, Virtual File Drivers (VFDs), and external applications, including conversion and analysis tools, may introduce new privacy exposure surfaces or modify existing ones.

The remainder of this section focuses on the assets that require protection in HDF5, the mechanisms through which sensitive information may be exposed, and potential approaches for mitigating privacy risks.

2.3.2. HDF5 Assets and their Exposure

This subsection identifies five classes of asset that may require protection in an HDF5 deployment: the scientific data themselves, user-defined metadata, structural metadata, information derived or reconstructed from combinations of the former, and artifacts produced by the surrounding tools and workflows. Each is treated in turn below. The classes are cumulative rather than alternative — a single sharing boundary can disclose through all five — and the enumeration in `CORE-PRIV-S1` expects a disposition for every class, including a positive statement where a class was examined and found empty.

Data Assets

The primary assets requiring protection are the scientific data themselves. In HDF5, these include data stored in HDF5 objects, such as files, groups, datasets, datatypes, and dataspaces, which represent experimental observations, measurements, simulations, images, sensor outputs, and computational results.

Scientific data may be exposed through direct access using HDF5-based software, including command-line utilities, graphical viewers, and programming interfaces such as Python. However, privacy-sensitive information may also be inferred through examination of the low-level storage organization and auxiliary metadata embedded within the file to support portability, interoperability, and efficient data access. For example, string attributes visible through standard operating system utilities (e.g., the Unix strings command) may disclose dataset names, experimental identifiers, or other descriptive information. Similarly, inspection of dataset headers may reveal storage layout, chunk organization, compression methods, or data acquisition patterns (e.g., frame-based acquisition stored as chunked datasets). Examination of embedded strings may also disclose the use of external storage or references to datasets located in other HDF5 files.

It is important to distinguish between protecting access to data values and protecting information about data representation. Even when raw data are protected through mechanisms such as compression or encryption, information describing their storage organization, layout, or access methods may remain accessible and reveal sensitive contextual information about the underlying data or the processes that generated them.

Metadata Assets

User-defined metadata often constitute one of the most privacy-sensitive asset classes in HDF5. The format was intentionally designed as a self-describing data model that allows extensive contextual information to be stored alongside scientific data. While HDF5 attributes are the most recognizable form of user-defined metadata, they represent only one category of descriptive information maintained within an HDF5 file.

User-defined metadata include all user-created HDF5 objects and object properties that organize, identify, or describe stored data. These include groups, datasets, their associated datatypes and dataspaces, committed datatypes, and attributes attached to HDF5 objects. Each HDF5 attribute is itself a structured metadata object with a name, datatype, and dataspace. Additional metadata assets include file and object names (link names), committed datatype names, compound datatype member names, enumeration labels, object comments, and information stored in the user block. Collectively, these metadata describe the logical organization, semantics, and relationships of the stored data and frequently contain experiment-specific, operational, or organizational information.

Like scientific data, metadata assets may be exposed not only through HDF5 applications but also through direct examination of the binary file format. Inspection of file structures or extraction of embedded strings without using HDF5 software may reveal object names, descriptive attributes, storage relationships, user block contents, and other contextual information that could disclose sensitive information about the data, their origin, or the organization that produced them.

Structural Metadata Assets

Privacy-sensitive information may also be inferred from the structural characteristics of HDF5 files. Structural metadata assets include user-defined organizational structures, such as the hierarchical arrangement of objects, dataset storage layouts (e.g., contiguous, chunked, compact, or external storage), filter pipelines, and relationships established through links, object references, region references, or virtual datasets (VDS). In addition, the HDF5 library maintains internal structural metadata that describe and manage these user-defined structures, including chunk indexing schemes, B-tree organizations, and other implementation-specific data structures.

Even when raw data are inaccessible, for example, because they are encrypted, compressed, stored externally, or protected by file-system access controls, structural metadata may disclose information about the purpose of the data, experimental design, workflow organization, relationships among datasets, or assumptions made by the generating application. Naming hierarchies, grouping strategies, storage layouts, and reference relationships may reveal semantic information about scientific activities or operational processes. Likewise, implementation-specific structures, such as chunk indexing formats or other internal metadata, may disclose characteristics of the HDF5 library version or software implementation used to create or process the file.

Like scientific data and user-defined metadata, structural metadata may be exposed through HDF5 applications or by direct examination of the binary file format without using HDF5 software.

Derived and Reconstructed Information

Privacy-sensitive information in HDF5 may also arise through inference or reconstruction from combinations of available data and metadata. Derived assets include relationships inferred from multiple HDF5 datasets, external links, object references, region references, virtual dataset (VDS) mappings, or information obtained by combining HDF5 data with external sources. These relationships may reveal information that is not explicitly represented within any single dataset or file.

For example, correlating quality-control datasets with experimental observations, combining temporal and spatial measurements, or reconstructing relationships across linked HDF5 files may disclose sensitive characteristics that are not apparent when individual datasets are examined in isolation. Consequently, privacy-sensitive assets extend beyond explicitly stored data and metadata to include information that can be inferred through aggregation, correlation, reconstruction, or other forms of analysis.

Workflow and Applications Artifacts

Privacy-sensitive assets in HDF5 extend beyond data and metadata stored within HDF5 files to include artifacts created by applications, processing workflows, and the surrounding software ecosystem. These artifacts include command outputs (e.g., h5dump output), diagnostic logs, temporary files, metadata snapshots, cached intermediate datasets, conversion artifacts, and outputs generated by plugins or connectors (e.g., binary files produced by an HDF5 VOL connector for another scientific data format).

Examples include VFD SWMR metadata snapshots, cached intermediate results, diagnostic logs, and metadata generated during translation between non-self-describing data formats and HDF5. Such artifacts may expose scientific data, metadata, storage organization, workflow decisions, or implementation details beyond what is contained in the original HDF5 file. Because they are often created automatically, these artifacts may persist beyond their intended lifetime and remain unnoticed by users focused primarily on the scientific datasets themselves.

Scientific data processing workflows routinely enrich, restructure, annotate, and transform datasets to improve interoperability, reproducibility, usability, or analytical value. However, these processes may also introduce or preserve contextual information that was not intended for disclosure. Automatically generated metadata, intermediate processing products, and derived datasets may reveal experimental assumptions, workflow decisions, data provenance, software implementation details, or scientific semantics that were absent from the original data. Consequently, privacy exposure may arise not from malicious activity or unauthorized access, but as an unintended consequence of routine scientific data processing.

2.4. HDF5 Tools

Not strictly part of the core library, but the tools are an important part of the ecosystem and have their own set of issues.

2.4.1. Privacy exposure through HDF5 command-line tools

HDF5 command-line tools provide convenient mechanisms for managing HDF5 files. They can be used to display or extract raw data, and user-defined and structural HDF5 metadata (e.g., `h5dump`, `h5ls`, `h5debug`), edit raw data and attributes with `h5edit`, extract or aggregate data using `h5copy`, change characteristics of HDF5 objects, for example, remove or add data filters and change storage properties using `h5repack`. HDF5 command-line tools are routinely employed throughout the HDF5 data lifecycle. While the command-line tools are essential for working with HDF5 data, they are ultimate sources of unintended privacy exposure.

The HDF5 tools were designed to ease access to HDF5 data, to reveal information about the contents and organization of HDF5 files, and to manage that information while avoiding steep learning curve of HDF5 programming. I.e., HDF5 command-line tools were designed to expose HDF5 assets. Depending on the tool and the selected options, they may display object names, attributes, datatype definitions, storage layouts, chunking parameters, filter pipelines, external storage locations, object references, virtual dataset mappings, and portions or all of the stored data to name a few. Even when users intend only to verify file integrity or inspect metadata, command output may disclose information that is considered sensitive in a particular context.

Privacy exposure is not limited to information displayed on the terminal by `h5dump` or `h5ls`. Command output is frequently redirected to files, incorporated into automated workflow logs, attached to bug reports, archived as diagnostic information, or shared. The output can be used by other tools (e.g., `h5import`) or applications to generate new data. Consequently, metadata and scientific data extracted by HDF5 tools may persist outside the original HDF5 file and

become accessible to individuals who were never intended to access the underlying information.

Several command-line tools may also create new HDF5 files or transform existing ones. `h5repack` and `h5copy` may generate additional copies of data and/or aggregate data, modify storage layouts, remove or add data filtering; `h5edit` may modify and introduce new metadata, change raw data values and preserve information that was not expected to be retained. I.e., the tools can alter the privacy characteristics of the resulting files, particularly when they are incorporated into larger workflows, backups, or public data repositories.

Because HDF5 command-line tools are widely used in automated scripts, continuous integration systems, and scientific workflows, their privacy implications should be evaluated alongside those of the HDF5 library and HDF5-based applications. The privacy mitigation strategies discussed in the next section apply not only to the HDF5 library itself but also, and often to an even greater extent, to HDF5 command-line tools. This is because these tools are designed to expose HDF5 assets and to generate persistent outputs (e.g., new binary and text files).

The table below summarizes the assets exposed by the listed command-line tools.

Table 9 – Privacy exposure through HDF5 command-line tools

Tool	Exposed assets	Workflow artifacts	Privacy considerations
`h5dump`	Raw data; user-defined and structural metadata	Text dumps, redirected output, logs	Displays datasets, attributes, datatypes, storage layout, filter information, object references, and VDS mappings. May expose data stored in other HDF5 files through external links and Virtual Datasets (VDS). Output is frequently redirected to files or logs, creating persistent copies of sensitive information.
`h5ls`	User-defined and structural metadata (optionally raw data)	Terminal output, logs	Reveals object hierarchy, object names, datatypes, storage properties, links, and object relationships. With the `-d` option, also displays dataset values.

Table 9 – Privacy exposure through HDF5 command-line tools, continued

Tool	Exposed assets	Workflow artifacts	Privacy considerations
`h5repack`	Raw data; user-defined and structural metadata	New HDF5 file	Creates additional HDF5 copies while potentially modifying storage layout, filters, and metadata. The `–merge` option can aggregate datasets reachable through external links into a single file. May access non-HDF5 data through configured VFDs and VOL connectors.
`h5diff`	Raw data; user-defined metadata	Comparison reports	Reads and compares datasets and attributes. Reports may disclose sensitive values or structural differences.
`h5copy`	Raw data; user-defined and structural metadata	Additional HDF5 files	Copies objects within or between HDF5 files, increasing the number of persistent copies of sensitive information.
`h5stat`	Structural metadata	Statistics reports, logs	Reports object counts, storage statistics, chunk information, free space, and other implementation characteristics that may reveal dataset complexity or workflow properties.
`h5debug`	Structural metadata	Debug output	Displays low-level internal file structures, addresses, B-trees, heaps, chunk indexes, and other implementation metadata intended for debugging.
`h5watch`	Raw data; user-defined metadata	Monitoring output, logs	Monitors SWMR datasets and reports newly written data, timestamps, and update activity in real time.
`h5import`	Raw data; user-defined metadata	New HDF5 file	Imports external data into HDF5, creating additional copies while potentially preserving metadata and semantics from the source format.
`h5jam/h5unjam`	User block contents	Extracted or embedded files	Stores or extracts arbitrary information in the HDF5 user block, including scripts, XML documents, application metadata, or proprietary information.
`h5mkgrp`	Structural metadata	Modified HDF5 file	Creates or modifies the group hierarchy. Limited direct privacy exposure but changes descriptive metadata and namespace organization.
`h5edit`	Raw data; user-defined metadata	Modified HDF5 file	Creates or modifies attributes; modifies raw data. Limited direct privacy exposure but changes data and user-defined metadata.

Table 9 – Privacy exposure through HDF5 command-line tools, continued

Tool	Exposed assets	Workflow artifacts	Privacy considerations
`h5format_convert`	Raw data; structural metadata	Converted HDF5 file	Converts files between HDF5 format versions, creating additional copies while preserving or rewriting internal metadata structures.
`h5clear`	Structural metadata	Modified HDF5 file	Clears SWMR status and metadata cache information. Primarily affects recovery metadata and file consistency information.

Note: All HDF5 command-line tools are built on top of the HDF5 library and therefore inherit its capabilities. When VFDs or VOL connectors are configured, these tools may transparently access data stored in remote services, external storage systems, or non-HDF5 data sources exposed through connectors. Consequently, privacy exposure is not necessarily limited to the HDF5 file specified on the command line but may extend to the broader HDF5 ecosystem.

2.5. Audit Findings and Mitigation Planning

Formal core and official-tool findings are registered as `CORE-SEC-001`, `CORE-SEC-002`, `CORE-TOOL-001`, `CORE-PRIV-001`, `CORE-SAFE-001`, and `CORE-SAFE-002` in Appendix A. The two shared-event security findings are provisional historical parser-boundary scenarios; they do not assert that every current HDF5 release contains every represented defect. The assert-masking observation is separately registered as evidence record `CORE-SEC-003` until exact source sites, supported release paths, and consequences are established.

Priority reconciliation.

The response order comes from Table 31, not from the order of recommendations in this chapter. Confirmed affected scope for `CORE-SEC-001`, `CORE-TOOL-001`, and `CORE-PRIV-001` receives Near-term treatment; `CORE-SEC-002` receives Planned treatment. `CORE-SAFE-001` and `CORE-SAFE-002` receive Near-term *containment* — capacity headroom and monitoring, flush and checkpoint dis-

cipline, a rehearsed recovery procedure, and independent copies of unregenerable data — with their structural treatment in CORE-SAFE-S1 and CORE-SAFE-S2 on the Planned and Phase 3 horizons; RM-2D produces their acceptance evidence rather than deciding whether they can be rated. The CORE-SEC-003 catalog and release-build differential study has a 30-day Phase 1 assurance target under RM-1E; this scheduling choice does not assign it a risk tier. A confirmed case is routed by its supported consequence and affected scope, including under CORE-SEC-001 for a memory-unsafe path or CORE-SEC-002 for termination or resource exhaustion. Phase 1 patching, isolation, and privacy controls treat concrete populations. Phases 2–4 provide reusable guardrails, class prevention, and sustainment; that structural work may begin in parallel but does not extend an earlier response clock.

2.5.1. Security and Safety

The selected CVE history shows repeated malformed-file parsing pressure against a complex binary metadata machine. Affected variants should be patched or contained promptly. Prevention must not stop at those local repairs: it should add systematic metadata validation at decode boundaries, explicit invariants for each on-disk structure, fuzz corpora tied to invariants, and hard separation between parser acceptance and internal object construction.

Goal: HDF5 should be able to parse any byte stream as hostile input and either reject it cleanly or produce a validated internal object graph. Anything else is security debt.

Structural prevention program: maintain current-scope fixes or containment while bootstrapping invariants, checked primitives, regression and fuzz harnesses, review gates, and metrics in parallel. Migrate each decode boundary in independently releasable tranches once its invariants and checked primitives exist, and make the gates enforce each accepted tranche. This program neither precedes nor delays Phase 1 treatment or Phase 2 release safeguards.

The CORE-S identifiers below are stable identifiers for recommendations in the core-security track. Their numbers do not express risk, urgency, effort, dependency, or implementation order.

Parser normalization is a central structural control: decode file bytes into a validated intermediate form before constructing internal objects so malformed metadata cannot reach trusted C paths. It is a recurrence-prevention control, not a substitute for patching or isolating a known affected release.

Our goal should be: *make malformed HDF5 files boring*. Every corrupted file should terminate in a controlled H5E_BADVALUE, H5E_OVERFLOW, H5E_CORRUPT, or H5E_RESOURCE path. No segfaults. No assertions. No leaks. No undefined behavior. No late-stage encode/flush crash from a bad decode.

The current CVE list includes many issues in H5FS, H5C, H5O, H5T, H5Z, and vector-copy paths, with repeated heap overflows, use-after-free, null dereference, and resource-consumption outcomes. HDF5's own format documentation explains why this happens: a file is a graph of groups, datasets, and named datatypes; on disk it is represented through lower-level structures such as the superblock, B-tree nodes, object headers, global heaps, local heaps, and free-space metadata. That means the library is parsing a dense graph of mutually referential, *attacker-controlled* metadata.

2.5.2. Prior Art and the Remaining Work

H5Lens / h5lens already provides machine-readable format definitions, h5policy preflight profiles and a regression corpus, a versioned invariant registry, the evidence-gated h5patch repair planner, and differential measurements against selected libhdf5 builds [30, 39, 38, 37]. Its strategy and bounded-decoder work also define the decode–validate–construct architecture [56, 58]. Two refinements are material: validation occurs at record-local, aggregate-object, and reference-graph scope; and materialization must remain separate from activation (cache publication, ID registration, plugin loading, decompression, callbacks, and external-file access). Locally valid metadata can fail only at a wider scope, and successful

parsing must not authorize a side effect. The latter is the core-library counterpart of APP-SEC-001.

Table 10 distinguishes reuse from remaining implementation. Effort is separate from inherent risk and is subject to the licensing boundary in Section C.3: identifiers, boundaries, fixtures, and measurements may be usable after provenance, licence, and, where needed, legal review; GPLv3 implementation source is not directly consumable into the BSD-licensed library.

Table 10 – Recommendations re-scoped against existing H5Lens work

Rec.	Existing asset	Remaining work	Effort
CORE-S1	Versioned per-family invariant registry with finding, test, fuzz, enforcement, and migration mappings.	Adopt the vocabulary; implement always-active enforcement inside libhdf5 and prove debug/release parity.	**Medium**
CORE-S2	Three validation scopes, materialization/activation rule, seven-phase migration design, and bounded-decoder demonstration.	Migrate the library; design reuse does not reduce that work.	**Large**
CORE-S4	Four exercised profiles with feature policy and resource budgets.	Port semantics to the H5P API and enforce defaults and policy inside libhdf5.	**Medium**
CORE-S5	Per-family mutation recipes, truncation sweeps, and a boundary-value coverage inventory.	Add OSS-Fuzz integration and the missing per-family targets.	**Medium**
CORE-S6	Manifested malformed-input and valid-control corpus with per-fixture outcomes.	Review specimen provenance; integrate HDF5 CI and debug/-DNDEBUG parity.	**Small**
APP-S2	Working profile-based preflight validator with structured results.	Bind the verdict to the same handle or verified identity used at load time; otherwise preflight remains advisory.	**Medium**

H5Lens does not supply this report's application-boundary findings, independent-implementation analysis, inherent/residual-risk separation, response clocks, accountable owners, or ecosystem attribution; those are complementary governance contributions. One loop also remains open: many tracked divergences are reported as oracle-rejected but libhdf5-unfixed. Two concern external-file-list handling,

including an uninvestigated case in which an external file is opened before its wrapped segment is validated [32]. If substantiated, it is an activation-before-validation instance relevant to APP-SEC-001; this report did not evaluate it. RM-1E therefore takes the full oracle-hardened set as triage input.

2.5.3. Core Security Control Program

The objective is controlled rejection of any hostile byte stream: no crash, assertion, leak, undefined behaviour, unbounded work, or late encode/flush failure caused by an invalid decode. The authoritative ownership, finding mapping, effort, package, and acceptance evidence for CORE-S0 through CORE-S12 are in Appendix B and Appendix C. The synthesis below preserves the control logic without duplicating repository layouts, sample APIs, YAML, or checklists.

Table 11 – Core security recommendations: strategic synthesis

ID	Control	Required outcome	Roadmap
CORE-S0	Separate defect and class closure.	A local repair may close the affected defect, including under embargo, but the paired class-prevention record stays open until sibling paths are structurally covered.	RM-1B, RM-3C, RM-4A
CORE-S1	Adopt the invariant registry.	Each declared on-disk condition maps to an always-active libhdf5 check, expected error, test, fuzz target, and migration state. Issue #87's seven families are the first audit perimeter; consequence and supported-release reachability, not candidate count, set tranche order.	RM-3A
CORE-S2	Enforce the parser boundary.	Bounded decode precedes record-local, object-wide, and graph validation; construction precedes a separately authorized activation step. No raw file quantity becomes a trusted pointer or side effect.	RM-3B
CORE-S3	Centralize checked arithmetic.	File-derived addition, multiplication, address ranges, allocation, copying, cursor movement, and graph traversal use checked wrappers or recorded exemptions.	RM-2A, RM-3A

Table 11 – Core security recommendations: strategic synthesis, continued

ID	Control	Required outcome	Roadmap
CORE-S4	Make open policy explicit.	`legacy`, `trusted-fast`, `untrusted-strict`, and `forensic` profiles bind validation, external access, plugins, traversal, metadata, allocation, and filter-expansion budgets to tested defaults.	RM-2A
CORE-S5	Fuzz structures and invariants.	Whole-file, structure-aware, and single-invariant mutation cover object headers, datatypes, dataspaces, layouts, heaps, filters, cache images, and tool rendering; coverage is reported per component.	RM-3C
CORE-S6	Make regression permanent and cheap.	Every closed case has a minimized, provenance-reviewed seed, expected controlled outcome, invariant mapping, and identical assertion-enabled/-DNDEBUG rejection.	RM-1B, RM-2C, RM-3C
CORE-S7	Block unsafe parser changes.	Review and automated gates require explicit bounds, checked arithmetic and copies, bounded traversal, partial-initialization cleanup, policy-aware plugin use, malformed-input tests, and always-active counterparts to file-derived assertions.	RM-1E, RM-3C
CORE-S8	Separate compatibility from safety.	Valid legacy files remain readable, but tolerated corruption does not become compatibility and repair never occurs silently during normal open.	RM-2C, RM-4A
CORE-S9	Treat tools as attack surfaces.	Inspection and conversion tools use forensic or strict profiles; rendering is bounded; legacy converters are explicit opt-ins; plugin-dependent tools require warning or approval.	RM-1B, RM-2A, RM-3C
CORE-S10	Govern executable extensions.	A protected manifest and allowlist bind identity, profile, capability, and resource limits; signatures prove provenance, not resource safety.	RM-3D
CORE-S11	Coordinate supply-chain controls.	Independently owned Scorecard, CodeQL, pinned-action, SBOM, signing, checksum, and SLSA controls run in parallel with parser work; only concrete dependencies affect sequence.	RM-3E, RM-4A

Table 11 – Core security recommendations: strategic synthesis, continued

ID	Control	Required outcome	Roadmap
CORE-S12	Measure class elimination.	A public dashboard reports a versioned denominator and named, expiring exceptions; metrics are coverage evidence, not defect closure.	RM-1B, RM-3C, RM-4A

The defect/class distinction requires the seven artifacts in Table 12. Recent fixes illustrate the intended direction: the file-space page-size fix converted an assertion into a corrupted-file error and reused its fuzzer, while cache-image reconstruction added buffer-size tracking, overflow checks, validation, and safe cleanup [35, 34]. The 2024 dataset-storage fix similarly added multiplication-overflow and file-bound checks [36]. These local repairs should become programme-wide invariants and primitives.

Table 12 – Required artifacts for defect closure and recurring-class tracking

Artifact	Purpose
crashing input	permanent corpus seed
minimized input	fast PR regression
root-cause category	invariant-class mapping
invariant update	sibling-defect prevention record
sanitizer regression	no crash, leak, or use-after-free
negative test	same controlled result with assertions enabled and disabled
reviewer-checklist update	reintroduction gate

Structure-aware fuzzing supplements whole-file mutation; OSS-Fuzz supplies long-running public execution and CI supplies fast feedback. These cadences are acceptance gates, not risk priorities.

Table 13 – Fuzzing acceptance criteria

Stage	Gate
PR	minimized CVE corpus under ASan/UBSan/LSan

Stage	Gate
nightly	structure fuzzers for core parsers
weekly	component-level coverage report
release candidate	72-hour soak on changed parser areas
CVE closure	crashing input added and mapped to an invariant

Backward compatibility does not authorize malformed metadata. The profile contract is therefore:

Table 14 – Compatibility and safety profile policy

Case	Required behaviour
valid legacy file	accept
ambiguous legacy construct	accept only in `legacy`/`trusted-fast`; warn in `forensic`; reject in `untrusted-strict`
formerly tolerated structural invalidity	reject in every non-legacy profile
repair required	explicit repair tool only; never silent normal-open repair
unknown message with unsafe flags	reject unless the profile explicitly allows preservation

Finally, the dashboard denominator must make declared coverage auditable:

Table 15 – Security programme metrics

Metric	Target
CVE corpus clean under ASan/UBSan/LSan	100%
decoders with explicit buffer size or cursor	100%
parser allocations through checked wrappers	100%
file-derived copies through checked wrappers or reviewed exemption	100%
declared-tranche structures with invariant files and fuzz targets	100%
catalogued file-derived assertions classified by dominance, reachability, and consequence	100%
declared invariants enforced with `-DNDEBUG`	100%

Metric	Target
new CVEs with invariant updates	100%
new crash to minimized seed	under 48 hours
minimized seed to invariant classification	under 5 business days
sanitizer-clean nightly fuzzing	sustained
untrusted-profile tests	every release

2.5.4. Safety: Durability and Recovery

The security track above answers "a hostile file arrived." The two recommendations here answer "my file did not survive." They address `CORE-SAFE-001` and `CORE-SAFE-002`, and neither has a counterpart elsewhere in this report. The `CORE-SAFE-S` identifiers name recommendations only; their numbers express no risk, urgency, effort, dependency, or implementation order.

Both findings are routed Near-term for *containment* and Planned or later for *structural treatment*. The containment obligations are operational and available today: capacity headroom with free-space monitoring, flush and checkpoint discipline, a rehearsed recovery procedure, and an independent copy of anything that cannot be regenerated. Nothing below is a prerequisite for those.

Recommendation CORE-SAFE-S1: Give HDF5 an `ENOSPC` story

Today an application can receive an HDF5 write, flush, or close failure, but it has no structured callback identifying the VFD's `ENOSPC` event, no supported protocol for choosing retry versus abandonment at that boundary, and no library state that marks the resulting file as untrustworthy. Four candidate mechanisms have already been proposed in the public design discussion; they are complementary rather than alternatives, and they differ sharply in cost [20].

Table 16 – Candidate `ENOSPC` mechanisms

Mechanism	What it provides	Cost and caveat
VFD-level error callback	The caller is told the driver hit `ENOSPC` and chooses retry, fail fast, or panic. This is the load-bearing item: it converts a silent unsafe state into an application decision.	Small to medium. Deliverable as a pass-through VFD wrapper first, without a public API commitment.
Poisoned-file state	A file that suffered a write failure is durably marked suspect, so a later open cannot silently treat it as sound.	Medium; needs a format-visible flag and a defined clearing procedure. Pairs with `h5clear`.
Space reservation (`posix_fallocate`)	Capacity is claimed before the write, converting a mid-write failure into an up-front one.	Medium; filesystem-dependent, and not available or meaningful on every backing store.
Pre-flush capacity check (`statvfs`)	A watchdog refuses to begin a flush that plainly cannot complete.	Small, but advisory only — it narrows the window and does not close it. A check that passes does not guarantee the flush succeeds.

Sequence them by that cost profile: the callback and the pre-flush check first, because they are small and independently useful; reservation and the poisoned state after, because they touch the format and the VFD contract. The public design discussion proposes a pass-through VFD wrapper as a low-friction proving ground before any public API change, and this report endorses that route.

Acceptance evidence. Fault injection returns `ENOSPC` at each of write, flush, and close, for each supported VFD and buffering profile. For every injected

point: the application receives a distinguishable `ENOSPC`-class error rather than a generic failure or a success; the last durable state is recorded; a subsequent open either succeeds, fails with a diagnostic naming the condition, or reports the file as suspect — never silently returns wrong data; and the documented recovery procedure is executed and its result recorded. A run in which the library reports success and the file is later found inconsistent is a failure of this recommendation regardless of the error codes returned.

Recommendation CORE-SAFE-S2: State, then improve, the crash-consistency guarantee

The immediate deliverable is not a mechanism. It is a written statement of what HDF5 guarantees today when a writer dies mid-update — for each of the default single-file layout, SWMR, and the split and multi drivers — and, explicitly, what it does not. Practitioners currently infer this from forum threads and folklore, which is why the same questions recur across sixteen years [16, 11, 15]. A documented weak guarantee is more useful than an undocumented one, because it lets an application decide whether it needs checkpointing, an external journal, or a different store.

The structural work is a multi-release programme and is already under investigation: metadata journaling, write-ahead logging, mechanisms derived from SWMR, and checkpointing [9]. This report does not select among them. It records three constraints on whichever is chosen.

1. **Recovery tooling is part of the guarantee, not an afterthought.** Whether a file reopens after `h5clear` is the difference between I2 and I3 in `CORE-SAFE-002`. Improving crash consistency without improving diagnosis and repair moves files from one failure class to another rather than reducing harm.
2. **Silent wrong data is the outcome to design against.** A file that fails to open is a bounded operational problem. A file that opens and returns data from an inconsistent state is a scientific integrity problem, and it is the one that will not be noticed.

3. **The guarantee must be testable by fault injection,** and its test must run in the release build configuration, not only under assertions — the same discipline `CORE-S7` applies to parser invariants.

Acceptance evidence. Published documentation states the crash-consistency guarantee per layout and driver, including its limits. A fault-injection harness kills the writer at a matrix of points across metadata update, flush, and close, and for every point records whether the file opens, opens after repair, or does not open, and whether any data read back differs from what was acknowledged as written. The last case is reported separately and treated as a defect, never as expected behaviour. Results are published per supported release so that the guarantee can be seen to hold or to change.

2.5.5. Privacy

This section summarizes the Privacy Exposure Model's mechanisms and the bounded `CORE-PRIV-001` finding; the other mechanisms are not separately rated. It then describes corresponding mitigation strategies.

Overview of Privacy Threat Categories

The HDF5 Privacy Exposure Model [88] identifies seven exposure families that describe mechanisms through which sensitive information may be unintentionally exposed in HDF5. These mechanisms can arise during normal scientific data creation, processing, sharing, and preservation; their frequency, affected population, and consequence remain deployment-specific.

Several threat categories involve direct disclosure of information stored within HDF5 files. Semantic Metadata Exposure (P1) occurs when descriptive information, such as object names, attributes, datatype member names, or other user-defined metadata, reveals sensitive contextual information even when the underlying scientific data are protected. Structural or On-disk Leakage (P2) arises from examination of HDF5's internal storage structures, including file layout, storage

organization, and implementation metadata, which may disclose characteristics of the stored data without requiring HDF5 software. Raw-data Pattern Leakage (P3) addresses situations in which information about scientific data becomes observable through storage patterns, embedded textual descriptions, or other implementation artifacts rather than through intended data access.

Another group of threats results from routine scientific data processing. Unintended Inclusion During Processing (P4) occurs when software automatically generates, expands, or preserves additional information during data conversion or processing. Aggregation, Correlation, or Inference (P5) recognizes that individually non-sensitive datasets may collectively reveal sensitive information when combined or interpreted together. Persistence Beyond Intended Lifetime (P6) addresses privacy risks associated with temporary files, metadata snapshots, cached datasets, archived copies, and other artifacts that remain accessible after their intended purpose has ended.

The remaining threat category Misplaced Trust in Privacy by Format (P7) reflects broader characteristics of modern HDF5 ecosystems. Examples include:

- Misinterpretation of data privacy in HDF5 format leads to the possibility of sensitive information disclosure through metadata, structural information, or inference.
- Conversion to HDF5 highlights that transforming legacy scientific formats into HDF5 often increases accessibility and interoperability by making implicit information explicit, but this additional semantic information may itself introduce new privacy risks.
- Loss of Data Control describes situations in which data movement, replication, backup, cloud migration, or transfer across organizational boundaries weakens the original privacy protections and governance mechanisms applied to the data.

Although presented as distinct categories, these mechanisms can occur together. A single scientific workflow may enrich metadata during format conversion, aggregate datasets from multiple sources, generate temporary processing artifacts, and replicate results across multiple storage systems. Consequently, privacy risks

should be evaluated across the complete scientific data lifecycle rather than by considering individual HDF5 files in isolation.

Recommendations

Because HDF5 was designed to maximize scientific data sharing, portability, interoperability, and long-term usability rather than enforce privacy, no single mechanism can eliminate all privacy risks. Effective privacy protection therefore requires a multi-facet approach that considers the entire scientific data lifecycle, including data creation, processing, storage, sharing, transfer, and archival. Privacy should be treated as a property of the complete HDF5 ecosystem including applications, plugins, workflows, storage systems, and organizational practices rather than solely of the HDF5 file format. The following `CORE-PRIV-S` identifiers identify recommendations in this track. Their numbers do not express risk, urgency, effort, dependency, or implementation order.

For a confirmed `CORE-PRIV-001` sharing boundary, the Near-term route applies: `RM-1D` first bounds the population and recipients and tests minimization and recipient controls; `RM-2G` is conditional when policy still requires whole-file or equivalent all-metadata protection. `CORE-PRIV-S2` and extension controls support the unrated `EXT-PRIV-001` evidence and assurance track and inherit no urgency from `CORE-PRIV-001`.

Recommendation CORE-PRIV-S1: *Privacy exposure awareness*

Scientists, software developers, data stewards, and infrastructure providers should recognize that HDF5's self-describing nature inherently exposes contextual information in many ways. Privacy reviews should therefore evaluate not only scientific datasets, but also metadata, structural organization, workflow artifacts, derived products, and information that may be inferred through aggregation or correlation. Reviews should also take into consideration possible privacy exposure during data lifecycle.

Awareness is not itself a control, so this recommendation is written to produce an artifact rather than a disposition of mind.

Acceptance evidence. A dated privacy review record exists for the defined data population and sharing boundary, and it contains, at minimum:

(a) the named data owner and the intended recipient set for each disclosure boundary;

(b) an enumeration of sensitive items found in each asset class identified in Section 2 — data, user-defined metadata, structural metadata, derived or reconstructed information, and workflow artifacts — or a positive statement that the class was examined and found empty;

(c) the lifecycle stages examined, with the stages not examined named rather than omitted; and

(d) a disposition for every enumerated item: retain, minimize, tokenize, restrict recipients, or accept with a recorded rationale.

A review that examines datasets only, or that records no negative findings for the metadata and artifact classes, does not satisfy this recommendation.

Recommendation CORE-PRIV-S2: *Privacy exposure through HDF5 tools and plugins awareness*

The HDF5 ecosystem extends beyond the core HDF5 software (i.e., file format and the library), and includes a diverse collection of command-line utilities, visualization tools, analysis libraries, VOL connectors, VFDs, filters, and third-party applications. These components can generate additional outputs such as logs, diagnostic messages, temporary files, metadata snapshots, converted datasets, and cached intermediate results that may expose sensitive information beyond the contents of the original HDF5 file. Furthermore, plugins may introduce new metadata, or transform data in ways that alter their privacy characteristics. Consequently, privacy assessments should evaluate not only HDF5 files but also the behavior of the software ecosystem that processes them. Developers should document the privacy implications of their tools, minimize unnecessary

generation of sensitive artifacts, and provide users with mechanisms to control logging, metadata generation, and retention of intermediate products.

Acceptance evidence. For each in-scope tool, connector, filter, and VFD:

(a) published documentation lists the outputs it can generate — stdout and stderr, redirected files, logs, temporary files, caches, snapshots, converted or derived files — and the option, property, or environment control that suppresses or bounds each one;

(b) the list is confirmed by a reproducible trace of a representative invocation rather than asserted from the source, and the trace is retained; and

(c) any output observed in the trace but absent from the documentation is recorded as a defect against that component.

Table 9 is the starting inventory for the official command-line tools, not a completed trace. This recommendation supports the unrated `EXT-PRIV-001` evidence track and inherits no response urgency from `CORE-PRIV-001`.

Recommendation CORE-PRIV-S3: *Privacy-aware system design*

HDF5 software systems should be built with data privacy in mind. Applications should minimize collection and retention of unnecessary metadata, avoid embedding sensitive information in object names or descriptive attributes, remove temporary processing artifacts when they are no longer needed, and carefully evaluate automatically generated metadata introduced during data conversion or interoperability. Workflow designers should review intermediate products, logs, cached datasets, and derived files before publication or long-term archival.

Acceptance evidence. The claim to be tested is that minimization actually removed the information, not that it was removed from the API view:

(a) before-and-after inspection of a representative file through *both* the HDF5 API and a raw-byte search — at minimum `strings`-class extraction over the whole file, including the user block and any free or unreferenced space — shows that each item marked *minimize* or *tokenize* in the `CORE-PRIV-S1` review is absent from both views;

(b) metadata introduced automatically during conversion is enumerated for the tested pipeline and each item is dispositioned; and

(c) temporary artifacts are removed under a stated retention rule, and the rule is verified by inspecting the working area after both normal and abnormal termination.

An item that disappears from `h5dump` output but is still recoverable from the raw bytes has not been minimized. Where the file is rewritten to achieve this, record pre- and post-checksums and preserve provenance.

Recommendation CORE-PRIV-S4: ***Data encryption as privacy control***
Encryption represents an important but limited privacy control.

Currently, HDF5 doesn't support data encryption that allows I/O on encrypted data, effectively disabling sensitive data exposure during data transfer and storage. Users may implement custom raw data encryption and partial metadata encryption (i.e., names of the objects) using existing HDF5 filtering mechanism. Such approach still leaves user-defined metadata (e.g., attributes) and structural metadata unencrypted. Information that remains unencrypted as application-specific workflow artifacts, temporary files, system logs, or derived datasets may continue to disclose sensitive information.

HDF5 encryption feature that properly implements encryption of all data stored in HDF5 while allowing partial I/O on encrypted data is currently under development. However, encryption does not eliminate all HDF5 privacy risks. Once encrypted data are decrypted for legitimate processing, applications, plugins, caches, diagnostic logs, and intermediate workflow products may again become sources of unintended exposure. Encryption therefore reduces the attack surface but does not address inference, aggregation, excessive metadata generation, data retention, or loss of governance. Encryption also complicates workflows and may lead to data loss if on some stage of data life cycle encryption keys become unavailable.

Therefore, encryption should be viewed as one component of a comprehensive privacy strategy rather than as a complete solution. Effective protection requires

combining encryption with careful metadata management, lifecycle-aware data governance, privacy-aware application design, access controls provided by the underlying computing environment, and education of users regarding the unique privacy characteristics of HDF5-based scientific data systems.

Acceptance evidence. Encryption is credited only for the boundaries actually tested, and only when recoverability is demonstrated alongside confidentiality:

(a) ciphertext and metadata inspection shows that the items the `CORE-PRIV-S1` review marked as requiring protection are not observable at the protected boundary, tested through both the HDF5 API and raw-byte search;

(b) an unauthorized-open test fails closed, and a partial-I/O workflow test confirms the protected file remains usable for its intended access pattern;

(c) a protected key inventory names a key custodian per key, and backup/restore evidence exists; and

(d) a loss-and-recovery exercise is performed and passes.

Item (d) is not optional. Encryption without a tested recovery path converts a disclosure risk into a durable data-loss risk, which is a different harm and not an improvement. Until all four pass, the affected sharing boundary is avoided or a tested access restriction is maintained within the `CORE-PRIV-001` Near-term clock. This recommendation is the conditional branch `RM-2G`, reached only where `RM-1D` minimization and recipient controls cannot meet the disclosure policy.

3. HDF5 Library Extensions

3.1. Assessment Boundary

Supply-chain issues are treated in Section 6. This chapter addresses a different question: how VOL connectors, filter plugins, and Virtual File Drivers (VFDs) can extend a deployment's privacy boundary beyond the logical HDF5 file.

The P1–P7 labels use the HDF5 Privacy Exposure Model summarized in the core chapter: P1 semantic metadata, P2 structural or on-disk leakage, P3 raw-data patterns, P4 unintended inclusion during processing, P5 aggregation or inference, P6 persistence beyond intended lifetime, and P7 misplaced trust in privacy by format. `EXT`, `OPS`, `TCD`, and `SCD` are separate SSP tags for extensions, operations, toolchains, and distribution. These labels classify candidate mechanisms, not inherent risk or response urgency. Without a bounded deployment, data population, recipient, retention condition, and consequence, the chapter-wide observation remains unrated evidence-backlog item `EXT-PRIV-001`.

3.2. Virtual Object Layer Connectors

A connector, or a connector stack, can present external and heterogeneous resources through the HDF5 API. Protecting or inspecting the apparent file is therefore not sufficient to establish where all data originated, where they were processed or retained, or which policies applied. Table 18 consolidates the mechanisms that must be tested in a concrete deployment.

Extension	Position in workflow	Boundary added to the assessment	Candidate families
VOL connector	Above or in place of native object storage	External and non-HDF5 sources, aggregation, generated objects and metadata, network services, credentials, caches, and connector code.	P1, P4–P7; SSP: EXT, OPS, SCD
Filter	Immediately before storage and after retrieval	Pipeline metadata, compressed-size patterns, plaintext buffers, derived information, temporary artifacts, and dynamically loaded code.	P1–P7; SSP: EXT, SCD
VFD	Physical byte storage and retrieval	Storage locations, replicas and histories, physical layout and access patterns, network and cloud infrastructure, and VFD code.	P2–P7; SSP: EXT, OPS, SCD

Table 17 – Extension mechanisms and the deployment boundary they can add

Table 18 – Candidate privacy-exposure families for VOL connector mechanisms

Mechanism	Deployment-relevant observation	Families / tags
External and non-HDF5 sources	A connector may retrieve and integrate cloud objects, databases, remote REST or other network services, live streams, TIFF/GeoTIFF, BUFR, GRIB2, CDF, DAOS, or other sources, and may generate derived data or virtual objects. Their origin, location, ownership, and governing policy need not be visible to an application that sees one logical HDF5 dataset.	P7; SSP: EXT
Aggregation and policy crossing	Independent sources can be combined without an explicit application-level policy decision. For example, geospatial records, near-real-time weather and disaster feeds, and property assessments could appear as one property-impact dataset even though ownership, access, retention, or privacy rules differ.	P5, P7; SSP: EXT, OPS
Generated metadata and provenance	Relationships, locations, URLs, bucket names, origin, timestamps, organizational structure, and modification history can disclose context while raw data remain protected. BUFR and GRIB2 connectors may create file inventories, derived attributes, and dimension scales.	P1, P4; SSP: EXT
Network and cache artifacts	Headers, DNS and proxy logs, client history, authentication records, local caches, temporary downloads, raw data, metadata, and credentials can record which datasets were used and may remain after execution or be copied and archived. Networked connectors therefore inherit the surrounding network's privacy and security properties.	P4, P6; SSP: EXT, OPS
In-process plugin trust	A dynamically loaded connector executes in the application's process and can inspect, modify, copy, cache, export, or share data before returning. Its code, provenance, search path, and configuration belong in the trusted computing base.	P4, P6; SSP: EXT, SCD

3.3. Filter Plugins

Filters process data immediately before storage and immediately after retrieval. Their behavior, execution environment, pipeline metadata, and artifacts are therefore part of the privacy assessment. Table 19 combines the cross-cutting mechanisms with representative built-in and third-party filters; it does not assign risk to a filter merely because a mechanism is possible.

Table 19 – Primary privacy mechanisms associated with HDF5 filters

Mechanism or filters	**Representative consideration**	**Families / tags**
Pipeline identifiers and parameters	The object header records the pipeline required to interpret stored data. Identifiers and settings may reveal a security mechanism, organizational standard, software ecosystem, application domain, precision, resolution, floating-point accuracy, loss tolerance, or experimental assumptions. This metadata generally cannot be removed without affecting interoperability.	P1, P3; SSP: EXT
Deflate (GZIP), Zstandard, LZ4, and BZIP2	Method, level, block choice, and resulting chunk sizes can reveal entropy, sparsity, redundancy, and structure. LZ4's fast, modest-ratio behavior and BZIP2 block characteristics are instances of the same observable.	P1–P3; SSP: EXT
Shuffle and Blosc	Reordering is reversible, but its use with compression, block sizes, shuffle, or bitshuffle options can disclose numeric representation and data organization.	P1, P3; SSP: EXT
Fletcher32	Checksum metadata reveals that integrity protection is used but generally introduces little additional privacy exposure.	P1; SSP: EXT
N-bit and Scale-Offset	Bit width, scale, and offset can reveal effective precision, expected value ranges, resolution, and acceptable quantization error.	P1, P3; SSP: EXT

Table 19 – Primary privacy mechanisms associated with HDF5 filters, continued

Mechanism or filters	Representative consideration	Families / tags
ZFP, SZ, and SZ3	Fixed-rate, fixed-precision, fixed-accuracy modes and error bounds disclose acceptable information loss and numerical-accuracy requirements.	P1, P3; SSP: EXT
Plaintext processing and derived information	Filters see plaintext buffers before protective compression or encryption and may compute or retain statistics, patterns, sparsity, or other derived information. Buffers may remain in memory, swap, or crash dumps; implementations may also create logs or temporary files.	P4–P7; SSP: EXT
Dynamic loading and plugin trust	A filter loaded from an environment-controlled search directory executes in the application process and may inspect memory, copy decoded data, log, retain, modify, or transmit it. Search results can differ between systems; untrusted or user-writable paths therefore create a supply-chain boundary.	P4, P6, P7; SSP: EXT, SCD

3.4. Virtual File Drivers

VFDs do not change the logical HDF5 data model, but determine where and how bytes are stored and retrieved. A logical file may reside in cloud object storage, a distributed file system, network storage, process memory, or across multiple devices, so its apparent name need not identify the governing storage or policy domain. Consequently, physical layout, copies, communications, storage-provider controls, and artifacts can matter even when logical HDF5 objects are adequately protected.

Table 20 – Candidate privacy-exposure families for HDF5 Virtual File Drivers

VFD or mechanism	Representative consideration	Families / tags
SEC2 and STDIO	Persistent storage can expose residual information in reused blocks if the storage layer does not securely initialize or overwrite them.	P6
Core	The file resides in process memory and may persist in swap or crash dumps; with the backing-store option, it is written to persistent storage at close.	P6
Family, Split, Multi, and Subfiling	One logical file becomes multiple independently managed files. Family and Subfiling divide data across members or devices; Split separates metadata and raw data; Multi separates HDF5 memory types. Different permissions, locations, or incomplete deletion can disclose fragments, metadata, or raw data.	P2, P6, P7; SSP: EXT
Mirror, MPI-I/O, and ROS3	Mirror replicates to a remote service and depends on communication, authentication, daemon trust, and protection of copies. MPI-I/O passes data through HPC communications unless protected below HDF5. ROS3 uses S3-compatible stores; HTTP requests and Range requests, DNS, and cloud access logs create artifacts and disclose access.	P4, P6, P7; SSP: EXT, OPS, SCD
Log (VFD SWMR) and Onion	Metadata journals retain snapshots for recovery. Onion retains historical versions, allowing recovery of deleted or overwritten data and reconstruction of metadata evolution, provenance, and workflow history.	P2, P5, P6; SSP: EXT
Physical layout and access patterns	Offsets, allocations, file organization, access frequency, read/write ratios, sequential or random I/O, update frequency, and touched regions can reveal dataset, algorithm, storage-architecture, or user characteristics to operating systems, distributed stores, servers, or cloud providers.	P2, P3; SSP: EXT

Table 20 – Candidate privacy-exposure families for HDF5 Virtual File Drivers, continued

VFD or mechanism	Representative consideration	Families / tags
Workflow artifacts and plugin trust	Temporary files, swap, memory images, crash dumps, replicas, journals, and cloud logs can persist outside the logical file. A dynamically loaded VFD can inspect, modify, retain, duplicate, or export data in process and therefore belongs in the trusted computing base.	P4, P6; SSP: EXT, SCD

3.5. Control Proposals and Routing

The mechanisms above support control design but do not silently promote `EXT-PRIV-001` into a formal finding. The authoritative recommendation register and phased roadmap provide ownership, dependencies, sequencing, and acceptance evidence. The identifiers below express neither rank nor urgency.

Table 21 – Extension privacy and trust recommendations

Identifier	Control outcome
`VOL-S1`	Document external sources, generated metadata, caching, network communications, authentication, workflow artifacts, and privacy assumptions for each connector.
`VOL-S2`	Standardize introspection for connector name, version, stack, source types and locations, network use, cached data or credentials, temporary or persistent artifacts, generated or derived data, cross-policy aggregation, and privacy configuration. Existing identity and capability callbacks do not expose all of these characteristics.
`VOL-S3`	Treat connectors as trusted code: restrict search paths, exclude user-writable locations, and verify identity, provenance, and integrity; support enforceable loading policy.
`VOL-S4`	Minimize unnecessary generated metadata and persistent caches, unless explicitly requested, and document unavoidable privacy characteristics during connector design.

Table 21 – Extension privacy and trust recommendations, continued

Identifier	Control outcome
FILTER-S1	Apply the same trusted-plugin, restricted-search-path, provenance, integrity, and enforceable-loading controls to filters.
FILTER-S2	Select and document pipelines with explicit consideration of what identifiers, parameters, precision, compression, and workflow choices reveal.
FILTER-S3	Where a file recipient must not see pipeline or structural metadata, evaluate whole-file protection covering raw data and metadata. Key management and recovery evidence follows RM-2G. Such protection can prevent that recipient from inspecting filter identifiers, parameters, object headers, and structural metadata, but does not treat caches, logs, authorized misuse, post-decryption inference, retention, or key-loss risk.
FILTER-S4	Minimize workflow artifacts, erase temporary buffers when possible using an appropriate secure method, avoid persistent temporary files, and document remaining artifacts.
FILTER-S5	Standardize filter documentation for generated metadata, plaintext processing, derived information, temporary artifacts, and privacy implications of parameters.
VFD-S1	Document storage locations, additional copies, network use, workflow artifacts, and storage assumptions for every VFD.
VFD-S2	Give replicated files, journals, histories, and backing stores secure deletion, retention, and lifecycle controls.
VFD-S3	Document authentication, communications, logging, and storage-provider privacy assumptions for cloud and distributed VFDs.
VFD-S4	Apply trusted-plugin, restricted-search-path, provenance, integrity, and enforceable-loading controls to dynamically loaded VFDs.

The trusted-loading proposals call for configurable policy based on plugin identity, exposed characteristics, or administrative rules, in addition to deployment-side search-path and provenance controls.

For VOL-S3, FILTER-S1, and VFD-S4, signed-plugin verification requires HDF5 2.2.0 or a later verified release built with HDF5_REQUIRE_SIGNED_PLUGINS=ON, a configured and protected trusted keystore, signed approved

artifacts, and a test that unsigned or untrusted plugins are rejected. Upgrading alone does not enable the control (Table 46).

No report-wide risk urgency is assigned to EXT-PRIV-001. Phase 2 RM-2E establishes a bounded deployment, data flow, recipient, retention condition, and consequence; that administrative target is not a risk-response clock or a Low-risk conclusion. RM-3D and RM-3F then provide proactive extension assurance and evidence-dependent privacy treatment. A deployment matching separately rated plugin-search finding APP-SEC-005 instead follows that finding's Near-term application clock; it does not assign a High tier to extensions as a class.

4. HDF5 Wrappers and Language Bindings

4.1. Boundary Model

The SSP SIG's HDF5 Python Wrapper (H5PW) model [44] defines ten mechanism families that generalize to any binding between HDF5 and a managed runtime: **W1** native parser or codec compromise, **W2** code in data, **W3** dynamic-loader boundary failure, **W4** concurrency or lifecycle mismatch, **W5** resource exhaustion or amplification, **W6** ABI, dependency, or packaging drift, **W7** metadata, log, or artifact exposure, **W8** trust-boundary confusion, **W9** misleading safety signal, and **W10** external-reference traversal. The labels classify mechanisms and carry no risk rating.

Bindings for C#, Java, Python, JavaScript, Rust, and other languages mediate HDF5's C interfaces into different object, memory, exception, packaging, and deployment models. The Python cases below are boundary references, not an exhaustive binding review; none establishes that a downstream framework failure was caused by the binding through which HDF5 was reached.

4.2. Python-Facing Cases

All four advisories reached users through Python-facing workflows, but the binding performed as specified and the missing decision was above it.

Case	Boundary conclusion	Routing / family
CVE-2025-9905 [68]	Keras legacy `.h5` loading executed serialized `Lambda` code despite `safe_mode=True`. Application-level framework-object deserialization, not HDF5 decode, was unsafe; the safety label did not govern this loader path.	`APP-SEC-002`; W2, W9

Case	Boundary conclusion	Routing / family
CVE-2026-1669 [70]	Model loading followed a documented HDF5 external raw-storage reference to a local path and disclosed its contents. The consumer lacked a policy deciding whether this file could exercise that capability.	APP-SEC-003; W10
CVE-2026-0897 [69]	Weight loading allocated from a file-declared shape without a budget. HDF5 can legitimately describe a logical shape much larger than the useful payload, so the consumer must enforce the budget.	APP-SEC-004; W5
CVE-2026-2492 [59]	TensorFlow's integration searched an unsecured location for plugins. Dynamic loading is an HDF5 mechanism; the defect was the embedding application's search-path policy.	APP-SEC-005; W3

Together the cases instantiate H5PW **W8**: a binding faithfully exposes an HDF5 capability, while the security decision—whether this process, with this authority, should honor it for this file—belongs above the binding. All five application-boundary findings in this report map to named W-families; the vocabulary anticipated the boundary independently of this assessment. Accordingly, APP-S1 asks bindings to surface controls through their APIs rather than leave them reachable only through environment variables.

4.3. Coverage and Mitigation Routing

No Java-specific adverse-event scenario, or additional binding-specific scenario, was sufficiently developed for a formal finding. Java remains in the ecosystem census and control-exposure scope; W1, W3, W4, and W6 are especially applicable to its JNI boundary and provide a vocabulary for the next review.

The four cases remain formal application findings in Appendix A, not h5py or Python-binding findings. Wrappers therefore receive no separate report-wide urgency: they participate in the Phase 1 version, linkage, feature, and authority census and expose the controls required by the applicable application package.

Where a binding is part of a deployment matching a formal finding, that finding's response clock controls; mentioning a CVE here does not create a second priority.

5. Applications using HDF5

Applications treat HDF5 as inert data, but HDF5 files can carry behavior-relevant metadata: paths, external references, huge declared shapes, filter/plugin requirements, serialized application objects, and model-layer definitions. That creates integration bugs even when the HDF5 library is behaving as designed.

The HDF Group's security policy [89] already attempts the right scope distinction: core libhdf5, official tools, Remote Code Execution (RCE) via parsing/API abuse, and supply-chain compromise are in scope; third-party plugins and application-level usage are usually triaged or referred to their maintainers.

Table 23 – Representative application-level HDF5 mechanisms

Mechanism	Consequence	Boundary evidence
Malformed metadata in C paths	Memory faults or termination	Recurring core/tool records; the bounded official-tool example is `CORE-TOOL-001` [28, 84].
Application deserialization	Code execution	Legacy Keras HDF5 loading bypassed `safe_mode`; application objects, not the format, supplied execution [68, 63].
External resources	File disclosure or traversal	File-selected paths were resolved with application authority in Keras and the Hugging Face service [70, 64, 67].
Declared size or shape	Memory exhaustion	Keras allocated from an extreme HDF5-declared shape before enforcing a budget [69, 62].
Dynamic plugin search	Unintended code loading	TensorFlow searched an unsecured plugin location [59, 91, 79].
Vendored/stale copies	Inherited defects and ambiguous applicability	Downstreams may statically link or lag HDF5, as illustrated by matio and an Apple removal [27, 6].

5.1. Boundary Case Study: External Raw Storage and the Hugging Face Incident

The July 2026 Hugging Face incident is a concrete example of an application trust-boundary failure involving an intentional HDF5 capability. Hugging Face reports that its dataset service processed attacker-supplied, valid HDF5 files whose datasets declared local paths as external raw storage. The worker read `/proc/self/environ` and source files with its own filesystem permissions and returned the bytes as dataset rows. Hugging Face expressly distinguishes this file-disclosure vector, which executed no code, from a separate Jinja2 injection that supplied code execution [67].

The HDF Group likewise concludes that external raw storage was the mechanism, not an HDF5 parser vulnerability: the documented feature behaved as designed, while the authority-bearing service acted as a confused deputy. Updating libhdf5 alone would not remove that service boundary; the relevant controls are fail-closed external-resource policy, preflight inspection, path confinement, least-privileged workers, and credential isolation [90].

The h5py community expressly considered the same boundary. Maintainers described per-dataset external-storage checks as possible but laborious, recommended sandboxing genuinely untrusted files, opened upstream issue #6618 because h5py had no direct enforcement mechanism, and discussed a simpler detection API. Issue #6618 in turn asks for unified controls at or before external-resource resolution across raw storage, external links, and virtual-dataset sources. The discussion records policy and API proposals, including use of the external `h5policy` prototype as a proving ground; this report does not treat those proposals as delivered controls [25, 78].

Table 24 separates the observed incident record from the control inference. That separation matters because the file was not reported to exploit memory corruption, and the service's authority — not the input's authority — made the disclosure possible.

Table 24 – Authority and control chain in the Hugging Face incident

Stage	Evidence and boundary	Audit consequence
Resource selection	The remote party supplied a valid HDF5 file whose external raw-storage metadata named local paths. The originator selected the resource but did not bring the worker's filesystem authority.	Validity and parser correctness do not answer whether the application may honor a file-selected resource. External access needs an explicit policy decision.
Authority-bearing resolution	The dataset worker resolved the paths with its own permissions and read `/proc/self/environ` and source files. The intentional HDF5 mechanism carried the request across the service boundary.	Deny or constrain resolution before the read; independently reduce worker authority, isolate credentials, and confine filesystem access.
Observable consequence	The service returned the bytes as dataset rows. Hugging Face distinguishes this file disclosure from the separate Jinja2 injection that supplied code execution [67].	Rate the demonstrated disclosure, not an unreported HDF5-code-execution path. Constrain output and retain resolution logs so attempted access is observable.
Reusable-control gap	The affected renderer was changed, while h5py and HDF5 discussions sought a less laborious detection or enforcement surface spanning raw storage, external links, and virtual-dataset sources [25, 78].	Service containment and reusable library support are different closure claims. Until enforcement ships and is integrated, sandboxing and application policy remain necessary.
Closure evidence	A preflight result can become stale if the file, path environment, or opened object changes before use. A detector alone also cannot decide which worker authority is acceptable.	Bind the decision to the same handle or immutable file identity, recheck at resolution, and test denial across every enabled external-resource family.

Formal finding APP-SEC-001 in Appendix A rates the incident at the affected service boundary. The similar Keras external-dataset failure is separately registered as APP-SEC-003; recurrence of a mechanism does not erase the differences in product, version, and deployment scope.

The unifying root cause. The repeated mistake is *trust-boundary confusion*: treating HDF5 as inert data even though it can influence allocation, shape, traversal, codec/plugin loading, external-file lookup, conversion, object reconstruction, and tool output. The SSP SIG separately names the same family as I9 in its independent-implementation model and W8 in its Python-wrapper model [42, 44]. That is internal consistency between programme artifacts, not independent validation. The failure occurs when a model loader, ingestion service, workflow, viewer, or plugin resolver interprets file-controlled metadata with authority the file's originator did not possess.

5.2. Audit Findings and Mitigation Planning

The formal application findings are APP-SEC-001 through APP-SEC-005 in Appendix A. Their ratings apply to the bounded versions and deployments stated there, not to Keras, TensorFlow, Hugging Face, or applications using HDF5 as undifferentiated projects.

Priority reconciliation.
Any equivalent deployment that still gives untrusted HDF5 input automatic access to file-selected external resources follows the Immediate route for APP-SEC-001: contain it before the next job and no later than 24 hours after identification. Confirmed affected scope for APP-SEC-002, APP-SEC-003, and APP-SEC-005 receives Near-term treatment; APP-SEC-004 receives Planned treatment. The historical Hugging Face rating is not a current residual-risk rating of that service. Application patches and compensating controls in Phase 1 must not wait for the reusable core APIs and integrations proposed for Phase 2.

Mitigation has to be layered. Patching `libhdf5` alone will not fix Keras model deserialization, TensorFlow plugin path handling, or application-level external-reference policy.

The `APP-S` identifiers below identify recommendations in the downstream- application track. Their numbers do not express risk, urgency, effort, dependency, or implementation order.

Table 25 – Application-boundary control set

ID	Outcome	Minimum control
`APP-S1`	Untrusted-HDF5 profile	Default-deny dynamic plugins and external resources; cap rank, objects, metadata, traversal, logical bytes, allocation, and filter expansion.
`APP-S2`	Enforcement-coupled preflight	Return machine-readable reasons before allocation or deserialization, bind the decision to the same handle or immutable file identity, and recheck targets at runtime. Existing validator prior art does not supply that coupling.
`APP-S3`	Safe ML-loader defaults	Reject executable legacy reconstruction, external datasets, dynamic plugins, and allocation beyond aggregate and per-object budgets.
`APP-S4`	External resources opt-in	Reject absolute paths, traversal, and working-directory lookup; require an explicit sandbox root or allowlist and log each resolution.
`APP-S5`	Plugins governed as code	Set loading policy explicitly; use protected allowlisted directories and provenance; isolate and minimize privilege where a plugin is indispensable.
`APP-S6`	Downstream provenance	Report HDF5 version and linkage, enabled filters/connectors, plugin-path and external-resource policy, and loader safe-mode behavior.
`APP-S7`	Advisory scope tags	Distinguish core parser, official or legacy tool, plugin loading, external resource, downstream deserialization/resource policy, vendored library, non-HDF5, and duplicate/rejected records. Tags support triage but do not rate risk.

Preflight must remain bound to enforcement. A metadata scan is advisory unless its decision is attached to the same open handle or a verified immutable file identity and the runtime fails closed when the file, policy, or external-resource target changes. The validator should report reasons such as an out-of-sandbox path, an excessive logical size, a required plugin, or executable application metadata before the loader allocates arrays or constructs application objects. Runtime hooks must then enforce the same limits at external resolution, traversal, filter activation, and object construction.

Use current controls while shared APIs mature. Applications can already disable dynamic plugins with `H5PLset_loading_state(0)` or `HDF5_PLUGIN_PRELOAD=::`; service launchers can clear or constrain `HDF5_PLUGIN_PATH` and `HDF5_EXTFILE_PREFIX`; and process isolation can remove filesystem, credential, and network authority. Those controls should be published as deployment recipes now. The Phase 2 profile and external-resource APIs make the policy cohesive and binding-friendly; they do not justify waiting to contain an identified deployment.

Deployment minimum. Unknown-file viewers and upload services require sandboxing, external-resource denial, plugin denial, and metadata/output budgets. Model loaders additionally reject executable reconstruction and enforce aggregate tensor budgets. HPC and desktop workflows separate trusted internal files from imports, pin protected plugin directories, and make any external-resource opt-in explicit. Bindings must expose these controls through APIs rather than environment variables alone.

Viewers, model loaders, upload services, HPC workflows, desktop applications, bindings, and distributors apply this set differently, but the common rule is the same: isolate unknown files, disable unneeded authority, enforce resource budgets before allocation, and publish the exact HDF5 boundary in use. The authoritative owner, work package, effort, and acceptance evidence are in Appendices B and C.

6. HDF5 in the Software Supply Chain

This chapter examines HDF5's position in the software supply chain from two directions: what an HDF5 build itself depends on, and what depends on HDF5. Both matter to the assessment for the same reason — a defect, a configuration difference, or a compromise at either end reaches far beyond the project that introduced it.

Two release-integrity mechanisms frame the discussion. Source-artifact integrity checking, applied to the HDF5 2.0.0 release artifacts [85], allows a consumer to establish that the source it builds is the source that was published. A software bill of materials extends the same idea to what a binary contains, and is the input the `RM-1A` census needs in order to answer whether a given deployment is affected by a given finding. Neither is currently a routine, verifiable property of every HDF5 release artifact; `CORE-S11` and `SUP-S1` propose making them so.

6.1. Upstream Dependencies

This subsection summarizes external software that can be required by an HDF5 build. It starts with the default serial C library and is organized by build option, because the default serial C library has a much smaller dependency set than feature-complete, testing, packaging, or developer builds. More details are provided in Appendix E.

HDF5's CMake build can either locate dependencies that are already installed or, for selected filter-related projects, add external source trees to the build with `FetchContent`. See table 41 for details.

The baseline HDF5 build has no mandatory external dependencies. The core library and C API tests can be built and run without any additional software. See table 42 for details.

The HDF5 build system has options that trigger additional dependencies when enabled. For example, enabling parallel HDF5 requires an MPI implementation,

while enabling Fortran support requires a Fortran compiler. See table 43 for details.

Enabling filter plugins or the plugin project can introduce additional filter-library dependencies. The HDF5 cache defaults include entries for bitgroom, bitround, bitshuffle, Blosc, Blosc2, bzip2, fpzip, JPEG, LZ4, LZF, SZ, ZFP, and Zstandard. Those libraries are controlled by the plugin project's own options when `hdf5_plugins` is built or found. See table 44 for details.

Enabling VFDs for ROS3, HDFS, or other storage systems can introduce dependencies on external libraries and credentials. The same applies to VOL connectors. These are not filter dependencies, but they can affect the security and robustness of HDF5 file access, especially for third-party data. See table 45 for details.

Additional dependencies are required for optional tools, tests, and security features. For example, the `hdf5_security` target depends on OpenSSL when built with `HDF5_ENABLE_SECURITY_FEATURES`. See table 46 for details.

Finally, developer builds and packaging scripts have their own dependencies. These include code linters, formatters, static analyzers, documentation tools, and packaging tools. These dependencies are not required for a default HDF5 build, but they are relevant for the supply-chain risk of development and release processes. For example, if a security vulnerability is introduced in a packager or source formatter, it could affect the integrity of HDF5 releases even if the vulnerability is not present in the HDF5 source code itself. See table 47 for details.

Several default upstream source locations are encoded in `config/CacheURLs.cmake` for dependencies that HDF5 can fetch or whose values are passed to external filter-plugin builds. See table 48 for details.

6.2. Downstream Dependencies

For this audit, downstream dependencies are projects that compile against, dynamically load, package, or define data conventions on top of HDF5. The HDF5 ecosystem is much larger than any static list, so representative downstreams were

selected for broad installation footprint, long-lived archival data, domain-standard file conventions, or use as an intermediate layer for other tools. These are sampling criteria, not formal risk ranks. The following categories are especially relevant to the assessment scope.

Scientific data models and exchange formats.
A prominent downstream is netCDF-4, which uses HDF5 as the storage layer for the enhanced netCDF data model and is central to climate, weather, ocean, atmospheric, and other Earth-system data workflows [92]. NASA HDF-EOS5, NeXus, and CGNS similarly define domain-specific conventions over HDF5 for Earth observation, neutron/x-ray/muon facilities, and CFD or aerospace simulation data [55, 74, 24]. For these formats, HDF5 compatibility is not just a library question: it affects the readability of long-lived archives and the ability of independent tools to interpret a shared schema.

Language bindings and analysis libraries.
h5py, PyTables, pandas HDFStore, Bioconductor's rhdf5, HDF5.jl, and MATLAB's HDF5 interfaces make HDF5 a routine dependency for Python, R, Julia, and MATLAB users [54, 77, 76, 21, 61, 71]. These packages are important because they often sit at the boundary between upstream HDF5 builds and end-user scripts. Their packaging choices determine whether users load a bundled HDF5, a system HDF5, or an environment-provided HDF5, and therefore whether ABI, plugin-path, thread-safety, MPI, and filter support match the files being opened.

Geospatial, visualization, and inspection tools.
GDAL's HDF5 driver, ParaView/VTK-family builds, ITK's HDF5 module, HDFView, and similar readers expose HDF5 files to interactive and automated analysis workflows [75, 65, 60]. This is an important security boundary: these tools are commonly used to inspect third-party data files, so robustness against malformed HDF5 objects, unusual filters, and unexpected VFD/plugin configuration has direct operational value.

HPC and parallel-I/O frameworks.
ADIOS2 includes an HDF5 engine, while netCDF-4, CGNS, h5py, and many simulation applications can be built against serial or parallel HDF5 variants [3].

These downstreams amplify configuration risk: MPI ABI compatibility, collective I/O semantics, file locking behavior, subfiling support, and optional VFDs must line up across the application, HDF5, MPI, and filesystem stack.

Machine-learning and model-artifact workflows.
TensorFlow/Keras still supports older HDF5 model and weight files even as newer native formats are preferred [80]. This keeps HDF5 in the model-exchange and training-checkpoint ecosystem, usually through h5py, and makes HDF5 parser behavior relevant for model files obtained from external sources.

Table 26 – Representative downstream projects that depend on HDF5.

Downstream	Role	Supply-chain relevance
netCDF-4	Common scientific data model and format layered on HDF5.	Large transitive footprint through climate, weather, ocean, remote-sensing, and HPC data stacks. Uses a constrained HDF5 subset, so compatibility depends on both HDF5 behavior and netCDF conventions.
HDF-EOS5	NASA Earth-observation format built on HDF5 concepts and libraries.	Long-lived public archives and many third-party readers make backward compatibility and robust metadata handling high-value.
NeXus	Facility data convention for neutron, x-ray, and muon science.	Uses HDF5 groups, datasets, and attributes as a standardized experimental-data container, so HDF5 schema interpretation affects reproducibility across facilities.
CGNS	CFD and aerospace data standard with an HDF5 mapping.	Important for mesh and simulation-result exchange; parallel-HDF5 behavior and large-file support affect production simulation workflows.
h5py, PyTables, and pandas HDFStore	Python interfaces and tabular/array storage layers.	Widely used bridge between binary HDF5 files and Python analysis. Wheel and conda packaging can hide which HDF5 build is actually loaded.

rhdf5, HDF5.jl, MATLAB HDF5 APIs	R, Julia, and MATLAB access layers.	Expose HDF5 to laboratory and scientific-computing workflows where files are often exchanged outside controlled build environments.
GDAL and geospatial readers	Raster and subdataset access for HDF5-based products.	Converts HDF5 issues into geospatial ingestion issues, especially for remote-sensing products with complex metadata and nested subdatasets.
ParaView/VTK, ITK, HDFView, and viewers	Interactive inspection, visualization, and conversion.	May open untrusted or externally generated files; defensive parsing, filter availability, and plugin search paths matter.
ADIOS2 and simulation I/O layers	Alternative I/O stacks with HDF5 read/write engines or compatibility paths.	Parallel and high-throughput workflows depend on matching MPI, filesystem, and HDF5 feature assumptions across the whole stack.
TensorFlow/Keras HDF5 artifacts	Legacy model and weight serialization.	Keeps HDF5 in ML artifact exchange; malformed or incompatible model files surface through h5py and HDF5 library behavior.

6.3. Audit Findings and Mitigation Planning

The build-and-feature mismatch theme is recorded as evidence-backlog item `SUP-SAFE-001` in Appendix A. It is not a formal finding pending a bounded affected population and evidence sufficient to assign exposure. The bullets below describe ecosystem conditions and potential failure mechanisms; they are not additional formally rated findings.

Priority reconciliation.
Ecosystem reach and transitive blast radius justify the census in `RM-1A`, but do not assign `SUP-SAFE-001` a tier or response urgency. The roadmap schedules a bounded producer/consumer mismatch study in `RM-2F`; a supported-profile compatibility and provenance program follows in `RM-3E` and `RM-4A`. If the study substantiates a concrete affected population and harm, the resulting finding

is rated and routed under the report-wide rubric rather than inheriting a priority from this chapter's project list.

- HDF5 changes have a large transitive blast radius because many users reach HDF5 through netCDF, GDAL, h5py, MATLAB, or domain formats rather than through the HDF5 C API directly.
- ABI and packaging mismatches are a plausible downstream mismatch mechanism; this assessment did not establish their frequency or affected population. Python, R, Julia, MATLAB, MPI, and system-package environments may all load different HDF5 builds in the same organization.
- File parsing is a practical security boundary. Viewers, converters, notebooks, and data-ingestion jobs often process files received from outside the organization that built the HDF5 library.
- Optional features can become hidden interoperability requirements: filter plugins, SZIP/zlib variants, ROS3/HDFS/VFD support, parallel HDF5, and file-locking behavior may be assumed by downstream data even when they are not present in a default HDF5 build.
- Long-lived scientific archives make deprecation slower than ordinary application software. Even when a downstream project prefers a newer format, historic HDF5 files remain operationally relevant.

The following identifier names a recommendation only; it does not assign risk, urgency, effort, dependency, or implementation order.

Recommendation SUP-S1: Maintain a producer/consumer profile compatibility and migration matrix

For every supported producer, reader, binding, and packaging profile, record the HDF5 version and linkage, ABI, enabled filters and plugins, VOL/VFD features, parallel and MPI assumptions, file-locking policy, and format bounds. Preserve representative artifacts and state the expected accept or controlled-reject result for each supported pair. When a mismatch is discovered, inventory the affected artifacts and publish a deterministic repair, rebuild, migration, or documented

compatibility path rather than relying on the fact that one older environment happened to accept the file.

Acceptance requires a finite, versioned supported-profile catalog, matrix, and fixture set in CI; diagnostics that identify the mismatched capability; and a tested deterministic disposition for every incompatible supported pair: repair, rebuild, migration, controlled rejection with a supported alternative, or an accountable exception. Release SBOM and feature reporting under `CORE-S11` and `APP-S6` supply provenance inputs but do not, by themselves, demonstrate compatibility or treatment of legacy artifacts.

Maintaining the supported-profile catalog and CI matrix is a proactive assurance control; it does not by itself create or close a formal finding. Mismatch-specific repair, rebuild, migration, controlled rejection, or exception is selected only after `RM-2F` bounds the affected profile population and consequence.

7. Independent HDF5 Implementations

7.1. Overview

PureHDF Managed .NET read/write without native binaries.

jHDF JVM HDF5 reading.

netCDF-Java JVM netCDF/HDF-EOS/HDF5 scientific-data reading.

pyfive Pure Python read-only inspection.

jsfive Browser-side direct HDF5 reading.

scigolib/hdf5 Pure Go read/write.

hdf5-pure and rustyhdf5-format/rustyhdf5 Pure Rust read/write.

async-hdf5 or h5coro Cloud object-store chunk mapping.

JLD2.jl Julia-native persistence in HDF5-compatible files.

h5lens / H5Lens Machine-readable format definitions with an independent reader, validator, and preflight oracle. Distinct from the others in that its interpretation of the format is *inspectable as data* rather than inferable from behaviour.

Vocabulary. This chapter uses the issue families defined by the SSP SIG's HDF5 Independent Implementation (H5II) model [42]: **I1** parser/spec divergence, **I2** fail-open compatibility behaviour, **I3** writer correctness and interoperability failure, **I4** codec, filter, or external-reference boundary failure, **I5** numeric, structural, or resource amplification, **I6** concurrency, durability, or lifecycle mismatch, **I7** metadata and artifact exposure, **I8** dependency, provenance, or compatibility-claim drift, and **I9** trust-boundary confusion. As elsewhere, these labels classify mechanisms; they are not risk ratings, and they do not carry the SSP SIG's numeric scores into Section 1.8.

The model's central observation frames this chapter: an independent implementation removes a dependency on the reference library but not the hard problem,

because the trusted computing base becomes the implementation's own parser, validator, writer, and support libraries. Its working assumption that *partial support is normal, but unsupported or unknown features must fail closed rather than being silently ignored or guessed* is the specific standard against which the two 2026 cases below are best read.

Why multiple implementations matter. Independent implementations create observable comparisons between the prose specification, the reference library, and separately developed readers and writers. A difference does not by itself identify which artifact is wrong, but it exposes a question that a single implementation may leave hidden until files cross an implementation boundary. The consequence can then present as a reader or writer failure even when the underlying problem is a reference-library divergence or an unresolved normative ambiguity.

The two relevant 2026 issues illustrate different outcomes and must not be used interchangeably. In issue #6409 the published requirement was sufficiently clear to decide the dispute: the reference reader imposed an extra condition and was changed. In issue #6616 the published optionality and reference-library behavior still admit competing interpretations, and no authoritative ruling has selected one. Only the latter unresolved adjudication and governance condition is the bounded observation recorded as evidence-backlog item `FMT-SAF-001`. The former remains evidence for the separately registered implementation divergence `IND-SAF-001`.

Independent readers and writers are valuable checks on both the prose specification and the reference implementation. They also expose a three-way distinction essential to interoperability analysis: a file can be accepted by one reader, conform to or violate the normative format independently of that acceptance, and remain safe or unsafe for a particular deployment. Reader acceptance alone is not conformance evidence.

7.2. Two Distinct 2026 Chunk-Layout Cases

The two cases below produced superficially similar upgrade failures, but they must not be combined. One involved a reader imposing a stronger condition than the format required; the other involved contradictory writer output that older readers had tolerated.

7.2.1. Sufficient but Nonminimal Encoding

HDF5 issue #6409 concerns version-4 chunk-layout dimension widths. The format required an encoded width sufficient to hold the dimension value, but did not require the smallest possible width. The exact-width check present in HDF5 2.0.0, 2.1.0, and 2.1.1 could therefore reject a specification-permitted, nonminimal encoding produced by an independent writer. Stated plainly, and this is the part the interoperability framing tends to soften: **the reference library was non-conforming**, the specification was correct, and it took an independent implementation to establish that. The merged HDF5-side change accepts any sufficient width and was tracked against 2.2.0 [4, 29]. PureHDF separately changed new output to match the reference reader's expectation [22, 66] — a rational local decision with a corrosive global effect, since converging on the reference implementation rather than on the specification is how a specification quietly stops being normative. The conformance answer in this case was nevertheless determinable. Formal finding `IND-SAF-001` records the historical reader-side interoperability risk; issue closure is not, by itself, verification of every release artifact or backport. In H5II terms this is **I1**, parser/spec divergence — and note that it is divergence by the *reference* reader, which is the direction the model's framing makes it easy to miss.

7.2.2. Contradictory PureHDF Metadata

HDF5 issue #6534 began as a suspected HDF5 2.0 regression because the same production-derived file opened with HDF5 1.10.10 and 1.14.6 but failed with HDF5 2.0. Maintainer byte-level analysis identified a different condition in the

affected PureHDF 2.1.2 output: float64 datatype metadata declared an eight-byte element while the version-4 chunk-layout message recorded a four-byte stored element. The file was internally inconsistent; the older readers had been permissive, and the HDF5 2.0 rejection was correct [2, 1].

The reporter described a continuously written production acquisition pipeline and a reader-version pin as the contemporaneous workaround. A technical metadata-repair path was proposed and older readers remained a workaround, although the reporter said rewriting the files was not operationally feasible in that pipeline. The pin restored availability but was neither a conformance repair nor the report's Immediate urgency category. The evidence therefore supports bounded operational and archival harm, not irreversible data loss. Formal finding `IND-SAF-002` records this writer-side condition. Existing affected files remain a migration and inventory question even after corrected writer output is deployed. In H5II terms it is **I3**, writer correctness and interoperability failure, and the older readers' acceptance of the file is **I2**, fail-open compatibility behaviour — the pair is the model's canonical chain, where a writer emits bytes that are only self-consistent to itself and a permissive reader conceals it until a stricter one does not.

7.3. When a Normative Ambiguity Remains Unresolved

The two chunk-layout cases above have determinable answers: once examined, the published requirement and the bytes were sufficient to identify whether the reader or writer was wrong. A third case shows that a divergence can instead remain unresolved because the normative text and reference behavior support competing dispositions.

HDF5 issue #6616 records that the file-format specification documents the group info message as *optional*, while the C library raises a "message type not found" error when it is absent and consequently cannot modify a file written without one. The reporter — again an independent-implementation maintainer, again working from a PureHDF change — sets out two candidate resolutions: the library should fall back to default group-info values, or the specification should be corrected to

state that the message is required. At the evidence cutoff the issue is open, labelled a documentation bug, assigned, and milestoned, with no ruling recorded [5].

Until that ruling exists, **this report cannot determine whether omission of the group info message conforms when the reference library is asked to modify the file**. The bounded observation is the unresolved dispute and the demonstrated library behavior, not a conclusion that the whole format is indeterminate. This is also a limitation of the report's method: the conformance analysis in Section 1.7 distinguishes the normative requirement from writer behaviour and reader validation, and here the authority needed to settle that distinction has not yet ruled. The missing control is therefore both an artifact that makes competing interpretations testable and a procedure that states who rules, on what timeline, which outcomes are permitted, and where the ruling is recorded. A machine-readable description could expose the question earlier; it would not select the answer without an adjudication decision.

The bounded #6616 adjudication and governance gap is recorded as evidence-backlog item `FMT-SAF-001` in Appendix A and addressed by `FMT-S1` below. It does not receive a tier until an affected population and adverse consequence are bounded.

Disposition matrix. Table 27 is the compact decision record for the three cases. It prevents two unsafe shortcuts: treating acceptance by an older reader as proof of conformance, or treating the reference library's behavior as more authoritative than a clear normative requirement.

Table 27 – PureHDF-related interoperability dispositions

Case	Disposition at the evidence cutoff	What acceptance or rejection establishes	Required control and closure evidence
#6409 `IND-SAF-001`	The nonminimal chunk-width encoding was permitted. The reference reader added an exact-width condition and was corrected; PureHDF also changed new output for compatibility [4, 29, 22, 66].	Rejection established a reader/specification divergence, not malformed writer output. Choosing one canonical writer form cannot narrow what readers must accept.	Keep the nonminimal form as a positive fixture. Verify the exact supported release and backport containing the reader correction; separately test the producer's canonical output.
#6534 `IND-SAF-002`	PureHDF 2.1.2 emitted contradictory element sizes. HDF5 2.0 correctly rejected the file; older readers had tolerated it [2, 1].	Earlier acceptance did not establish conformance, and later rejection did not establish a regression. Permissiveness concealed the writer defect until an upgrade exposed it.	Keep the file as a negative fixture; enforce cross-message invariants; inventory the bounded legacy population; and provide reproducible repair, regeneration, migration, or an accountable compatibility disposition.
#6616 `FMT-SAF-001`	The specification calls the group-info message optional, while the C library cannot modify a file when it is absent. Two resolutions were proposed, but no authoritative ruling was recorded [5].	The observed divergence establishes a disputed rule, not which artifact is wrong, a class-wide conformance failure, or a bounded risk tier.	Keep a contested fixture under both interpretations. The named format authority must rule, update the losing artifact, and record any explicitly implementation-defined outcome and compatibility consequence.

A canonical producer profile may select minimal widths, mutually consistent metadata, and — pending adjudication — a group-info message to reduce cross-reader variation. It remains an interoperability subset, not a replacement for the normative format: it cannot make the #6409 form invalid, make the #6534 contradiction harmless, or decide #6616 without the responsible authority. This is why `IND-S1` needs separate positive, negative, and contested fixtures and why only the third case depends on `FMT-S1`'s adjudication procedure.

7.4. Audit Findings and Mitigation Planning

The main interoperability challenge across direct-format implementations is not opening simple files; it is coverage of the long tail: dense groups, fractal heaps, shared object-header messages, chunk-index variants, references, VLEN, compound and nested datatypes, virtual datasets, external storage, SWMR flags, user blocks, non-core filters, unusual library-version bounds, and files produced by different HDF5 releases. This architectural observation is not assigned a risk tier without a concrete affected workflow. The H5II model's `II-##` review register is the natural instrument for working through that long tail feature by feature, and `IND-S1` below supplies the fixtures it would need. The two bounded chunk-layout findings are `IND-SAF-001` and `IND-SAF-002` in Appendix A. The unresolved group-info-message dispute is separately recorded in the evidence backlog as `FMT-SAF-001`; it is not a class-level rating assigned to every implementation divergence.

Priority reconciliation.
`FMT-SAF-001` has no inherent-risk response urgency. It has a 90-day administrative assurance target: rule on the #6616 dispute, record the ruling, and change the artifact that the ruling identifies or document the construct as implementation-defined with explicit compatibility behavior. That bounded action has a named owner and is not contingent on completing `FMT-S1`; the 90-day target does not imply a risk tier. `IND-SAF-001` and `IND-SAF-002` are Moderate and follow the Planned route; neither is ordered ahead of the other by issue chronology or recommendation numbering. Phase 2 therefore verifies the exact supported reader artifacts, preserves positive and negative fixtures, inventories the affected legacy PureHDF population, and provides a deterministic repair, migration, or compatibility disposition. Invariant and sustainment work continues in Phases 3–4. A production or archival population discovered to be actively interrupted may be expedited, but that scoped decision does not revise the historical inherent ratings.

The `FMT-S` and `IND-S` identifiers below identify recommendations only; their numbers do not express risk, urgency, effort, dependency, or implementation order.

Recommendation FMT-S1: Make the specification executable and adjudicable

For a declared format subset, maintain a machine-readable description that generates valid, invalid, and contested fixtures, and adopt a standing dispute procedure naming the decision authority, deadline, permitted dispositions, and authoritative record. A disposition must identify a library defect, a description/specification defect, or an explicitly implementation-defined construct with documented compatibility consequences. A producer change made without that ruling does not close the normative question.

The candidate `h5lens` foundation is GPLv3 while HDF5 is BSD-style; reuse therefore requires the provenance and licensing review in Section C.3, and generated artifacts become normative only through an explicit governance decision [30].

Recommendation IND-S1: Maintain a differential conformance corpus

Preserve positive, negative, and contested fixtures across current and supported older readers, recording writer/reader versions, format bounds, and expected outcomes. Keep #6409 (permitted nonminimal width), #6534 (contradictory element sizes), and #6616 (unresolved group-info optionality) as distinct cases. Acceptance requires supported readers to accept the valid fixtures, reject invalid ones with controlled diagnostics, and provide a reproducible repair, regeneration, migration, or accountable compatibility disposition for the complete defined legacy PureHDF population, with provenance preserved when bytes change.

The authoritative package, owner, clock, and closure evidence for both recommendations are in Appendix C.

8. Long-term Data Accessibility and Interpretability

Long-term accessibility depends on more than preservation of the primary file bytes. It can also depend on format bounds, required filters and extensions, external resources, application schema and provenance, and the continued availability of readers that implement the relevant interpretation. A file whose bytes are perfectly preserved and whose required filter no longer exists has been archived in the same sense that a sealed envelope with no known language inside has been archived.

8.1. How Extension Dependencies Affect Preservation

HDF5 extension mechanisms can make interpretation depend on software or storage outside the preserved bytes, but the evidence is not equivalent across the three classes. Missing filter plugins have documented short-horizon access failures. The VFD and VOL observations below are architectural extrapolations that identify questions for further study; they are not included in the formal rated scenario in this revision.

Filter plugins.
A filtered dataset records its filter by numeric identifier, so the dependency is at least *discoverable* from the file, and a reader can in principle re-implement a sufficiently specified codec. This is the evidenced failure mode: users report being unable to read data because a required filter was missing from their build or unavailable on their platform, across a fifteen-year span and several filter families [19, 13, 10, 14]. Registered filter identifiers help; an unregistered or private identifier removes even discoverability.

Virtual file drivers.
A VFD determines whether the bytes can be reached at all. If a file was written through a driver with a non-trivial storage mapping, preserving "the file" may not preserve every object needed to reconstruct the mapping. This assessment did not identify a bounded archive population or a realized VFD-dependent preservation failure, so this remains an unrated extension of the analysis.

VOL connectors.

A VOL connector can service the HDF5 API from a store that is not an HDF5 file. The preservation object may therefore include an external store, connector semantics, and configuration rather than a self-contained HDF5 file. This report did not identify a bounded population, realized loss, or preservation horizon for that condition. It is an unrated evidence question, not part of LONG-SAF-001.

Community discussion has long recorded the operational importance of preserving plugin and reader knowledge alongside archived data [7]. For this revision, LONG-SAF-001 is limited to archived files that require a non-core filter which is unavailable to the later reader. The broader VFD and VOL questions should be carried into a future evidence backlog with a defined archive population, preservation object, dependency inventory, and measured read outcome before any rating is assigned. This assessment did not inventory an archive or test a representative preservation horizon, which is why the formal missing-filter scenario remains Provisional with Low confidence.

8.2. Audit Findings and Mitigation Planning

The evidenced missing-filter scenario is registered as LONG-SAF-001 in Appendix A, rated **I2/E2, Moderate, Provisional, Low confidence** at a stated assessment horizon of ten years or more. Because that horizon is material, decision rule 7 of Section 1.8 forbids ordering this rating directly against ratings that state no horizon. The VFD and VOL observations above have no impact or exposure axes, tier, confidence, or response urgency in this chapter.

The bounded reader/writer compatibility cases in Section 7, IND-SAF-001 and IND-SAF-002, remain separately registered. They demonstrate a related consequence — loss of access when reader expectations change — but arise from different mechanisms and require different controls. They are not evidence for the frequency or horizon of filter-plugin unavailability and are not used to rate LONG-SAF-001.

Priority reconciliation.
LONG-SAF-001 follows the default *Planned* route for its Moderate rating. Within 90 days, RM-2H defines a bounded corpus, inventories each file's filter dependencies, and records an accountable availability, reproducibility, migration, or acceptance disposition. The longer structural work remains effective at *write and deposit* time: once an archive holds a file whose required filter is unavailable, treatment may require recovering or rebuilding the filter, re-implementing it from a specification, or migrating the data with an older environment. If an archive is already failing, that present-tense instance is separately scoped and routed without the horizon qualifier.

The following identifier names a recommendation only; its number expresses no risk, urgency, effort, dependency, or implementation order.

Recommendation LONG-S1: Make extension dependencies discoverable, then removable

For a defined archival corpus, generate one machine-readable report of filters, format bounds, external resources, and relevant VFD/VOL dependencies. Prefer a tested core-only or widely specified archival copy; where a filter cannot be removed, preserve its identifier, specification, test vectors, source/build provenance, and a reproducible environment. A deposit must also say whether it is a self-contained HDF5 file or a view onto an external system.

Acceptance for LONG-SAF-001 requires every file in the defined corpus to have a complete dependency report; the archival profile to open and read with all non-core plugins disabled; and every retained filter to be independently reproducible from preserved evidence at least once. VFD/VOL inventory results remain evidence only until a separate bounded study establishes consequence and population. Owners, work packages, and closure records are in Appendix C.

9. Conclusion

9.1. Overall Assessment Conclusion

Within the engagement scope and the August 27, 2026 evidence cutoff, this assessment supports a bounded conclusion: HDF5 input is not necessarily inert data. File-controlled metadata can reach memory, allocation, traversal, external-resource, plugin, deserialization, storage, and disclosure boundaries. Risk arises at the component that supplies and controls the relevant authority, and not every appearance of HDF5 in an adverse event is therefore a defect in the core library or file format.

The evidence supports four ecosystem-level judgments. First, historically affected core and official-tool paths exhibit recurring failures to validate file-derived values before trusted C operations; local fixes can treat a known variant, but class elimination requires invariant-driven prevention and continuing regression evidence. Second, applications can turn documented HDF5 capabilities into security mechanisms when they process untrusted files with ambient filesystem, plugin-loading, deserialization, or resource authority. Third, reader acceptance, format conformance, and safe deployment are separate decisions: permissive historical acceptance is not proof that a file is consistent, and stricter rejection is not automatically a regression — and in at least one documented case the reference library, not the independent writer, was the non-conforming party, while in another it is still undetermined which artifact is wrong. Fourth, privacy harm can arise during correct and authorized operation because payload protection does not necessarily protect metadata, structure, derived information, caches, logs, or other workflow artifacts. These four judgments do not exhaust the safety surface: storage exhaustion, abrupt termination, and filter-dependent loss of interpretability also threaten data that no adversary ever touched. This revision registers those bounded scenarios while leaving unmeasured VFD, VOL, and specification-governance questions in the evidence backlog.

The registered scenarios have a determinate, conditional mitigation-priority order, and confirmed applicable scope follows it. No exception permits a known, uncontained Critical-equivalent external-resource path to process untrusted input; any other variance requires a scoped, accountable, expiring record and tested interim control. The rubric and response routing control—not chapter order, recommendation number, CVE count, publicity, effort, or dependency. This is neither a universal HDF5 rating nor a certification or conclusion about an unassessed release or deployment.

The authoritative counts and rating bases are maintained in Appendix A; they describe bounded scenarios, not prevalence or a technology-wide score.

9.2. Reconciled Mitigation Priorities

Appendix A supplies the authoritative response routing. A finding-response clock starts when affected scope is identified; the deployment census and assert-site catalog are due within 30 days of roadmap adoption. Scope discovery is not treatment, and prevention work does not defer containment.

Immediate. Before the next untrusted job and within 24 hours of identification, stop or sandbox any equivalent `APP-SEC-001` external-resource path. This is conditional on a still-exposed deployment, not a rating of every HDF5 consumer or Hugging Face's present state.

Near-term. Within 30 days of identifying affected scope, treat the High and upper-bound-High core, tool, application, privacy, and write-side findings listed in the register. For `CORE-SAFE-001` and `CORE-SAFE-002`, `RM-1F` operational containment—not the `RM-1A` census or a future library change—is the time-bound action.

Planned. Within 90 days or the next supported release or migration window, whichever is sooner, bound resource use and resolve the registered interoperability and filter-dependent preservation scenarios.

Evidence action. `CORE-SEC-003`, `FMT-SAF-001`, `EXT-PRIV-001`, and `SUP-SAFE-001` retain administrative evidence schedules but no assigned inherent-risk urgency. Backlog status is not Low risk.

Continuing prevention—invariant-driven parsing, release-build checks, differential conformance, governed extensions, privacy-lifecycle controls, and release gates—runs in parallel and never changes these treatment clocks.

The chapter analyses supply evidence; Appendix C translates these lanes into accountable work packages without changing inherent risk. No chapter order or recommendation number creates priority.

9.3. Significance of the 2026 Developments

The Hugging Face incident and HDF5 issue #6618 sharpen the application-boundary conclusion. The incident demonstrates that a valid, documented external-storage capability can become a repeatable disclosure mechanism when an automatic service supplies filesystem authority on behalf of an untrusted file. That evidence supports Immediate containment of equivalent deployments and the need for unified, enforcement-coupled external-resource policy. It does not support reclassifying the mechanism as a core parser vulnerability, treating a policy proposal as an implemented control, or assigning the incident's Critical tier to every HDF5 consumer [67, 90, 25, 78].

The PureHDF case sharpens the safety and interoperability conclusion in a different way. Older-reader acceptance of contradictory chunk metadata did not make the file conforming, while the separate sufficient-but-nonminimal case shows that stricter validation can itself reject a permitted encoding. Together the cases require a normative invariant decision, paired positive and negative fixtures, exact supported-reader verification, an inventory of affected legacy files, and deterministic repair or migration. The correct response is neither permissiveness by default nor strictness without compatibility evidence [4, 29, 2, 1].

HDF5 Pickles issue #87 shows why a predicate derived from file bytes is not a release control when it exists only inside `assert(...)` and `-DNDEBUG`

removes it. Its seven component families and cursory 50–100 estimate define an audit perimeter, not a vulnerability count. One source-reported v2 B-tree case supplies a concrete `SIGFPE` fixture on a release-configured HDF5 2.3.0 development tree, but does not establish 32-bit behavior, supported-release scope, or a version range [57, 33]. It therefore sharpens the 30-day catalog and paired-build test action without acquiring a risk tier.

These developments validate rather than override the rubric: the Hugging Face case supports a conditional Immediate application-boundary response; the two independent-implementation cases support Planned Moderate safety treatment; issue #6616 requires adjudication without inherent-risk urgency; and the assert observation remains an evidence action until its supported population and consequence are established.

9.4. Assurance, Closure, and Residual Risk

No mitigation or residual-risk reduction is verified at publication. Unless a record says otherwise, evidence came from public document and artifact review, not independent reproduction; installed populations, supported-release fix coverage, and deployment control effectiveness remain incomplete. Every work package is therefore *Proposed / implementation not verified*.

Priority becomes auditable only after the roadmap authority defines the census boundary, assigns owners and dates, and preserves finding-by-scope response records. Local treatment may close a scoped scenario after independent verification and an accountable residual-risk disposition, while recurrence prevention remains open until its separate evidence passes. For High and Critical scope, the verifier must be separate from the package owner, rollout approver, and risk acceptor.

The practical conclusion is consequently twofold. Known affected populations should be treated on their response clocks without waiting for architectural work, and HDF5's recurring classes should be addressed structurally without pretending that local closure proves class elimination. Applied together, those tracks convert the report from a catalog of concerns into a testable mitigation program: constrain

file-selected capabilities at authority boundaries, validate file-derived state before trusted object construction, decide compatibility against normative evidence, and govern privacy across the full data lifecycle.

References

[1] Abbo93. *Compatibility Warning: HDF5 2.0.0 / h5py 3.16.0 Regression Breaks Reading PureHDF Files*. 2026. URL. (accessed: 27.08.2026).

[2] Abbo93. *Regression in 2.0.0 Chunk Layout Decoder Fails on Valid Files from Third-Party Writers (PureHDF)*. 2026. URL. (accessed: 27.08.2026).

[3] ADIOS2 Developers. *ADIOS2 Supported Engines: HDF5*. 2026. URL. (accessed: 29.05.2026).

[4] Apollo3zehn. *HDF5 lib is more restrictive than file format spec*. 2026. URL. (accessed: 27.08.2026).

[5] Apollo3zehn (PureHDF maintainer). *Group info message is documented as optional but the C library requires it*. 2026. URL. (reported: 16.08.2026; open, labelled documentation bug, milestone 2.3.0; accessed: 30.08.2026).

[6] Apple. *CVE-2021-31009*. 2021. URL. (accessed: 04.06.2026).

[7] Multiple authors. *223 results for 'plugin'*. 2007-2026. URL. (accessed: 21.05.2026).

[8] Multiple authors. *Avoiding a corrupted HDF5 file or being able to recover it*. 2019. URL. (accessed: 30.08.2026).

[9] Multiple authors. *Crashproofing HDF5*. 2024. URL. (accessed: 30.08.2026).

[10] Multiple authors. *Dynamic filters on Windows*. 2017. URL. (accessed: 30.08.2026).

[11] Multiple authors. *File state after flush and crash*. 2015. URL. (accessed: 30.08.2026).

[12] Multiple authors. *Flushing files sometimes corrupts them*. 2018. URL. (accessed: 30.08.2026).

[13] Multiple authors. *H5Dump dataset retrieve error (compressed datasets)*. 2015. URL. (accessed: 30.08.2026).

[14] Multiple authors. *h5dump error: unable to print data (missing LZO filter)*. 2017. URL. (accessed: 30.08.2026).

[15] Multiple authors. *HDF5 data corruption issues after a crash?* 2020. URL. (accessed: 30.08.2026).

[16] Multiple authors. *HDF5 file state in case of crash*. 2010. URL. (accessed: 30.08.2026).

[17] Multiple authors. *Preventing file corruption on crash*. 2012. URL. (accessed: 30.08.2026).

[18] Multiple authors. *Recover a corrupt HDF5 file*. 2008. URL. (accessed: 30.08.2026).

[19] Multiple authors. *Troubles with compressed data*. 2012. URL. (accessed: 30.08.2026).

[20] Andreas Beckmann and Gerd Heber. *How to properly close HDF5 files when disk is full? — design discussion*. 2023–2026. URL. (accessed: 30.08.2026).

[21] Bioconductor Project. *rhdf5*. 2026. URL. (accessed: 29.05.2026).

[22] Blackclaws. *Chunked Datasets Are Unreadable by HDF5 2.x: Chunk Dimension Size Encoded Length Is Hardcoded*. 2026. URL. (accessed: 27.08.2026).

[23] Sean Brooks et al. *An Introduction to Privacy Engineering and Risk Management in Federal Systems*. NISTIR 8062. Gaithersburg, MD: National Institute of Standards and Technology, Jan. 2017. DOI: 10.6028/NIST.IR.8062. URL.

[24] CGNS Steering Committee. *CGNS/HDF5 – An HPC implementation*. 2026. URL. (accessed: 29.05.2026).

[25] chinhnguyen007. *Local File Access Through Untrusted HDF5 External Dataset References*. 2026. URL. (accessed: 27.08.2026).

[26] Cisco Talos. *HDF5 Group libhdf5 H5T_ARRAY Code Execution Vulnerability*. 2016. URL. (accessed: 27.08.2026).

[27] CVE Project. *CVE-2021-36977*. 2021. URL. (accessed: 04.06.2026).

[28] CVE Project. *CVE-2026-34734*. 2026. URL. (accessed: 04.06.2026).

[29] Neil Fortner. *Allow Reading of Files with Chunk Dimensions Encoded Using More Bytes than Necessary*. 2026. URL. (accessed: 27.08.2026; merged pull request).

[30] The HDF Group. *A machine-readable specification of the HDF5 file format*. 2026. URL. (accessed: 21.05.2026).

[31] The HDF Group. *Case Record: Cache-Image LRU Rank Guarded Only by an Assertion*. 2026. URL. (accessed: 30.08.2026).

[32] The HDF Group. *Case Record: External Data File Opened Before Segment Validation*. 2026. URL. (accessed: 30.08.2026).

[33] The HDF Group. *Case Record: v2 B-tree Record Size Guarded Only by an Assertion*. 2026. URL. (measured and re-measured: 15.08.2026; pinned source state: 25.08.2026; accessed: 30.08.2026).

[34] The HDF Group. *Commit 3914bb7*. Sep 25, 2025. URL. (accessed: 21.05.2026).

[35] The HDF Group. *Commit 804f3bae*. Aug 13, 2025. URL. (accessed: 21.05.2026).

[36] The HDF Group. *Commit ce53bc0*. Mar 31, 2024. URL. (accessed: 21.05.2026).

[37] The HDF Group. *H5Patch: Evidence-Gated Repair Planning, Application, and Audit Logging*. 2026. URL. (accessed: 30.08.2026).

[38] The HDF Group. *H5Policy Invariant Registry, Validation Coverage, and libhdf5 Evidence*. 2026. URL. (accessed: 30.08.2026).

[39] The HDF Group. *H5Policy: Metadata Preflight, Security Profiles, Regression Corpus, and Differential Testing*. 2026. URL. (accessed: 30.08.2026).

[40] The HDF Group. *HDF5 CVE Test Suite*. 2026. URL. (accessed: 21.05.2026).

[41] The HDF Group. *HDF5 extension skeletons, literally*. 2026. URL. (accessed: 21.05.2026).

[42] The HDF Group. *HDF5 Independent Implementation (H5II) Model*. 2026. URL. (accessed: 30.08.2026).

[43] The HDF Group. *HDF5 Plugins*. 2026. URL. (accessed: 21.05.2026).

[44] The HDF Group. *HDF5 Python Wrapper (H5PW) Model*. 2026. URL. (accessed: 30.08.2026).

[45] The HDF Group. *HDF5 Safety Hazard Model (H5CS)*. 2026. URL. (accessed: 30.08.2026).

[46] The HDF Group. *HDF5 Safety, Security & Privacy (SSP) Special Interest Group's governance, processes, and artifacts*. 2026. URL. (accessed: 21.05.2026).

[47] The HDF Group. *HDF5 Security Threat Model*. 2026. URL. (accessed: 30.08.2026).

[48] The HDF Group. *HDF5 SSP SIG Policy Manual and `HDF5-SHINES-*` Control Catalog*. 2026. URL. (accessed: 30.08.2026).

[49] The HDF Group. *HDF5 SSP SIG Privacy Exposure Registry (`EXP-001`–`EXP-010`)*. 2026. URL. (accessed: 30.08.2026).

[50] The HDF Group. *HDF5 SSP SIG Safety Hazard Registry (`HAZ-001`–`HAZ-016`)*. 2026. URL. (accessed: 30.08.2026).

[51] The HDF Group. *HDF5 SSP SIG Security Threat Registry (`ATK-001`–`ATK-013`)*. 2026. URL. (accessed: 30.08.2026).

[52] The HDF Group. *OSPS → SHINES Coverage Matrix*. 2026. URL. (accessed: 30.08.2026).

[53] The HDF Group. *Top Weaknesses (CWE) that Become Vulnerabilities in HDF5 Contexts*. 2026. URL. (accessed: 30.08.2026).

[54] h5py Developers. *HDF5 for Python*. 2026. URL. (accessed: 29.05.2026).

[55] HDF-EOS Tools and Information Center. *HDF-EOS Libraries*. 2026. URL. (accessed: 29.05.2026).

[56] Gerd Heber. *A CVE Strategy for the HDF5 Library*. 2026. URL. (accessed: 30.08.2026).

[57] Gerd Heber. *Catalog assert-masked de-serializer invariants in the HDF5 library source*. 2026. URL. (created: 24.08.2026; accessed: 28.08.2026; open issue).

[58] Gerd Heber. *What is Bounded Raw Decode?* 2026. URL. (accessed: 30.08.2026).

[59] Zero Day Initiative. *CVE-2026-2492*. 2026. URL. (accessed: 21.05.2026).

[60] Insight Software Consortium. *ITK Module: ITKHDF5*. 2026. URL. (accessed: 29.05.2026).

[61] JuliaIO. *HDF5.jl*. 2026. URL. (accessed: 29.05.2026).

[62] Keras Team. *Denial of Service via Malicious HDF5 Shape Metadata in a Keras Model*. 2026. URL. (accessed: 27.08.2026).

[63] Keras Team. *Legacy HDF5 Model Loading Ignores safe_mode=True*. 2025. URL. (accessed: 27.08.2026).

[64] Keras Team. *Local File Disclosure via HDF5 External Storage During Keras Weight Loading*. 2026. URL. (accessed: 27.08.2026).

[65] Kitware, Inc. *Building ParaView*. 2026. URL. (accessed: 29.05.2026).

[66] Mark Lambert. *Make Tests Runnable on Windows and with Newer h5dump, with Related Fixes*. 2026. URL. (accessed: 27.08.2026; merged pull request).

[67] Hugo Larcher et al. *Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident*. 2026. URL. (accessed: 27.08.2026).

[68] Google LLC. *CVE-2025-9905*. 2025. URL. (accessed: 21.05.2026).

[69] Google LLC. *CVE-2026-0897*. 2026. URL. (accessed: 21.05.2026).

[70] Google LLC. *CVE-2026-1669*. 2026. URL. (accessed: 21.05.2026).

[71] MathWorks. *HDF5 Files*. 2026. URL. (accessed: 29.05.2026).

[72] MITRE. *HDF5 CVE: Common Vulnerabilities and Exposures*. 2026. URL. (accessed: 21.05.2026).

[73] National Institute of Standards and Technology. *NIST Privacy Framework: A Tool for Improving Privacy through Enterprise Risk Management, Version 1.0*. Framework. Gaithersburg, MD: National Institute of Standards and Technology, Jan. 2020. URL (visited on 06/25/2026).

[74] NeXus International Advisory Committee. *NeXus Data Format Manual*. 2026. URL. (accessed: 29.05.2026).

[75] Open Source Geospatial Foundation. *HDF5 – Hierarchical Data Format Release 5 (HDF5)*. 2026. URL. (accessed: 29.05.2026).

[76] pandas Developers. *IO tools: HDF5 / PyTables*. 2026. URL. (accessed: 29.05.2026).

[77] PyTables Developers. *PyTables Documentation*. 2026. URL. (accessed: 29.05.2026).

[78] takluyver. *Option(s) to Disable Various Kinds of External File Access*. 2026. URL. (accessed: 27.08.2026).

[79] TensorFlow Authors. *Commit 46e7f7f: Disable Loading HDF5 Plugins*. 2025. URL. (accessed: 27.08.2026).

[80] TensorFlow Authors. *Save, serialize, and export models*. 2026. URL. (accessed: 29.05.2026).

[81] The HDF Group. *H5D__bt2_filt_decode() at d8ec639*. 2026. URL. (source snapshot: 25.08.2026; accessed: 28.08.2026).

[82] The HDF Group. *H5EA__hdr_init() at d8ec639*. 2026. URL. (source snapshot: 25.08.2026; accessed: 28.08.2026).

[83] The HDF Group. *H5HF cache direct-block deserialization at d8ec639*. 2026. URL. (source snapshot: 25.08.2026; accessed: 28.08.2026).

[84] The HDF Group. *H5T__conv_struct Use After Free*. 2026. URL. (accessed: 27.08.2026).

[85] The HDF Group. *HDF5 2.0.0 source-artifact integrity check*. URL. (accessed: 03.05.2026).

[86] The HDF Group. *HDF5 assertion configuration in HDFCompilerFlags.cmake at d8ec639*. 2026. URL. (source snapshot: 25.08.2026; accessed: 28.08.2026).

[87] The HDF Group. *HDF5 Data Model and File Structure*. 2026. URL. (accessed: 04.06.2026).

[88] The HDF Group. *HDF5 Privacy Exposure (H5PE) Model*. 2026. URL. (accessed: 14.07.2026).

[89] The HDF Group. *HDF5 Security Policy*. 2026. URL. (accessed: 29.05.2026).

[90] The HDF Group. *HDF5 Was the Mechanism, Not the Vulnerability*. 2026. URL. (accessed: 27.08.2026).

[91] Trend Micro Zero Day Initiative. *TensorFlow HDF5 Library Uncontrolled Search Path Element Local Privilege Escalation Vulnerability*. 2026. URL. (accessed: 27.08.2026).

[92] Unidata Program Center. *NetCDF-4 File Format*. 2026. URL. (accessed: 29.05.2026).

A. Findings Register

This appendix is the authoritative register of formal findings for this revision and maintains a separate evidence-and-scoping backlog. It applies the method in Section 1.8 to public evidence available through August 27, 2026. A formal finding is a concrete adverse-event scenario at a stated boundary; it is not a score assigned to a technology, feature, CVE count, recommendation, or project. Broad mechanisms that do not yet define an adverse consequence and affected population appear only in Table 30; they are not formal findings. Unless a record expressly says otherwise, the assessment did not independently reproduce the reported condition and did not assess the current installed population.

Table 28 records inherent-risk axes and tiers where evaluated, together with every finding's rating status. It does not imply that every version or deployment is affected today. A public fix, issue closure, or reporter statement is noted where relevant, but it is not credited as current residual-risk reduction without sufficient evidence of the implemented control and its coverage. An *NE* entry is an evidence or scoping gap, not a Low-risk conclusion.

Response routing is authoritative in Table 31. Recommendation coverage is indexed once in Appendix B; packages, dependencies, owners, effort, acceptance, and residual-risk decisions are maintained in Appendix C. None alters the inherent ratings here. The SSP SIG correspondence is likewise stated once in Table 29. Unless a record says otherwise, current installed-population residual risk was not evaluated.

Table 28 – Formal findings and report-wide rating status

ID	Scenario shorthand	Dim.	I/E	Tier	Status	Conf.
CORE-SEC-001	Malformed on-disk metadata reaches memory-unsafe core-library paths in historically affected releases.	Security	I3/E2–E3	High	Provisional	Moderate
CORE-SEC-002	File-controlled work, allocation, traversal, or error paths terminate or exhaust an affected reader.	Security	I2/E2–E3	Moderate	Provisional	Moderate

Table 28 – Formal findings and report-wide rating status, continued

ID	Scenario shorthand	Dim.	I/E	Tier	Status	Conf.
CORE-TOOL-001	A malicious file triggers the reported `h5dump` use-after-free in HDF5 1.14.1-2 and earlier.	Security	I2–I3/E2	Moderate–High	Provisional	Moderate
CORE-PRIV-001	Sensitive plaintext metadata is disclosed when a file is shared under payload-only protection.	Privacy	I2–I3/E2	Moderate–High	Provisional	Moderate
CORE-SAFE-001	Storage exhaustion during write or close leaves an incomplete or corrupted file after the operation reports failure, with no defined recovery or poisoned-file protocol.	Safety	I2–I3/E2	Moderate–High	Provisional	Low
CORE-SAFE-002	Abrupt process or host termination during write leaves a file that is corrupted or not reopenable.	Safety	I2–I3/E3	Moderate–High	Provisional	Low
LONG-SAF-001	An archived dataset becomes unreadable because a required non-core filter is no longer available.	Safety	I2/E2	Moderate	Provisional	Low
APP-SEC-001	The affected Hugging Face dataset service followed attacker-selected external raw-storage paths with worker authority and returned local files.	Security	I3/E4	Critical	Rated	High
APP-SEC-002	Affected Keras legacy-HDF5 model loading executes embedded code despite `safe_mode=True`.	Security	I3/E2	High	Rated	High
APP-SEC-003	Affected Keras model loading follows attacker-controlled HDF5 external dataset references and discloses local files.	Security	I3/E3	High	Rated	High
APP-SEC-004	Affected Keras weight loading allocates from an extreme file-declared shape and exhausts memory.	Security	I2/E3	Moderate	Rated	High

Table 28 – Formal findings and report-wide rating status, continued

ID	Scenario shorthand	Dim.	I/E	Tier	Status	Conf.
APP-SEC-005	TensorFlow 2.17.0 loads HDF5 plugins from an uncontrolled search location, enabling local privilege escalation.	Security	I3/E2	High	Rated	Moderate
IND-SAF-001	HDF5 2.0/2.1 exact-width validation rejects a sufficient but nonminimal chunk encoding permitted by the specification.	Safety	I2/E2	Moderate	Rated	High
IND-SAF-002	Older permissive readers accept contradictory PureHDF 2.1.2 chunk metadata that HDF5 2.0 rejects after upgrade.	Safety	I2/E2	Moderate	Rated	High

A.1. Correspondence with the SSP SIG Registries

Table 29 is editorial mechanism correspondence, not an adopted mapping or a conversion between this rubric and the SSP SIG's Severity × Likelihood scale. "No direct counterpart" is a scope statement, not criticism: this report adds application, service, and independent-implementation boundaries, while the registries enumerate hazards not examined here.

Table 29 – Editorial correspondence between this register and the SSP SIG registries

This report	SSP SIG entry	Note
CORE-SEC-001	ATK-001–ATK-008	Eight mechanisms correspond to this report's shared memory-unsafe event.
CORE-SEC-002	ATK-009, ATK-010, ATK-012	Resource exhaustion, termination, and I/O amplification; the development-tree assert case maps partially to ATK-010 but remains outside the rating.
CORE-TOOL-001	ATK-007 (mechanism only)	The registry does not separately rate the official-tool boundary.
CORE-PRIV-001	EXP-002; also HAZ-004	Plaintext metadata and structural disclosure.

Table 29 – Editorial correspondence between this register and the SSP SIG registries, continued

This report	SSP SIG entry	Note
CORE-SAFE-001	HAZ-005	Same storage-exhaustion scenario; see Table 6.
CORE-SEC-003	ATK-010 (partial)	No entry directly covers build-mode-dependent enforcement.
CORE-SAFE-002	HAZ-001; also HAZ-002, HAZ-006, HAZ-012, HAZ-015	This report consolidates the registry's durability cluster into one event.
LONG-SAF-001	HAZ-014	Same missing-filter mechanism; this report states a ten-year-or-more horizon.
APP-SEC-001	No direct counterpart	No confused-deputy entry; EXP-008/EXP-009 are related privacy mechanisms.
APP-SEC-002	No direct counterpart	Application deserialization is outside registry scope.
APP-SEC-003	EXP-009 (partial)	Disclosure mechanism, rated here at the application authority boundary.
APP-SEC-004	ATK-009	Same mechanism, rated at the allocating application boundary.
APP-SEC-005	ATK-011	Methodological divergence; see Table 5.
FMT-SAF-001	No direct counterpart	Related to H6/H5II **I1**, but neither records an unresolved normative-artifact question.
IND-SAF-001, IND-SAF-002	No direct counterpart	No independent-implementation or reader/writer-conformance entry.
EXT-PRIV-001	EXP-004–EXP-007; also HAZ-003	Four bounded extension exposures corresponding to this unrated theme.
SUP-SAFE-001	ATK-013, HAZ-014	Distribution compromise and missing-filter portability mechanisms.

A.2. Evidence and Scoping Backlog

The observations below identify work needed to determine whether additional formal findings exist. They have no impact or exposure axes, risk tier, response urgency, or implied Low-risk status.

Table 30 – Observations requiring a concrete adverse-event scenario

ID	Observation	Evidence needed for a finding
`CORE-SEC-003` *candidate population*	Candidate deserializer invariants derived from in-file bytes are masked inside `assert(...)` expressions and therefore may not be enforced in `-DNDEBUG` release builds. Issue #87 identifies seven component families and estimates 50–100 candidates, but does not provide a verified site catalog or consequence analysis. A fixed develop-source sample corroborates the build mechanism and representative sites, not their runtime consequences [57, 86, 81, 82, 83].	For each supported source line and build profile, record the source site, file-byte provenance, non-asserting dominating guard if any, release reachability, first security-relevant sink, affected versions, and consequence. A redundant or unreachable assertion needs a documented negative disposition.

Table 30 – Observations requiring a concrete adverse-event scenario, continued

ID	Observation	Evidence needed for a finding
`CORE-SEC-003` *measured exemplars*	One v2 B-tree `record_size` case advances the evidence without closing the supported-release gap: its source-reported reproduction records `SIGFPE` on an `-DNDEBUG`, release-configured HDF5 2.3.0 development tree, but explicitly records no 32-bit measurement and no affected supported version or version range [33]. A second candidate is reported for the cache-image LRU rank relation without a measured release-build consequence [31]. The remaining population is unquantified, and this record stays open.	Record paired debug and `-DNDEBUG` fixture results. Promote each supported adverse-event scenario into `CORE-SEC-001`, `CORE-SEC-002`, or a new formal finding as the evidence requires. Begin from the measured development- tree case's sibling audit, which names further unaudited assertions in the same B-tree header initializer and in fractal-heap initialization.

Table 30 – Observations requiring a concrete adverse-event scenario, continued

ID	Observation	Evidence needed for a finding
FMT-SAF-001	The reference library and the published file-format specification disagree on whether the group-info message is optional, and no ruling had been recorded at the cutoff. Issue #6409 separately demonstrates the value of an independent implementation, but not indeterminacy: the published rule settled that case and the reader was corrected [4, 29, 5].	Record an authoritative disposition for issue #6616, identify any file or workflow whose behavior depends on the disputed rule, and measure the resulting access, integrity, or compatibility consequence. Until both an adverse outcome and affected population are bounded, this is a format-governance observation, not a rated safety finding. The 90-day RM-2C ruling target is administrative, not risk urgency; FMT-S1/RM-3I supplies the standing adjudication and machine-readable-subset programme.
EXT-PRIV-001	Extensions can move data and metadata across storage, cache, logging, network, and executable-plugin boundaries.	Identify a deployment, sensitive-data population, unauthorized recipient or retention event, affected scope, and resulting harm.
SUP-SAFE-001	Downstream builds and workflows can disagree about ABI, linkage, filters, plugins, VFDs, parallel features, or file locking.	Identify a concrete producer/consumer profile pair, affected population, failure mode, recoverability, and recurrence evidence.

A.3. Response Routing

The routing below follows the default rule in Section 1.8. Where current residual risk was not evaluated, it uses inherent risk as the conservative provisional baseline; a tier range is routed at its upper bound. Routing is conditional on con-

firming the stated affected scope and is not a claim that every present deployment needs the same action. The target horizons begin when an affected deployment or equivalent exposure is identified. They are assessment recommendations, not stakeholder-approved delivery commitments, and do not incorporate effort or dependencies. In particular, Hugging Face reports that it stopped wrongly processing HDF5 external references, but this assessment did not validate the implemented control or assign a current residual tier. Phases 0–2 of Appendix C operationalize these clocks; later structural work neither replaces nor extends them.

Table 31 – Provisional response urgency and target horizons

Urgency and horizon	Findings	Decision rule for this register
Immediate—before the next untrusted processing job and no later than 24 hours after identification	APP-SEC-001	Immediately contain any equivalent service that still automatically follows file-selected external resources with access to sensitive local files. The historical incident rating is not a current rating of Hugging Face.
Near-term—within 30 days	CORE-SEC-001, CORE-TOOL-001, CORE-PRIV-001, CORE-SAFE-001, CORE-SAFE-002, APP-SEC-002, APP-SEC-003, APP-SEC-005	Identify affected releases and exposed deployments, apply available fixes or containment, minimize disclosed metadata, and verify the trust boundary. For the two write-side safety findings the Near-term obligation is *tested operational containment* — capacity headroom and monitoring, flush and checkpoint discipline, a rehearsed recovery procedure, and an independent copy of data that cannot be regenerated. The library changes in CORE-SAFE-S1 and CORE-SAFE-S2 are structural and follow the Planned and Phase 3 horizons.
Planned—within 90 days or the next supported release or migration window, whichever is sooner	CORE-SEC-002, APP-SEC-004, IND-SAF-001, IND-SAF-002, LONG-SAF-001	Plan treatment and acceptance testing; expedite where an affected production pipeline, archive, sensitive-data population, or service is identified. For LONG-SAF-001, identify the bounded archive corpus and its filter dependencies within the clock; longer-term profile and preservation work does not extend that first deliverable.

A.4. Evidence Actions Without Assigned Urgency

Items CORE-SEC-003, FMT-SAF-001, EXT-PRIV-001, and SUP-SAFE-001 require the evidence in Table 30; this is not a fifth urgency. The I2/E2 LONG-SAF-001 rating follows the default Planned route, but its ten-year-or-more horizon still prevents direct ordering against scenarios with no stated horizon. Its 90-day deliverable is the bounded corpus, dependency inventory, and disposition—not completion of decadal preservation. A presently failing archive is scoped and routed separately.

A.5. Core Library, Tool, and Format Findings

CORE-SEC-001 — Memory-unsafe processing of malformed metadata

Scenario. An untrusted originator supplies a crafted HDF5 file to a process using a historically affected HDF5 C library. Insufficiently validated file-derived sizes, counts, offsets, message state, layout state, filter parameters, or datatype state reach allocation, copy, object-construction, or cleanup paths and cause memory corruption, including credible process compromise for the variants whose primary records support that consequence.

Scope and evidence. The assessed population consists only of deployments matching an affected version-and-path pair represented in Appendix D and processing untrusted files. The H5FS, H5C, H5O, H5T, H5Z, heap, dataspace/layout, and vector-copy variants are evidence for the shared adverse event, not separately scored objects. A detailed primary report for HDF5 1.8.16 traces file-controlled array rank to an out-of-bounds heap write and possible code execution in the consuming application [26]; the cited later fixes demonstrate early sanity checks, bounds and overflow tracking, and controlled rejection replacing unsafe behavior [40, 35, 34, 36]. This record excludes termination-only and resource-consumption variants assigned to CORE-SEC-002 and the tool-specific UAF assigned to CORE-TOOL-001. It does not assert that every CVE had code-execution impact or that a current HDF5 2.x release contains every historical condition.

Rating basis. I3 reflects credible process compromise or major loss of process integrity in the evidenced variants. E2 applies to interactive or otherwise constrained file processing; E3 applies where an affected deployment automatically ingests untrusted files through a common workflow. Both produce a High tier, while exact version and deployment prevalence remain incomplete. The rating is therefore **I3/E2–E3, High, Provisional, Moderate confidence**. Current release and installed-population residual risk were not evaluated.

CORE-SEC-002 — Uncontrolled work and resource consumption

Scenario. A crafted, truncated, or inconsistent file reaches affected decode, traversal, arithmetic, allocation, recursion, or lifecycle paths and causes an assertion, null dereference, divide-by-zero, leak, stack exhaustion, or uncontrolled resource consumption, interrupting the consuming process or workflow.

Scope and evidence. The assessed population consists only of deployments matching an affected version-and-path pair in Appendix D and processing the triggering file. This record includes termination and resource-consumption outcomes without evidenced memory corruption; it excludes the memory-unsafe variants assigned to `CORE-SEC-001` and the tool-specific UAF assigned to `CORE-TOOL-001`. Exact fixed versions across the represented paths and the present installed population are not established. The file-space repair is one direct example of converting an assertion reached from corrupted metadata into a controlled error [35, 40].

Related development-tree evidence. The `CORE-SEC-003` backlog contains a source-reported v2 B-tree case in which an assert-only zero-record-size guard is absent in an `-DNDEBUG` build and the public API path terminates with `SIGFPE`. It was measured on a locally built HDF5 2.3.0 development source tree, not an established supported release; the record explicitly says that 32-bit behavior and the affected version range were not measured [33]. It corroborates the mechanism and provides a concrete fixture for `RM-1E`, but it is not included in this formal finding's rated population. A related zero-capacity variant reported against an older h5py-linked library is likewise uncharacterized and unrated.

Rating basis. I2 reflects a meaningful but ordinarily recoverable process or workflow interruption; a mission-level outage requires a separate deployment-specific record. E2 applies to interactive or otherwise constrained file processing; E3 applies where an affected deployment automatically ingests untrusted files through a common workflow. Both produce a Moderate tier, while incomplete version prevalence makes the assessment provisional. The rating is **I2/E2–E3, Moderate, Provisional, Moderate confidence**. Current release and installed-population residual risk were not evaluated.

CORE-TOOL-001 — h5dump use-after-free

Scenario. An attacker supplies a malicious HDF5 file to a user or automated workflow that invokes an affected `h5dump`; the file triggers the reported `H5T__conv_struct` heap use-after-free and may compromise or terminate the inspection process.

Scope and evidence. The HDF Group advisory identifies HDF5 1.14.1-2 and earlier as affected and reproduces AddressSanitizer-detected termination when the supplied file is processed by the utility [28, 84]. It states that practical code-execution exploitability is unknown. This is a tool finding, not evidence that every libhdf5 consumer reaches the same path.

Rating basis. The demonstrated termination supports I2; credible but unverified process compromise from the UAF supports I3 as the upper bound. E2 reflects the affected-version, crafted-file, and explicit inspection preconditions. The rating is **I2–I3/E2, Moderate–High, Provisional, Moderate confidence**. Current installed-population residual risk was not evaluated.

CORE-PRIV-001 — Plaintext metadata disclosure

Scenario. An HDF5 file containing sensitive object names, attributes, member labels, comments, user-block content, provenance, or structural relationships is shared with a recipient who is authorized for a protected payload but not for that context. Direct HDF5 or byte-level inspection reveals the unintended metadata.

Scope and evidence. This finding is limited to files that actually contain sensitive plaintext metadata and a sharing boundary that protects only the nominal payload. The format's self-describing structure establishes the mechanism; the Privacy Exposure Model identifies metadata and structural exposure as privacy threat classes [87, 88].

Rating basis. I2 reflects bounded sensitive-context disclosure; I3 is plausible where the same mechanism reveals high-sensitivity context or substantially affects a defined population. E2 reflects the need for both sensitive metadata and a mis-scoped disclosure boundary. Data sensitivity, affected population, and workflow prevalence were not measured, so the rating is **I2–I3/E2, Moderate–High, Provisional, Moderate confidence**. Deployment-specific findings should replace this range when those facts are known. Current installed-population residual risk was not evaluated.

CORE-SAFE-001 — Storage-exhaustion write failure

Scenario. An application writes an HDF5 file to storage that fills during the write, flush, or close sequence. The operation reports failure through the ordinary HDF5 error path, including `ENOSPC` in the cited example. The library provides no structured VFD callback at the failure point, no defined recovery or safe-abandonment procedure, and no supported way to mark the resulting file as suspect. The file may be incomplete or internally inconsistent, and data that has not been independently preserved may be lost.

Scope and evidence. The assessed population is deployments that write HDF5 files to finite storage without an assured reserve and that lack a separately verified recoverable copy of the in-flight data. The controlling evidence is architectural rather than statistical. In the cited discussion the application receives write and close errors, while a maintainer states that HDF5 has no recovery story for `ENOSPC` and no callback for the VFD event itself. The discussion records a workaround — configuring the metadata cache to discard on evict — and enumerates four candidate treatments: space reservation via `posix_fallocate`, an explicit poisoned-file state, a pre-flush capacity check, and a virtual-file-driver

error callback that lets the caller choose to retry, fail fast, or panic [20]. General file-corruption reports provide contextual recurrence evidence but do not establish that storage exhaustion caused each outcome [18, 8].

Why this is no longer NE. An earlier revision recorded this finding as NE. The additional evidence now establishes a bounded control deficiency: ordinary operation reports an I/O failure, but the cited API and maintainer discussion provide no defined recovery, safe-abandonment, or poisoned-file contract. That is sufficient to state the scenario and estimate impact and exposure provisionally. It is not evidence that every `ENOSPC` event corrupts a file, that applications receive no error, or that storage exhaustion is broadly prevalent; those questions remain part of `RM-2D`.

Rating basis. I2 reflects a recoverable loss where the workload can be rerun or an independent copy exists. I3 reflects major loss or corruption where it cannot; I4 is not assigned because irreversible loss of unique or mission-critical records is not established by the cited evidence. E2 reflects the material preconditions: finite storage must fill during an active write, and the deployment must lack a control that preserves a known-good recoverable state. The rating is **I2–I3/E2, Moderate–High, Provisional, Low confidence**. Confidence is Low because no failure sequence, durable-state result, recoverability outcome, recurrence rate, or affected population has been measured; `RM-2D` can change the tier as well as confidence. Current residual risk was not evaluated.

CORE-SAFE-002 — Corruption from abrupt termination

Scenario. A writer process is killed, crashes, or loses power while an HDF5 file is open and metadata updates are in flight. Because in-memory metadata-cache state and the durable on-disk state can disagree, and because the format has no journaling or write-ahead log, the file may be left internally inconsistent: not reopenable, reopenable only after `h5clear`, or reopenable but missing or misrepresenting recently written data. Data written since the last consistent state may be lost, and the inconsistency may not be detected until a later read.

Scope and evidence. The assessed population is any deployment holding an HDF5 file open for write across a window in which the process or host can fail — long acquisitions, simulation checkpointing, and continuous logging in particular. The evidence is a recurrence pattern in the community forum spanning sixteen years, from requests to recover corrupted files and questions about file state after a crash, through reports that flushing itself sometimes corrupts files, to a 2024 design discussion in which The HDF Group states that a crash or failure "can lead to data loss or corruption of the file" and describes restarting work on crash-proofing through metadata journaling, write-ahead logging, SWMR-derived mechanisms, and checkpointing [18, 16, 17, 11, 12, 8, 15, 9]. That discussion also records a practitioner report that trapping signals and closing the library prevents the worst outcomes and usually leaves files intact, with `h5clear` still needed at times — which is compensating-control evidence, not a supported guarantee.

Rating basis. I2 reflects a recoverable interruption where the file reopens after `h5clear` or the workload can be rerun. I3 reflects major loss or durable corruption where neither holds. E3 reflects a common workflow — any long-running write — with few material preconditions; abrupt termination requires no adversary and no unusual configuration. The rating is **I2–I3/E3, Moderate–High, Provisional, Low confidence**. Confidence is Low because the assessment measured no failure injection, established no version boundaries, and quantified no population: the evidence establishes recurrence and maintainer acknowledgement, not distribution of outcomes.

A.6. Application-Boundary Findings

APP-SEC-001 — External raw storage confused deputy

Scenario. In the affected July 2026 Hugging Face deployment, a remote party supplied a valid HDF5 file whose dataset declared local files as external raw storage. The dataset-processing worker opened those paths with its own filesystem authority and returned their bytes to the requester, disclosing its environment, credentials, and source code.

Scope and evidence. Hugging Face reports that this vector performed file disclosure without executing code; the separate Jinja2 injection supplied code execution. The HDF Group identifies external raw storage as an intentional documented capability and the service as the authority-bearing confused deputy, not an HDF5 parser failure [67, 90]. The rating is confined to the affected service at incident time and to equivalent deployments with the stated preconditions; it is not an ecosystem-prevalence claim.

Rating basis. I3 reflects actual unauthorized resource access and substantial disclosure of secrets, credentials, and implementation details. E4 reflects normal automatic processing of an ordinary attacker upload, transparent reference following, and a direct, repeatable disclosure path within that deployment. The inherent rating is **Critical, Rated, High confidence**.

Current residual and response. Hugging Face states that its renderer no longer wrongly processes HDF5 external references. Public evidence is insufficient to validate control design and coverage, so current residual risk for that service was not evaluated and no residual tier is assigned. The h5py discussion records laborious application- level detection, a sandbox recommendation, and the decision to seek upstream control; HDF5 issue #6618 records the resulting request for unified controls over external data, external links, and virtual sources. Proposals discussed there are not treated as shipped controls [25, 78]. Equivalent still-exposed deployments require Immediate containment.

APP-SEC-002 — Keras legacy-HDF5 code deserialization

Scenario and scope. An attacker supplies a crafted legacy `.h5`/`.hdf5` model to Keras 3.0.0 through 3.11.2. A victim loads it expecting `safe_mode=True` to prevent unsafe reconstruction, but a Lambda layer's serialized code executes in the loader process [68, 63].

Rating basis. I3 reflects arbitrary code execution in the loader's authority. E2 reflects the legacy format, crafted Lambda content, affected version, and passive load preconditions. The rating is **High, Rated, High confidence**.

APP-SEC-003 — Keras external-dataset local-file read

Scenario and scope. A remote attacker supplies a crafted `.keras` model to Keras `>=3.0.0,<3.12.1` or `>=3.13.0,<3.13.2`. During model loading, an embedded HDF5 external dataset reference selects a local path and discloses a file readable by the loader process [70, 64].

Rating basis. I3 reflects unauthorized local-resource access and potentially substantial sensitive-information disclosure. E3 reflects a remotely supplied artifact, few technical preconditions, and a normal model-load action, while remaining bounded to the affected versions and model-loading surface. The rating is **High, Rated, High confidence**.

APP-SEC-004 — Keras shape-driven memory exhaustion

Scenario and scope. A remote attacker supplies a crafted `.keras` archive to Keras `>=3.0.0,<=3.12.0` or `>=3.13.0,<3.13.2`. Its valid `model.weights.h5` declares an extremely large dataset shape; weight loading allocates without an adequate budget, exhausts memory, and crashes the Python interpreter [69, 62].

Rating basis. I2 reflects the evidenced process and workflow interruption; no broader outage or durable data loss is assumed. E3 reflects low-complexity, remotely supplied content crossing a normal model-loading boundary, subject to the victim or service loading the artifact. The rating is **Moderate, Rated, High confidence**.

APP-SEC-005 — TensorFlow plugin search-path escalation

Scenario and scope. A low-privileged local attacker on a TensorFlow 2.17.0 installation places a plugin in an unsecured location searched by the HDF5 integration. TensorFlow loads the plugin and executes its code with the target user's authority [59, 91, 79].

Rating basis. I3 reflects privilege escalation and arbitrary code execution. E2 reflects the required local foothold, affected version, writable searched location, and plugin-load path. The condition is supported by the coordinated advisory and vendor fix, but the report did not reproduce it. The rating is **High, Rated, Moderate confidence**.

A.7. Independent-Implementation and Interoperability Findings

IND-SAF-001 — Sufficient nonminimal chunk encoding rejected

Scenario. A workflow upgrades its reader to an HDF5 2.0/2.1 release containing the exact-width check—confirmed in 2.0.0, 2.1.0, and 2.1.1—and opens a version-4 chunk layout whose encoded dimension width is sufficient but nonminimal. Validation rejects the file even though the format did not require the smallest possible width, interrupting access to data that earlier readers accepted.

Scope and evidence. HDF5 issue #6409 distinguishes the specification-permitted encoding from an undersized invalid encoding and records effects on independent writers. The merged reader-side change accepts any sufficient width and was tracked against the HDF5 2.2.0 milestone; this report does not claim that a specific release artifact or backport contains it without release-level verification [4, 29, 22, 66].

Rating basis. I2 reflects a meaningful, bounded interoperability and data-access failure. E2 reflects the specific layout version, nonminimal encoding, and reader-upgrade preconditions. The rating is **Moderate, Rated, High confidence** for the historical affected scope. Current residual risk was not evaluated, and no residual tier is assigned.

IND-SAF-002 — Contradictory PureHDF chunk metadata

Scenario. A production or archival workflow upgrades from a tolerant HDF5 1.x reader to HDF5 2.0 and reads an affected compressed file produced by PureHDF 2.1.2. The file's version-4 chunk layout contradicts its datatype metadata; the

stricter reader correctly rejects the affected datasets, interrupting processing and stranding existing data until the reader is pinned or the files are repaired or regenerated.

Scope and evidence. In the supplied production-derived file, the HDF5 maintainer analysis found float64 datatype element size 8 while the chunk layout recorded stored element size 4. Older HDF5 1.10.10 and 1.14.6 readers accepted the bytes; HDF5 2.0 rejected them. The issue was closed as malformed writer output, and the PureHDF discussion records the writer-side response [2, 1]. This condition is distinct from the sufficient nonminimal width in `IND-SAF-001`; permissive acceptance is not evidence of conformance.

Rating basis. I2 reflects actual production workflow interruption and a persistent but repairable compatibility problem; no irreversible data loss is evidenced. E2 reflects the confirmed writer version, affected layout, compressed dataset, and reader-upgrade preconditions. The rating is **Moderate, Rated, High confidence**. A technical metadata-repair path was proposed and older readers remained a workaround, although the reporter said rewriting affected production files was not operationally feasible in that pipeline. Later PureHDF output restored production operation, but the inventory and repair status of existing files are unknown. Current residual risk for the legacy corpus was not evaluated, and no residual tier is assigned.

A.8. Format and Specification Evidence Record

`FMT-SAF-001` remains unrated for the reasons and evidence requirements in Table 30; its separate administrative and structural actions are canonical in Appendix C.

A.9. Long-Term Accessibility Findings

LONG-SAF-001 — Filter-dependent loss of interpretability

Scenario. An HDF5 file is retained in an archive and a dataset in it requires a non-core filter plugin. At some later point the filter is absent from the reader's build or platform, unbuildable against the current toolchain, or no longer obtainable. The file bytes survive, but the affected dataset cannot be decoded.

Assessment horizon. This finding states an assessment horizon of **ten years or more** from the date of writing. The cited reports establish that missing-filter failures already occur; they do not measure how extension availability changes over a decade. The horizon is therefore a material assumption and, under decision rule 7 of Section 1.8, this rating must not be directly ordered against ratings that state no horizon.

Scope and evidence. The assessed population is archived files containing datasets that require a non-core filter. The mechanism is observable at short horizons: community reports record users unable to read data because a required compression filter was absent from their build or platform, across a fifteen-year span and several filter families [19, 13, 10, 14]. The archival extrapolation — that availability decreases rather than increases over a decade — is reasoned inference, not measurement, and is the reason confidence is Low.

VFD and VOL dependencies are excluded from the rated population. A missing VFD can affect whether bytes can be reached at all, while a VOL connector may map to a different store rather than to a self-contained HDF5 file. Those are materially different mechanisms and affected populations, and the report has no comparable archival failure evidence for them. They remain reasoned extensions of the analysis and require separate scenarios before rating.

Rating basis. I2 reflects the evidenced material workflow interruption: an affected dataset cannot be read until the filter is recovered, rebuilt, or re-implemented. I3 would require evidence that unavailable decoding caused consequential or irrecoverable loss of significant records; the cited reports do not establish that outcome. E2 reflects the material preconditions — an archived dataset must

depend on a non-core filter and that filter must be unavailable to the reader. The rating is **I2/E2, Moderate, Provisional, Low confidence**, at the stated horizon. Confidence is Low because the archived population, dependency survival distribution, and decadal availability trend were not measured.

B. Recommendation Register

This appendix is the authoritative reverse index for the forty-three recommendations issued across Sections 2–8. It records each recommendation's purpose, owner, covered finding or evidence record, work package, and effort. Response routes remain authoritative in Table 31; dependencies, clocks, acceptance evidence, closure, and residual-risk decisions remain authoritative in Appendix C. Keeping those facts in one place prevents a summary from silently diverging from the delivery plan.

Recommendation numbers express no risk, urgency, effort, dependency, or delivery order. A link to an evidence record is evidence-conditional and inherits no urgency. Effort uses Table 4 and describes work breadth, not elapsed time or risk. At publication every item is *Proposed*, implementation is *not verified*, and no residual-risk decision is recorded.

Six effort estimates assume that reviewed H5Lens / `h5lens` identifiers, boundaries, fixtures, and measurements can reduce implementation work after artifact-level provenance, license, and, where needed, legal review under Section C.3: `CORE-S1`, `CORE-S2`, `CORE-S4`, `CORE-S5`, `CORE-S6`, and `APP-S2`. They are marked **re-scoped**; this planning assumption is not a legal opinion and does not shorten a response clock.

Table 32 – Recommendation reverse index

ID	Purpose / owner	Finding or evidence record	Package / effort
Core security and safety			
`CORE-S0`	Separate defect from recurring-class closure / `CORE`	`CORE-SEC-001`, `CORE-SEC-002`	`RM-1B`, `RM-3C`, `RM-4A` / Medium
`CORE-S1`	Build a file-format invariant registry / `CORE`	`CORE-SEC-001`, `CORE-SEC-002`, `IND-SAF-001`, `IND-SAF-002`	`RM-3A` / Medium **(re-scoped)**
`CORE-S2`	Introduce a two-phase parser boundary / `CORE`	`CORE-SEC-001`, `CORE-SEC-002`	`RM-3B` / Large **(re-scoped; unchanged)**
`CORE-S3`	Centralize checked arithmetic and allocation / `CORE`	`CORE-SEC-001`, `CORE-SEC-002`, `APP-SEC-004`	`RM-2A`, `RM-3A` / Large

Table 32 – Recommendation reverse index, continued

ID	Purpose / owner	Finding or evidence record	Package / effort
CORE-S4	Create file-opening security profiles / CORE	APP-SEC-001, APP-SEC-003, APP-SEC-004, CORE-SEC-001, CORE-SEC-002	RM-2A / Medium **(re-scoped)**
CORE-S5	Fuzz by structure / CORE	CORE-SEC-001, CORE-SEC-002	RM-3C / Medium **(re-scoped)**
CORE-S6	Make malformed-input regression cheap / CORE	CORE-SEC-001, CORE-SEC-002, CORE-TOOL-001, IND-SAF-001, IND-SAF-002	RM-1B, RM-2C, RM-3C / Small **(re-scoped)**
CORE-S7	Block risky parser changes at review / CORE	CORE-SEC-001, CORE-SEC-002; CORE-SEC-003 evidence	RM-1E, RM-3C / Medium
CORE-S8	Separate compatibility from safety / CORE	CORE-SEC-002, IND-SAF-001, IND-SAF-002	RM-2C, RM-4A / Medium
CORE-S9	Harden tools as distinct attack surfaces / CORE	CORE-TOOL-001, CORE-SEC-001, CORE-SEC-002	RM-1B, RM-2A, RM-3C / Medium
CORE-S10	Govern and authenticate plugins / EXT	APP-SEC-005; EXT-PRIV-001 evidence	RM-3D / Large
CORE-S11	Coordinate supply-chain and parser controls / REL	SUP-SAFE-001 evidence; provenance support for all records	RM-3E, RM-4A / Large
CORE-S12	Publish behavior-forcing metrics / CORE	CORE-SEC-001, CORE-SEC-002, CORE-TOOL-001; CORE-SEC-003 evidence	RM-1B, RM-3C, RM-4A / Medium
CORE-SAFE-S1	Define storage-exhaustion behavior / CORE, APP	CORE-SAFE-001	RM-1F, RM-2D, RM-3G / Medium
CORE-SAFE-S2	State and improve crash consistency / CORE, APP	CORE-SAFE-002	RM-1F, RM-2D, RM-3G / Large
Privacy			
CORE-PRIV-S1	Establish privacy-exposure awareness / PRIV	CORE-PRIV-001	RM-1D / Small
CORE-PRIV-S2	Document tool and plugin exposure / PRIV	EXT-PRIV-001 evidence	RM-3D / Medium
CORE-PRIV-S3	Apply privacy-aware system design / PRIV	CORE-PRIV-001; EXT-PRIV-001 evidence	RM-1D, RM-3F / Medium

Table 32 – Recommendation reverse index, continued

ID	Purpose / owner	Finding or evidence record	Package / effort
CORE-PRIV-S4	Use encryption where policy requires / PRIV	CORE-PRIV-001 conditionally	RM-2G / Large
Downstream applications and services			
APP-S1	Define an untrusted-HDF5 profile / APP	APP-SEC-001, APP-SEC-003, APP-SEC-004, CORE-SEC-002	RM-0A, RM-1C, RM-2A, RM-2B / Medium
APP-S2	Ship an enforcement-coupled preflight validator / APP, CORE	APP-SEC-001, APP-SEC-003, CORE-SEC-002, CORE-TOOL-001	RM-2A, RM-2B / Medium **(re-scoped)**
APP-S3	Disable dangerous ML-loader features by default / APP	APP-SEC-002–APP-SEC-004	RM-1C / Medium
APP-S4	Make external references opt-in / APP	APP-SEC-001, APP-SEC-003	RM-0A, RM-1C, RM-2B / Medium
APP-S5	Treat plugins as executable code / APP	APP-SEC-005	RM-1C, RM-3D / Medium
APP-S6	Require downstream SBOM and version reporting / REL, APP	SUP-SAFE-001 evidence; census support	RM-1A, RM-3E, RM-4A / Medium
APP-S7	Apply advisory scope tags / GOV	None; triage support	RM-1A, RM-4A / Small
Extensions			
VOL-S1	Document connector privacy characteristics / EXT	EXT-PRIV-001 evidence	RM-3D / Small
VOL-S2	Expose connector runtime introspection / EXT	EXT-PRIV-001 evidence	RM-3D / Medium
VOL-S3	Strengthen connector trust and deployment / EXT	EXT-PRIV-001 evidence; mechanism shared with APP-SEC-005	RM-3D / Medium
VOL-S4	Publish privacy-aware connector guidance / EXT, PRIV	EXT-PRIV-001 evidence	RM-3F / Small

Table 32 – Recommendation reverse index, continued

ID	Purpose / owner	Finding or evidence record	Package / effort
FILTER-S1	Strengthen filter trust and deployment / EXT	EXT-PRIV-001 evidence; mechanism shared with APP-SEC-005	RM-3D / Medium
FILTER-S2	Document privacy-relevant filter configuration / EXT	EXT-PRIV-001 evidence	RM-3D / Small
FILTER-S3	Protect privacy-sensitive filter metadata / PRIV	EXT-PRIV-001 conditionally	RM-3F / Medium
FILTER-S4	Minimize filter-workflow artifacts / PRIV	EXT-PRIV-001 evidence	RM-3F / Medium
FILTER-S5	Standardize filter privacy documentation / EXT	EXT-PRIV-001 evidence	RM-3D / Small
VFD-S1	Document storage behavior / EXT	EXT-PRIV-001 evidence	RM-3D / Small
VFD-S2	Minimize unintended persistence / PRIV	EXT-PRIV-001 evidence	RM-3F / Medium
VFD-S3	Strengthen cloud/distributed-storage guidance / EXT	EXT-PRIV-001 evidence	RM-3D / Small
VFD-S4	Strengthen VFD trust and deployment / EXT	EXT-PRIV-001 evidence; mechanism shared with APP-SEC-005	RM-3D / Medium
Interoperability and supply chain			
FMT-S1	Make the specification adjudicable and machine-readable / INTEROP	FMT-SAF-001 evidence	RM-2C, RM-3I / Large
IND-S1	Maintain a differential conformance corpus / INTEROP	IND-SAF-001, IND-SAF-002	RM-2C, RM-4A / Large
LONG-S1	Make extension dependencies discoverable and removable / INTEROP, EXT	LONG-SAF-001	RM-2H, RM-3H, RM-2C, RM-3E / Medium
SUP-S1	Maintain a producer/consumer compatibility matrix / REL	SUP-SAFE-001 evidence	RM-3E, RM-4A / Large

B.1. Correspondence with SSP SIG Policy Controls

The SSP SIG HDF5-SHINES-⋆ controls use a different namespace [48]. Table 33 is editorial, family-level correspondence, not a control-satisfaction claim; the SSP SIG retains that judgment. A control-level mapping against the OpenSSF matrix is an adoption-time GOV task [52].

Table 33 – Recommendation families mapped to SSP SIG policy-control families

Recommendations	Control family	Scope note
CORE-S0, APP-S7	HDF5-SHINES-VULN-⋆	Triage and disclosure.
CORE-S1, CORE-S2	HDF5-SHINES-TM-⋆	Threat model and architecture.
CORE-S3, CORE-S7	HDF5-SHINES-SDLC-⋆	Review-blocking SDLC gates.
CORE-S4, CORE-S9, APP-S1–APP-S4	HDF5-SHINES-HARD-⋆	Secure profiles and tools.
CORE-S5, CORE-S6, IND-S1	HDF5-SHINES-TEST-⋆	Testing and fuzzing.
CORE-S8, SUP-S1, LONG-S1, FMT-S1	HDF5-SHINES-COMP-⋆	Compatibility and lifecycle.
CORE-SAFE-S1, CORE-SAFE-S2	No family	Durability/control-catalog gap.
CORE-S10, APP-S5, VOL-S1–VOL-S4, FILTER-S1–FILTER-S5, VFD-S1–VFD-S4	HDF5-SHINES-PLUG-⋆	Extension governance.
CORE-S11, APP-S6	HDF5-SHINES-BUILD-⋆, HDF5-SHINES-DEP-⋆, HDF5-SHINES-REL-⋆	Build and provenance.
CORE-S12	HDF5-SHINES-GOV-⋆	Program metrics.
CORE-PRIV-S1–CORE-PRIV-S3	HDF5-SHINES-PRIV-⋆	Data handling.
CORE-PRIV-S4	HDF5-SHINES-CRYPTO-⋆	Keyed protection and recovery.

The catalog's incident-response family and CI/CD least-privilege and input- sanitization controls have no recommendation here. RM-0A preserves evidence only incidentally; build-pipeline security is outside scope.

B.2. Recommendations Not Tied to a Formal Finding

Four groups deliberately address no formal finding: APP-S7 supports triage; the extension track and CORE-PRIV-S2 address EXT-PRIV-001; CORE-S11, APP-S6, and SUP-S1 address SUP-SAFE-001; and FMT-S1 addresses FMT-SAF-001. They inherit no urgency and do not convert an observation into a rating. Every formal finding nevertheless has at least one linked recommendation, including CORE-SAFE-001 through CORE-SAFE-S1.

C. Phased Mitigation Roadmap

This appendix turns Table 31 into a proposed delivery plan; it does not show stakeholder acceptance or implementation. Phase 0–2 clocks begin when an affected deployment or equivalent exposure is identified. The `RM-1A` census, `RM-1E` assertion catalog, evidence actions, and Phase 3 programme begin at roadmap adoption. Adoption must assign dated owners and inventory deadlines; a pre-adoption exposure enters its response route immediately.

Phase, dependency, owner, effort, and sequence are treatment-planning fields, not risk inputs. A blocked final control does not extend a response clock: the owner maintains tested containment, records the variance, and obtains an explicit residual-risk decision. At publication every package is *Proposed*, implementation is *not verified*, and no residual risk has been accepted. Within ten business days of adoption, `GOV` should add a named accountable person, programme record, date or release, and status to each untriggered package. A triggered Phase 0 record receives its `APP` owner and `GOV` authority at T0.

C.1. Accountability Roles

Table 34 – Proposed roadmap accountability roles

Code	Role	Accountability
`GOV`	Roadmap and risk authority	Scope, assignments, gates, exceptions, and residual-risk decisions.
`CORE`	Core library and tools	Parser, arithmetic, policy API, tool, regression, fuzz, and durability changes.
`APP`	Application/service/storage operations	Deployment policy, containment, rollout, capacity, checkpoint, recovery, and independent copies.
`PRIV`	Privacy and data governance	Classification, minimization, disclosure, retention, encryption, and privacy acceptance.
`INTEROP`	Format and interoperability	Specification, reference and independent implementations, fixtures, repair, and migration.

Table 34 – Roadmap roles, continued

Code	Role	Accountability
EXT	Extension ecosystem	Plugin, VOL, VFD, and filter policy, manifests, trust, introspection, and documentation.
REL	Release and supply chain	Provenance, dependencies, SBOMs, release gates, and configuration reporting.
ASSURE	Independent verifier	Reproduces acceptance evidence and signs the verification used for closure.

Each package has one accountable owner, plus a responsible implementer, ASSURE verifier, and release or deployment approver. Cross-project work is instantiated once per governed scope. For High or Critical scope, the verifier is not also owner, rollout approver, or risk acceptor. Only GOV approves gates, exceptions, or residual-risk dispositions.

C.2. Phase Gates and Clocks

Phases are delivery waves, not severity levels; work may run in parallel, but a later structural package never extends an earlier response clock. Each finding-and-scope instance therefore has an SLA child record containing its scope, T0, route, due and performed dates, interim action, status, and evidence, separate from a shared package's structural due date.

Table 35 – Roadmap phases and exit gates

Phase	Clock	Outcome	Exit gate
0—Contain	Before the next untrusted job and no later than 24 hours after identification	Remove the external-resource path from attacker reach; preserve evidence and rotate exposed secrets where indicated.	No confirmed Critical-equivalent path remains active without tested containment; owner, scope, and decision are recorded.

Table 35 – Roadmap phases and exit gates, continued

Phase	Clock	Outcome	Exit gate
1—Verify/remediate	Census and assertion catalog within 30 days of adoption; Near-term treatment within 30 days of affected-scope identification	Bound the population and apply a supported fix or tested operational control.	Each applicable finding has scope evidence, a negative test or rehearsed control, implementation record, and residual-risk disposition; confirmed assertion paths are routed without waiting for global migration.
2—Release safeguards	Planned treatment within 90 days of identification or the next supported release/migration window, whichever is sooner; evidence actions target 90 days after adoption	Deliver strict-processing, interoperability, archive, and bounded evidence packages.	Tests pass; compatibility cases have deterministic outcomes; evidence records are promoted and routed, negatively dispositioned, or remain bounded and open.
3—Recurring classes	Target 3–12 months after adoption through staged supported releases	Deliver invariant parsing, validation, fuzz/review gates, durability architecture, and governed extensions.	Each named tranche passes acceptance; untreated tranches remain visible with containment and dates.
4—Sustain/reassess	PR, nightly, release, and at least annual cadence as applicable	Detect regressions and ecosystem mismatch; maintain verified residual decisions.	Gates run; failures block or receive named, expiring exceptions; registers are updated.

When a package covers several routes, each finding keeps its own maximum horizon. Co-delivery does not raise or lower urgency, and a package cannot hide an overdue SLA child behind its later architectural date.

C.3. Licensing Constraint on Reuse of Existing Work

`RM-1E`, `RM-2A`, `RM-2C`, and `RM-3A`–`RM-3C` may draw on the H5Lens / `h5lens` decoder, invariant registry, preflight oracle, corpus, and measurements [30, 39, 38]. H5Lens reports GPLv3-or-later licensing while HDF5 uses a permissive

BSD-style license. This is an implementation, provenance, and governance constraint, not a legal conclusion. Before reuse, GOV/REL must inventory rights and licenses, record provenance per artifact, and obtain qualified review where needed.

Planning assumes only that approved factual interfaces, reproducible fixtures, and measurements may inform independent native implementation. It does not authorize source transfer, linking, vendoring, redistribution, or relicensing. Invariant vocabulary is candidate design input; an external oracle is evaluated separately from component reuse; each fixture receives its own provenance and redistribution decision; and case records remain triage inputs requiring local reachability, version, and consequence verification. If review rejects these uses, affected effort estimates are revised.

C.4. Work-Package Register

This is the authoritative package, dependency, owner, phase, and effort register. Finding coverage is indexed in Table 39; recommendation coverage is indexed in Table 32; observable acceptance is stated once in Table 38. Dependencies enable final controls but never defer containment. Effort is breadth, not risk or elapsed time.

Table 36 – Proposed mitigation work packages

ID	Phase / owner	Objective	Dependencies	Effort
RM-0A	0 / APP	Disable or sandbox all supported external resource forms in untrusted processing; remove unnecessary worker authority, preserve evidence, and rotate indicated secrets.	None; isolate or stop if no library control exists.	Medium
RM-1A	1 / GOV	Census deployments, supported versions, features, provenance, authority, scope tags, build/assertion mode, owners, and current control evidence.	Deployment and project evidence.	Medium

Table 36 – Mitigation work packages, continued

ID	Phase / owner	Objective	Dependencies	Effort
RM-1B	1 / CORE	Patch or isolate affected core/tools; replace confirmed sole assertion guards; retain minimized debug/-DNDEBUG and sanitizer regressions.	RM-1A; promoted RM-1E cases; supported fix or containment.	Medium
RM-1C	1 / APP	Patch or contain affected loaders; test safe deserialization, external-reference denial, allocation budgets, and plugin-path ownership.	Triggered RM-0A; RM-1A; supported release or local control.	Medium
RM-1D	1 / PRIV	Bound sensitive metadata and recipients; minimize/tokenize or restrict the sharing boundary.	RM-1A; data owner and classification.	Medium
RM-1E	1 / CORE	Within 30 days of adoption, catalog supported file-byte-dependent assertion sites, guards, reachability, sinks, paired build results, versions, consequences, and dispositions; immediately route confirmed adverse paths.	Supported-branch owners and reproducible builds.	Medium
RM-1F	1 / APP	Within 30 days of deployment identification, test headroom alerts, checkpoint/quiescence discipline, recovery rehearsals, and an independent copy against stated recovery objectives.	Deployment, storage, and data owners; census is not prerequisite for known scope.	Medium
RM-2A	2 / CORE	Ship an enforceable untrusted-input profile and identity-bound preflight with external-resource, plugin, traversal, metadata, allocation, and expansion budgets; harden tools.	RM-1A; RM-1B fixtures are coordination input.	Large
RM-2B	2 / APP	Integrate runtime denial and identity-bound preflight; enforce overflow-safe object and aggregate budgets before materialization.	RM-1A, RM-1C; coordinate with RM-2A, using an interim wrapper if needed.	Medium
RM-2C	2 / INTEROP	Maintain valid, invalid, and contested fixtures across supported readers/writers; inventory and disposition legacy files; adjudicate each identified spec/library dispute within 90 days of adoption.	RM-1A; external readers, writers, and fixtures.	Large
RM-2D	2 / CORE	Inject storage exhaustion and abrupt termination across write/flush/close, versions, VFDs, filesystems, and buffering; measure errors, durable state, reopen/repair, read-back divergence, and loss.	Fault injection, representative workloads, and data-owner input.	Large

Table 36 – Mitigation work packages, continued

ID	Phase / owner	Objective	Dependencies	Effort
RM-2E	2 / PRIV	Trace one bounded extension deployment's data, credentials, caches, logs, networks, recipients, and retention to a consequence or defensible negative result.	Sponsor, EXT, classification, and observable environment.	Medium
RM-2F	2 / REL	Reproduce a producer/consumer mismatch with exact profiles, artifacts, failure, recoverability, population, and recurrence.	INTEROP, downstream owners, builds, and fixtures.	Medium
RM-2G	2 / PRIV	Where minimization and recipient controls are insufficient, deploy all-metadata protection with tested workflow, custody, backup, and recovery.	RM-1D; supported design and key custodian.	Large
RM-2H	2 / INTEROP	Within 90 days of archive identification, bound the corpus, manifest every non-core dependency, test absence, classify self-contained/external views, and disposition unavailable/removable dependencies.	Repository owner and preservation objective; other package inputs are non-blocking.	Medium
RM-3A	3 / CORE	Deliver versioned invariants and mandatory checked arithmetic, bounds, allocation, and traversal primitives, each mapped to always-active enforcement.	RM-1B; RM-1E/RM-2C coordination by tranche.	Large
RM-3B	3 / CORE	Migrate prioritized structures to bounded raw decode, semantic validation, then construction, in releasable tranches.	RM-3A.	Large
RM-3C	3 / CORE	Make class closure auditable through structure-aware fuzzing, permanent regression, parser/tool review gates, build- mode parity, and public metrics.	RM-3A; harness/corpus may start earlier.	Large
RM-3D	3 / EXT	Establish plugin identities, protected paths, manifests, resource limits, signature tests, and bounded extension privacy documentation/introspection.	RM-2A; privacy priority from RM-2E.	Large
RM-3E	3 / REL	Publish SBOMs, pinned automation, signatures, provenance, and feature reports; maintain a tested producer/consumer profile matrix and deterministic mismatch dispositions.	RM-1A; mismatch evidence from RM-2F.	Large

Table 36 – Mitigation work packages, continued

ID	Phase / owner	Objective	Dependencies	Effort
RM-3F	3 / PRIV	Implement evidenced extension minimization, temporary-artifact handling, retention/deletion, and protection of sensitive filter metadata.	RM-1D, RM-2E; encryption uses RM-2G.	Large
RM-3G	3 / CORE	Deliver VFD-level storage-exhaustion notification, suspect-file state, optional reservation/pre-flush controls, a per-layout/driver crash guarantee, and selected crash-proofing mechanism.	RM-2D; a pass-through VFD may prove the design.	Large
RM-3H	3 / INTEROP	Automate per-file dependency reports; define an archival profile and deposit classification; preserve reviewed codec specifications and test vectors.	RM-2H, RM-2C, RM-3E, and repository partner.	Medium
RM-3I	3 / INTEROP	Maintain a declared machine-readable format subset generating validators/fixtures and a standing adjudication procedure; GOV decides whether it is normative.	RM-2C; reuse requires Section C.3 review.	Large
RM-4A	4 / GOV	Operate gates and metrics; rerun applicability and residual reviews; sustain corpora and close or re-plan exceptions.	Begins with Phase 1 and absorbs accepted packages.	Medium recurring

C.5. Minimum Viable Subset Under Constrained Capacity

Capacity ordering is not risk ranking and never changes a route or clock. Phase 0–1 obligations and the bounded Planned deliverables remain due even if the structural programme is deferred. The prevention ordering below is the assessor's qualitative judgment under the stated reuse assumptions; no engineer-week, cost, or comparative-effectiveness model was available.

Table 37 – Capacity-constrained programme ordering

Tier	Items and basis	Boundary if deferred
A—not discretionary	Triggered RM-0A; RM-1A–RM-1F. These contain Critical-equivalent paths, establish population, treat known scope, contain write-side hazards, and bound assertions.	Omission breaches an Immediate or Near-term clock. Census incompleteness cannot be used to defer a known deployment.
Bounded Phase 2	Case-specific RM-2C treatment and RM-2H archive disposition complete within their Planned 90-day clocks.	Broader matrices, machine-readable format work, automated reporting, and archival profile design may be sequenced; the bounded deliverables may not.
B—fund first	CORE-S6, CORE-S7, CORE-S3, CORE-S4, then CORE-S1 through their mapped packages. Existing corpus, profile, and invariant artifacts may reduce design/enumeration work after review; checked primitives are tranchable.	Regression, change-time gates, shared policy, and measurable invariant coverage remain incomplete; local treatments stay open as local rather than class closure.
C—explicit deferral	Remaining structural packages, including parser migration, expanded fuzzing, plugin/provenance, encryption, full differential and profile matrices, durability architecture, extension privacy, archive automation, and machine-readable format governance.	Record a scoped, expiring exception and retain interim controls. Deferral prevents the corresponding class, lifecycle, durability, or governance closure claim.

Funding A, bounded Phase 2, and B is coherent but does not eliminate recurring classes. CORE-S0 prevention records remain open and CORE-S12 metrics must show the untreated denominator.

C.6. Acceptance and Closure Evidence

Table 38 is the canonical acceptance specification. For every row, the implementer links artifacts, ASSURE reproduces or independently verifies them, the package owner attests readiness, and the release/deployment approver authorizes rollout. GOV then records the credited scope and a current residual-risk decision. Guidance

or code shipment alone is insufficient; failed, incomplete, or inconclusive tests remain open.

Table 38 – Required work-package acceptance evidence

Package	Required observable evidence
RM-0A	Configuration/code diff and deployment proof; negative fixtures for every supported external-resource form; sandbox-escape test; indicated credential/secret rotation.
RM-1A	Machine-readable inventory of version, linkage, features, build/assertion mode, scope tag, trust/authority, exposure, owner, boundary, completeness attestation, and evidence date.
RM-1B	Version matrix, patch/isolation record, minimized fixtures, and assertion-enabled plus -DNDEBUG ASan/UBSan/LSan controlled-error results.
RM-1C	Rollout evidence and negative tests for executable reconstruction, external paths, allocation budgets, and plugin paths; startup state and canaries in CWD, environment-selected, and writable searched paths.
RM-1D	Classified metadata/data-flow and recipient matrix; before/after HDF5-API and raw-byte inspection; decision whether RM-2G is required.
RM-1E	Versioned site catalog with branch, function, predicate, file-byte provenance, dominating guard, first sink, reachability, paired build results, version scope, consequence, and disposition; tests and promoted finding/SLA children for confirmed adverse paths.
RM-1F	Defined scope and recovery objectives; alert before injected exhaustion; checkpoint/quiescence runbook; successful exhaustion and abrupt- termination rehearsals; verified independent-copy recency, integrity, location, and failure-domain separation.
RM-2A	API/policy documentation, default-state tests, hostile corpus, resource/plugin/external-reference tests, file-replacement identity test, and tool-default tests.
RM-2B	Per-downstream integration and deployment evidence; extreme, overflow, cumulative, and valid-boundary allocation tests before materialization; external-reference and file-replacement tests.

Table 38 – Work-package acceptance evidence, continued

Package	Required observable evidence
RM-2C	Versioned fixture metadata and reader matrix; exact release/ backport verification; #6409 sufficient/undersized pair; corrected-writer round trip; complete legacy enumeration and checksum/provenance-preserving repair or disposition; dated rulings naming authority and changed artifact.
RM-2D	Fault-injection matrix for both triggers across API sequence, version, VFD/filesystem/buffering profile, recording surfaced error, durable state, reopen/repair, acknowledged-versus-read-back divergence, uniqueness, and workload boundary.
RM-2E	Reproducible extension data-flow test with classified population, recipients, endpoints, caches/logs/artifacts, retention, controls, and bounded consequence or documented negative result.
RM-2F	Exact producer/consumer profiles, reproducible fixture and diagnostic, artifact inventory, recoverability, population, recurrence, and documented negative or promoted result.
RM-2G	Ciphertext/metadata inspection, unauthorized-open and partial-I/O tests, protected key inventory, backup/restore, and successful key-loss recovery.
RM-2H	Defined corpus and completeness attestation; per-file manifest; self-contained/external-view classification; absent-dependency tests; dated owner disposition; provenance, license-review, integrity, and reconstruction evidence for retained runtime artifacts.
RM-3A	Versioned invariants, code mapping, checked primitives, migration inventory, single-invariant rejection in both build modes, and proof that no declared invariant relies only on an assertion.
RM-3B	Per-component decode/validate/construct boundary; no native object before validation; valid/malformed corpus plus build-mode sanitizer and fuzz evidence. Closure names migrated components only.
RM-3C	PR/nightly/weekly/release/CVE-closure jobs, minimized seeds, structure coverage, review records, metrics, build-mode parity, and a test showing an assertion-only file predicate fails the gate.

Table 38 – Work-package acceptance evidence, continued

Package	Required observable evidence
RM-3D	Plugin manifest/policy, protected allowlist or keystore, rejection of unsigned/untrusted/writable-path/over-budget/wrong-profile plugins, and extension documentation/introspection tests.
RM-3E	SBOMs, checksums, signatures, verified provenance, pinned automation, dependency scans, feature report, finite profile catalog, fixtures, CI results, and deterministic disposition for each unsupported pair.
RM-3F	Tests showing minimized extension metadata/caches, secure temporary handling, enforced retention/deletion, and, where needed, ciphertext inspection plus key recovery.
RM-3G	Distinguishable storage-exhaustion errors per supported VFD/ buffering profile; tested suspect-state lifecycle; published layout/driver guarantee; writer-kill outcomes and read-back divergence. Reported success followed by inconsistency blocks acceptance.
RM-3H	Generated per-file dependency reports verified by absence tests; archival-profile files readable with non-core plugins disabled; deposit classifications; independent codec reproduction from archived specification and vectors.
RM-3I	Declared machine-readable subset and generated fixtures; divergences classified as library defect, description/specification defect, or explicit implementation-defined behavior with documented consequence; adjudication procedure, decision on normative status, and current dispute ledger. An unruled dispute remains open.
RM-4A	Production-equivalent cadenced results, trends, updated mappings, expired-exception report, and signed point-in-time residual decisions.

C.7. Finding and Evidence-Action Coverage

A formal finding-and-scope child closes only when the affected population is treated or shown inapplicable, its applicable package evidence passes, and GOV records current residual risk. This never changes the historical inherent rating. Structural work stays open independently unless explicitly credited. An evidence record closes only with a bounded positive, negative, or promoted disposition;

inconclusive work is re-planned. Table entries are ID-only indexes into the authoritative package and acceptance tables.

Table 39 – Register-to-roadmap coverage

Record	Bounded treatment or evidence	Continuing prevention / boundary
CORE-SEC-001	RM-1A, RM-1B	RM-2A, RM-3A–RM-3C, RM-4A; only named tranches close.
CORE-SEC-002	RM-1A, RM-1B	RM-2A, RM-2C, RM-3A–RM-3C, RM-4A.
CORE-TOOL-001	RM-1A, RM-1B	RM-2A, RM-3C, RM-4A.
CORE-PRIV-001	RM-1A, RM-1D; conditional RM-2G	RM-4A; closure states the tested disclosure boundary.
CORE-SAFE-001	RM-1A, RM-1F, RM-2D	RM-3G; closure applies only to the named deployment and measured profile.
CORE-SAFE-002	RM-1A, RM-1F, RM-2D	RM-3G; structural closure names the delivered mechanism/layout/driver.
LONG-SAF-001	RM-2H	RM-3H, RM-2C, RM-3E, RM-4A; closure is corpus-specific.
APP-SEC-001	RM-0A, RM-1A, RM-1C	RM-2A, RM-2B, RM-4A; each governed service closes separately.
APP-SEC-002	RM-1A, RM-1C	RM-4A.
APP-SEC-003	RM-1A, RM-1C; RM-0A for a confirmed Critical-equivalent service	RM-2A, RM-2B, RM-4A.
APP-SEC-004	RM-1A, RM-1C	RM-2A, RM-2B, RM-4A.
APP-SEC-005	RM-1A, RM-1C	RM-3D, RM-4A.
IND-SAF-001	RM-1A, RM-2C	RM-3A, RM-4A.
IND-SAF-002	RM-1A, RM-2C	RM-3A, RM-4A.

Table 39 – Register-to-roadmap coverage, continued

Record	Bounded treatment or evidence	Continuing prevention / boundary
CORE-SEC-003	Evidence RM-1A, RM-1E; conditional RM-1B	RM-3A–RM-3C, RM-4A; each supported adverse path is promoted and routed.
FMT-SAF-001	Administrative adjudication RM-2C	RM-3I, RM-4A; closure is dispute/subset-specific and assigns no urgency.
EXT-PRIV-001	Evidence RM-2E	Conditional RM-3D, RM-3F; rate only after population and harm are supported.
SUP-SAFE-001	Evidence RM-1A, RM-2F	Conditional RM-3E; rate only after a bounded adverse scenario is supported.

C.8. Change Control and Residual Risk

Each implementation record contains package, accountable owner, implementer, verifier, rollout approver, scope, linked records and SLA children, trigger and T0, route and due date, dependencies and owners, effort assumptions, status, evidence, target and verified residual tier/status, risk authority, exception, reassessment trigger, and decision date. Status is *Proposed*, *Accepted*, *In progress*, *Blocked*, *Implemented pending verification*, *Verified*, *Deferred under exception*, *Risk accepted*, or *Superseded*. A missed date remains immutably *Overdue*; later exception does not erase it.

Every exception names GOV, rationale, exact scope and unmet requirement, tested interim controls, monitoring owner/cadence, issue and expiry dates, and automatic reopen triggers. It neither revises inherent risk nor permits a known uncontained Critical-equivalent path to keep processing untrusted input. Exceptions are finite.

After verification and rollout, GOV reassesses current residual risk under Section 1.8, records credited controls and a decision to accept, reduce, avoid, transfer, or escalate, and preserves the inherent rating. The result is a dated immutable snap-

shot, not a permanent closure claim. Phases 0–2 are reviewed at least weekly while open, Phase 3 at release gates, and Phase 4 at its cadence. Control failure, new incident/reproducer, changed release/default/trust/privilege/automation/sensitivity/dependency, test or telemetry regression, and exception expiry trigger reassessment and a new SLA child while preserving history.

D. CVE History Summary

Source, selection rule, and known limits of this table

Selection. The table enumerates every identifier naming HDF5 in a MITRE-index search retrieved 21 May 2026, supplemented by the HDF Group CVE test suite [72, 40]. It is not a curated defect list and retains marked duplicate, rejected, unreproducible, and non-HDF5 records. It is records-based evidence, not a prevalence estimate.

Retrieval date versus evidence cutoff. Five later records were added individually through the 27 August cutoff; the search was not rerun, so other records from that interval are outside this table. A cutoff-date rerun is an RM-1A census input.

The version-and-path gap. Except where the last column says otherwise, the table does not establish affected or fixed versions, fix commits, or reachability. Those fields are required to apply CORE-SEC-001/CORE-SEC-002 to a deployment and are explicit RM-1A deliverables. "Not established" is an evidence gap, not an unaffected-version determination.

Scope tags. The APP-S7 tag identifies the relevant boundary but assigns no rating or priority.

CWE classification. Tags classify the reported outcome, not the root cause, using the SSP SIG list and CWE [53]. Almost every classified row also manifests CWE-20 improper input validation (including CWE-1284–1287): the listed overflows, over-reads, and arithmetic faults are consequences of using file-derived values before validation. That recurring class is stated once rather than repeated in every row. Three records remain unclassified because the source is indeterminate, rejected, or not filed against HDF5.

Known defects in the source records. CVE-2025-6516 remains in two source-associated paths because whether they are one defect is unresolved. Rows grouping identifiers by root-cause summary may span different affected versions and must be split during RM-1A before deployment applicability is inferred.

Table 40 – HDF5 CVE history summary

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2026-34734	h5dump datatype conversion (H5T__conv_struct)	heap use-after-free CWE-416	a crafted file drives the tool's type-conversion path into a use-after-free. Reported with AddressSanitizer-detected termination; the advisory states that practical code-execution exploitability is unknown. This is the tool finding CORE-TOOL-001, not evidence that every libhdf5 consumer reaches the path.	HDF5-official-tool Affected: 1.14.1-2 and earlier per the advisory [28, 84]. Fixed version and commit not established (RM-1A).
CVE-2026-2492	TensorFlow HDF5 plugin search path	local privilege escalation / arbitrary code loading CWE-427	the HDF5 integration searched an unsecured location for plugins, so a low-privileged local user could place a library that loads with the target user's authority. Rated at the application boundary as APP-SEC-005; the mechanism is HDF5 dynamic plugin loading, the defect is the search-path policy.	HDF5-plugin-loading Affected: TensorFlow 2.17.0; vendor fix cited [59, 91, 79]. Not an HDF5-release defect; no HDF5 version range applies.
CVE-2026-1669	Keras model loading via HDF5 external dataset references	arbitrary local file read / disclosure CWE-73	an embedded external dataset reference selected a local path during model load, disclosing a file readable by the loader process. Rated at the application boundary as APP-SEC-003; external raw storage is a documented HDF5 capability, not a parser defect.	HDF5-external-reference Affected: Keras >=3.0.0,<3.12.1 and >=3.13.0,<3.13.2 [70, 64]. No HDF5 version range applies.

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2026-0897	Keras weight loading from `model.weights.h5`	memory exhaustion / interpreter crash `CWE-400`	a valid HDF5 member declared an extreme dataset shape and weight loading allocated from it without an adequate budget. Rated at the application boundary as `APP-SEC-004`; HDF5 metadata may legitimately declare a logical shape far larger than the useful payload.	`HDF5-downstream-resource-policy` Affected: Keras `>=3.0.0,<=3.12.0` and `>=3.13.0,<3.13.2` [69, 62]. No HDF5 version range applies.
CVE-2025-9905	Keras legacy `.h5/.hdf5` model loading	arbitrary code execution `CWE-502`	legacy HDF5 model loading did not honor `safe_mode=True`, so a `Lambda` layer's serialized code executed in the loader process. Rated at the application boundary as `APP-SEC-002`; this is application deserialization layered on HDF5, not an HDF5 parser finding.	`HDF5-downstream-deserialization` Affected: Keras 3.0.0 through 3.11.2 [68, 63]. No HDF5 version range applies.
CVE-2025-7069, CVE-2025-6270, CVE-2025-2914	file-space info / free-space manager	heap buffer overflow `CWE-787`	file-derived file-space metadata, especially page size / section data, was accepted without sufficient sanity checks. Fix rejects invalid page size and corrupted free-space info early.	`HDF5-core-parser` Fix evidence: early page-size sanity check [35]. Affected and fixed versions not established (`RM-1A`).
CVE-2025-6858, CVE-2025-6817, CVE-2025-2913, CVE-2025-2912	object-header / metadata-cache cascade	null dereference, resource consumption, use-after-free, heap overflow `CWE-787`, `CWE-416`, `CWE-476`, `CWE-400`	corrupted object-header continuation metadata caused later cache/free-list/object-message failures. Root cause is bad continuation-message size/state not rejected at decode time.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2025-6856, CVE-2025-6816, CVE-2025-2923	object-header continuation messages	heap buffer overflow / use-after-free `CWE-787`, `CWE-416`	crafted object header with continuation message pointing back to itself led to under-sized internal buffer allocation. Fix compares expected and actual object-header chunk counts during deserialization.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2025-6750, CVE-2024-33874	mtime object-header messages	heap buffer overflow CWE-787	old/new modification-time messages were not properly decoded; an invalid message size could produce a zero-size buffer passed to an encoder. Fix decodes both mtime formats and rejects invalid size.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-6269, CVE-2025-6516	metadata cache image / address decode	heap buffer overflow CWE-787	cache-image reconstruction used insufficient bounds and input validation. Fix adds buffer-size tracking, overflow checks, input validation, and cleanup on failure.	HDF5-core-parser Fix evidence: buffer-size tracking, overflow and input validation, cleanup [34]. Versions not established (RM-1A).
CVE-2025-44905, CVE-2025-2308, CVE-2024-29159, CVE-2024-32615, CVE-2016-4331	scale-offset / n-bit filters	heap overflow / buffer overrun / arbitrary code execution CWE-787, CWE-502	filter decoders trusted encoded precision, byte counts, or scale-offset parameters from the file. Root cause is insufficient bounds checking in decompression/filter parameter handling.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-44904, CVE-2024-32614, CVE-2024-32605, CVE-2024-29165, CVE-2019-9151, CVE-2018-14035, CVE-2018-13875, CVE-2018-11205	vectorized memory copy / scatter-fill paths	heap/stack overflow or over-read CWE-787, CWE-125	memory-vector copy routines copied between mismatched source/destination selections or trusted element counts too far. Root cause is insufficient validation of vector extents and copy sizes before memcpy-style operations.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2926	object-header cache checksum serialization	null pointer dereference CWE-476	object-header checksum/serialization path could operate on incomplete or invalid object-header state. Root cause is missing null/state validation before serialization.	HDF5-core-parser Versions and fix commit not established (RM-1A).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2025-2925	memory management wrapper	double free CWE-415	reallocation/error path mishandled ownership of mem. Root cause is ambiguous ownership/cleanup semantics around failed or edge-case realloc.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2924, CVE-2024-32613, CVE-2024-32612	local heap free-list deserialization	heap buffer overflow CWE-787	local-heap free-block metadata from the file was trusted. Root cause is insufficient validation of free-block size/offset before rebuilding heap structures.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2915, CVE-2018-13867	file accumulator	heap overflow / out-of-bounds read CWE-787, CWE-125	accumulator overlap/offset math accepted invalid ranges. Root cause is weak range validation for file accumulator reads/frees.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2310, CVE-2019-9152	datatype / attribute string decoding	heap buffer overflow / out-of-bounds read CWE-787, CWE-125	string duplication from metadata used file-derived length or unterminated data. Root cause is unsafe string length handling during datatype/attribute decode.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2309, CVE-2024-29163	datatype bit operations	heap buffer overflow CWE-787	bit-copy / bit-find logic trusted bit offsets, lengths, or precision metadata. Root cause is insufficient bounds checking in datatype bitfield operations.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-2153	shared-message deletion	heap buffer overflow CWE-787	shared-message deletion path could process corrupted shared-message/object-header state. Root cause appears to be insufficient validation of shared-message metadata and cleanup state before deletion.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2025-26200, CVE-2024-33877, CVE-2017-17507	compound datatype conversion	heap overflow / out-of-bounds read/write CWE-787, CWE-125	compound datatype conversion trusted complex datatype layout details. Root cause is unsafe datatype conversion when member layout, unused bits, or conversion metadata are malformed.	HDF5-core-parser Versions and fix commit not established (RM-1A).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2024-33876	point selection deserialization	heap overflow CWE-787	serialized point-selection metadata was trusted. Root cause is missing bounds/count validation when rebuilding point selections.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-33875, CVE-2019-8396	dataset layout encode	heap/buffer overflow CWE-787	dataset layout metadata could encode inconsistent dimensions/storage sizes. Fix family adds dataset layout size checks, multiplication overflow checks, and file-bound checks.	HDF5-core-parser Fix evidence: layout size, multiplication-overflow, and file-bound checks [36]. Versions not established (RM-1A).
CVE-2024-33873	dataset scatter memory	heap overflow CWE-787	scatter operation accepted inconsistent memory/file selection sizes. Root cause is insufficient validation of selected element counts and destination capacity.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32624, CVE-2024-32610	reference datatype memory nulling	heap overflow CWE-787	reference-memory initialization/nulling used malformed datatype/reference metadata. Root cause is missing size/type validation before clearing reference memory.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32623	generic array fill	heap overflow CWE-787	array fill operation trusted element count/size metadata. Root cause is missing multiplication/range validation before filling memory.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32622, CVE-2024-29158, CVE-2024-29161 note	free-list array allocation	out-of-bounds read / stack overflow CWE-787, CWE-125	array allocation/free-list path accepted invalid counts or sizes. Root cause is weak validation of allocation quantity derived from file metadata.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32621, CVE-2024-29162, CVE-2024-29157	global heap read	heap/stack overflow CWE-787	global heap object size/offset metadata could be inconsistent with actual buffer. Root cause is insufficient validation of heap object boundaries before read.	HDF5-core-parser Versions and fix commit not established (RM-1A).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2024-32620, CVE-2025-6516, CVE-2021-45830, CVE-2018-13866	address-length decode	heap/stack overflow / over-read CWE-787, CWE-125	encoded file addresses/lengths could exceed expected bounds. Root cause is insufficient validation of decoded address/length byte counts and destination capacity.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32619, CVE-2018-14031	datatype copy/reopen	heap overflow / over-read CWE-787, CWE-125	datatype copy path accepted malformed nested datatype metadata. Root cause is missing validation while copying or reopening datatype structures.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32618	native datatype conversion	heap overflow CWE-787	native-type construction trusted malformed datatype metadata. Root cause is inadequate validation during recursive datatype normalization.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32617	H5MM_xstrdup	heap overflow CWE-787	unsafe string duplication from file-derived metadata. Root cause is missing bounded string handling.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32616	datatype object-header encode helper	heap overflow CWE-787	datatype message encode helper accepted inconsistent datatype-message size/state. Root cause is missing encode-time size validation.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32611	attribute release table	uninitialized value / DoS CWE-908	attribute-table bookkeeping conflated allocated capacity and actual attribute count. Fix separates current count and maximum capacity and cleans partial tables on error.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32608, CVE-2024-32607	attribute close	memory corruption / segfault CWE-787	attribute close/release path could operate on partially initialized or corrupted attribute state. Root cause is unsafe cleanup after failed attribute-table construction or decode.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2024-32606	h5tools string printing	uninitialized dereference / DoS CWE-824	tool-side string formatting could dereference uninitialized values. No commit URL listed, so fix review was not possible.	HDF5-official-tool No commit URL in the source record; fix not reviewed. Versions not established (RM-1A).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2024-29166	link-info decode	buffer overrun CWE-787	object link-info metadata decode trusted malformed encoded size/flags. No commit URL listed.	HDF5-core-parser No commit URL in the source record; fix not reviewed. Versions not established (RM-1A).
CVE-2022-26061, CVE-2022-25972, CVE-2022-25942, CVE-2020-10809, CVE-2018-17436, CVE-2018-17433	gif2h5 tool / GIF parser	heap overflow, out-of-bounds write/read, invalid write CWE-787	legacy GIF conversion code parsed attacker-controlled GIF/HDF inputs unsafely. Commentary says the tool was removed rather than surgically hardened.	HDF5-legacy-tool Disposition: gif2h5 removed rather than hardened. Removing release not verified (RM-1A).
CVE-2021-46244	datatype copy completion	divide by zero / DoS CWE-369	datatype copy logic failed to guard arithmetic denominators or zero-sized datatype state.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2021-46243	datatype object-header decode	untrusted pointer dereference / DoS CWE-822	datatype decode accepted malformed pointer/metadata state. Root cause is missing validation before dereference.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2021-46242, CVE-2020-10810	metadata cache pin/unpin	use-after-free / null dereference CWE-416, CWE-476	cache entry lifecycle operations did not robustly handle malformed or unexpected cache state.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2021-45833	chunk file-map hyperslab	stack overflow / DoS CWE-787	chunk/hyperslab mapping trusted malformed dataspace/chunk metadata. Root cause is inadequate bounds checking in chunk mapping.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2021-45832	error stack internals	stack overflow / DoS CWE-787	listed as cannot reproduce; no fix commit. Treat root cause as unconfirmed.	HDF5-core-parser Disposition: recorded as cannot reproduce; no fix commit. Root cause unconfirmed.

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2021-45829	unspecified HDF5 path	segfault / DoS *not classified*	Root cause undetermined.	`HDF5-core-parser` Disposition: root cause undetermined; no fix commit.
CVE-2021-37501	`h5dump` string sprint	buffer overflow / DoS `CWE-787`	h5dump formatting path wrote beyond buffer while rendering file-derived data.	`HDF5-official-tool` Versions and fix commit not established (`RM-1A`).
CVE-2021-36977	mat file I/O library using HDF5	heap overflow `CWE-787`	not filed against HDF5. Root cause belongs to downstream integration/context, not the HDF5 fix set.	`not-HDF5` Not filed against HDF5; downstream integration of HDF5 1.12.0 as reported. No HDF5 fix set applies.
CVE-2021-31009	Apple platform issue	multiple HDF5 issues *not classified*	not filed against HDF5; fixed by Apple by removing HDF5 from affected components.	`not-HDF5` Not filed against HDF5; vendor addressed it by removing HDF5 from the affected components.
CVE-2020-18494, CVE-2020-18232	dataspace close	buffer overflow / possible RCE `CWE-787`	malformed file could corrupt dataspace cleanup/close state. Root cause is unsafe lifecycle handling of malformed dataspace metadata.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2020-10812	file reference count query	null pointer dereference / DoS `CWE-476`	file query path failed to validate file/reference state before dereference.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2020-10811, CVE-2018-14033	layout decode	heap over-read `CWE-125`	dataset layout message decoder trusted encoded layout fields. Root cause is missing bounds checks around decoded layout data.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2019-8398, CVE-2019-8397	datatype size/close	out-of-bounds read `CWE-125`	datatype object state could be malformed before size/close operations. No commit URL listed.	`HDF5-core-parser` No commit URL in the source record; fix not reviewed. Versions not established (`RM-1A`).
CVE-2018-17439	dataspace extent dimensions during HDF to GIF conversion	stack overflow `CWE-787`	conversion tool path trusted dataspace dimensions from file.	`HDF5-legacy-tool` Disposition: HDF-to-GIF conversion path removed. Removing release not verified (`RM-1A`).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2018-17438	selected I/O	divide by zero / DoS CWE-369	selected-I/O path failed to guard divisor/count metadata from crafted file. Tool-removal commit, so the listed fix is coarse.	HDF5-core-parser Disposition: tool-removal commit, so the listed fix is coarse and does not establish parser-path closure.
CVE-2018-17437	datatype decode helper	memory leak / DoS CWE-401	error path leaked allocations when datatype metadata was malformed.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-17435	attribute decode	heap over-read CWE-125	attribute message decoder trusted encoded sizes/offsets.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-17434	h5repack filter application	divide by zero / DoS CWE-369	tool-side filter setup failed to guard zero-valued filter/chunk parameters.	HDF5-official-tool Versions and fix commit not established (RM-1A).
CVE-2018-17432	simple dataspace encode	null pointer dereference / DoS CWE-476	dataspace encode path accepted invalid dataspace state before dereference.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-17237, CVE-2018-17233, CVE-2018-11207, CVE-2018-11203, CVE-2017-17508	chunking / b-tree / datatype arithmetic	divide by zero / SIGFPE / DoS CWE-369	crafted file could set chunk, b-tree, datatype, or layout parameters to zero. Root cause is missing arithmetic precondition checks.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-17234, CVE-2018-13873, CVE-2018-11204	object-header chunk deserialize	memory leak / over-read / null dereference CWE-125, CWE-476, CWE-401	object-header chunk deserialization accepted malformed chunk metadata and mishandled error cleanup.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-16438	external links	out-of-bounds read CWE-125	external-link query trusted malformed external-link metadata.	HDF5-core-parser Versions and fix commit not established (RM-1A).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2018-15672, CVE-2018-14032	duplicate/rejected IDs	not applicable *not classified*	rejected duplicate CVEs; no HDF5 root cause to summarize beyond the referenced canonical CVEs.	`duplicate/rejected` Rejected duplicate identifiers; no HDF5 root cause or fix set beyond the canonical CVEs.
CVE-2018-15671	property-list callback lookup	excessive stack consumption / DoS `CWE-674`	recursive or malformed property-list callback state allowed stack exhaustion.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2018-14460	simple dataspace decode	heap over-read `CWE-125`	dataspace message decoder trusted encoded extent/rank metadata.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2018-14034	filter pipeline reset	out-of-bounds read `CWE-125`	malformed filter-pipeline metadata caused unsafe reset/read behavior. No commit URL listed.	`HDF5-core-parser` No commit URL in the source record; fix not reviewed. Versions not established (`RM-1A`).
CVE-2018-13876, CVE-2018-13874	sec2 VFD read path	stack overflow `CWE-787`	low-level file-driver read/memset path could be driven out of bounds by malformed file metadata. No commit URL listed.	`HDF5-core-parser` No commit URL in the source record; fix not reviewed. Versions not established (`RM-1A`).
CVE-2018-13872	group entry decode	heap overflow `CWE-787`	group-entry decoder trusted malformed symbol-table/group metadata. No commit URL listed.	`HDF5-core-parser` No commit URL in the source record; fix not reviewed. Versions not established (`RM-1A`).
CVE-2018-13871	free-list block allocation	heap overflow `CWE-787`	free-list block allocator accepted invalid allocation size/count.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2018-13870, CVE-2018-13869	link decode	heap over-read / overlapping memcpy `CWE-125`	link-message decoder trusted malformed link metadata, including overlapping source/destination regions.	`HDF5-core-parser` Versions and fix commit not established (`RM-1A`).
CVE-2018-13868	old fill-value decode	heap over-read `CWE-125`	old fill-value decoder trusted encoded fill-value size. No commit URL listed.	`HDF5-core-parser` No commit URL in the source record; fix not reviewed. Versions not established (`RM-1A`).

Table 40 – HDF5 CVE history summary, continued

CVE(s)	Affected area	Reported failure mode	Root cause summary	Scope tag; version and fix evidence
CVE-2018-11206	fill-value decode	out-of-bounds read / possible disclosure CWE-125, CWE-73	fill-value message decoders trusted encoded size/type fields.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2018-11202	hyperslab spans	null pointer dereference / DoS CWE-476	hyperslab span construction failed to validate malformed selection state.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2017-17509	group cache entry decode vector	out-of-bounds write CWE-787	group-entry vector decoder trusted file-derived counts/array bounds. No commit URL listed.	HDF5-core-parser No commit URL in the source record; fix not reviewed. Versions not established (RM-1A).
CVE-2017-17506, CVE-2017-17505	filter pipeline decode	out-of-bounds read / null dereference CWE-125, CWE-476	pipeline decoder failed to validate malformed filter counts or fields. No commit URL listed.	HDF5-core-parser No commit URL in the source record; fix not reviewed. Versions not established (RM-1A).
CVE-2016-4333	array allocation / initialization loop	out-of-bounds write CWE-787	file-controlled value could modify loop termination while initializing an allocated array.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2016-4332	object-header message flags	heap out-of-bounds write / code execution CWE-787	library failed to check whether a message type supported a flag before casting to an incompatible structure.	HDF5-core-parser Versions and fix commit not established (RM-1A).
CVE-2016-4330	dataspace array rank/dimensions	heap overflow / code execution CWE-787	file-supplied array dimensionality was not checked against allocated space.	HDF5-core-parser Affected: 1.8.16 as reported by the primary analysis [26]. Fixed version and commit not established (RM-1A).

E. Software Supply Chain Details

The HDF5_ALLOW_EXTERNAL_SUPPORT modes apply to filter-related dependencies. Some test and example configurations have their own source-fetch paths, such as KWSys for the API test driver and h5py for Python examples. MPI, HDFS, ROS3, OpenSSL, Java, documentation, packaging, and developer tools must be provided by the build environment.

Table 41 – Dependency source modes

Mode	Relevant options	Meaning
System packages	HDF5_ALLOW_EXTERNAL_SUPPORT=NO	CMake uses find_package(), find_library(), or find_program() to locate installed dependencies. This is the default.
Git fetch	HDF5_ALLOW_EXTERNAL_SUPPORT=GIT plus a dependency-specific *_USE_EXTERNAL=ON option	CMake downloads source from the configured *_GIT_URL and *_GIT_TAG values and builds it with HDF5.
Tarball fetch	HDF5_ALLOW_EXTERNAL_SUPPORT=TGZ plus a dependency-specific *_USE_EXTERNAL=ON option	CMake downloads from *_TGZ_ORIGPATH and *_TGZ_NAME, or uses TGZPATH when the dependency-specific *_USE_LOCALCONTENT=ON option is set.

Table 42 – Baseline build

Dependency	Trigger	Notes
CMake 3.26 or newer	Always	Required by the top-level cmake_minimum_required().
C compiler and native build tool	Always	The top-level project is a C project.
Platform C library, headers, and system calls	Always	CMake probes standard headers, types, functions, byte order, large-file support, file locking, pread/pwrite, and related platform features.
Platform link libraries	Platform-dependent	CMake adds libraries only when probes show they are needed: m, dl, rt, posix4, ucb, ws2_32, wsock32, and shlwapi.

PkgConfig	Optional probe	Used only when available. It can help some find modules, such as zlib-ng and libaec, but it is not a hard requirement for the default build.
Threads	Found in every configure; required by thread features	CMake searches for C11 threads, Win32 threads, or pthreads. When found, HDF5 records thread support and links `Threads::Threads`; `HDF5_ENABLE_THREADSAFE`, `HDF5_ENABLE_CONCURRENCY`, and `HDF5_ENABLE_SUBFILING_VFD` require it.

Table 43 – HDF5 feature triggers

Feature or component	Build options	Dependency	Notes
Parallel HDF5	`HDF5_ENABLE_PARALLEL=ON`	MPI C with MPI-IO, MPI standard 3.0 or newer	Adds `MPI::MPI_C` to public link libraries. Parallel filtered writes and large parallel I/O are enabled only when the required MPI symbols are present. Parallel tests also need an MPI launcher such as `mpiexec`.
Parallel Fortran	`HDF5_ENABLE_PARALLEL=ON`, `HDF5_BUILD_FORTRAN=ON`	MPI Fortran	The Fortran build adds MPI Fortran libraries. The `mpi_f08` module is used when available.
Thread-safe library	`HDF5_ENABLE_THREADSAFE=ON`	Threads	Unsupported with parallel, Fortran, C++, or high-level library unless `HDF5_ALLOW_UNSUPPORTED=ON`. On Windows, static libraries are rejected.

Multi-threaded concurrency	HDF5_ENABLE_CONCURRENCY=ON	Threads	Experimental. Mutually exclusive with HDF5_ENABLE_THREADSAFE; has the same unsupported combinations as thread-safety unless overridden with HDF5_ALLOW_UNSUPPORTED=ON.
C++ wrappers	HDF5_BUILD_CPP_LIB=ON	C++ compiler	Unsupported with parallel unless HDF5_ALLOW_UNSUPPORTED=ON. High-level C++ wrappers also require HDF5_BUILD_HL_LIB=ON.
Fortran wrappers	HDF5_BUILD_FORTRAN=ON	Fortran compiler with Fortran 2003 ISO_C_BINDING support	Unsupported with thread-safety or concurrency unless HDF5_ALLOW_UNSUPPORTED=ON. High-level Fortran wrappers also require HDF5_BUILD_HL_LIB=ON.
Java wrappers	HDF5_BUILD_JAVA=ON	Java/JDK, shared HDF5 libraries, Java dependency JARs	Java requires shared HDF5 libraries. Java 11 or newer is required for the JNI implementation. Java tests additionally require the JUnit and Hamcrest JARs.
Java JNI implementation	HDF5_BUILD_JAVA=ON, HDF5_ENABLE_JNI=ON, or any Java version earlier than 25	JNI headers and libraries	JNI is the default Java implementation. HDFS also requires JNI.
Java FFM implementation	HDF5_BUILD_JAVA=ON, Java 25 or newer, HDF5_ENABLE_JNI=OFF	jextract from JEXTRACT_HOME/bin or JAVA_HOME/bin	Generates FFM bindings at configure time.

Java logging/test JARs	`HDF5_BUILD_JAVA=ON`	`slf4j-api`, `slf4j-nop`, `slf4j-simple`; plus JUnit 4 and Hamcrest for tests	The default paths point to bundled JARs under `java/lib`. They can be overridden with `HDF5_JAVA_*_JAR` cache variables. `HDF5_ENABLE_MAVEN_DEPLOY=ON` generates Maven-style artifacts and POM files; the CMake build does not locate or run `mvn`.
Documentation	`HDF5_BUILD_DOC=ON`	Doxygen	`HDF5_ENABLE_DOXY_WARNINGS=ON` makes Doxygen warnings fail the build.
Header regeneration	`HDF5_GENERATE_HEADERS=ON`	Perl	Perl is optional otherwise. If it is missing, generated headers are not regenerated.
Python examples	`H5EXAMPLE_BUILD_PYTHON=ON`	Python3 interpreter, Python development files, NumPy, h5py source, pip	The examples CMake project fetches `h5py` from Git and installs it with `python3 -m pip`.

Table 44 – HDF5 filter dependency choices

Feature	Build options	Dependency	Source choices
DEFLATE filter	`HDF5_ENABLE_ZLIB_SUPPORT=ON`, `HDF5_USE_ZLIB_NG=OFF`	zlib	System package by default; `ZLIB_USE_EXTERNAL=ON` can build zlib from Git or TGZ. `HDF5_USE_ZLIB_STATIC=ON` prefers a static library.

DEFLATE filter through zlib-ng	HDF5_ENABLE_ZLIB_SUPPORT=ON, HDF5_USE_ZLIB_NG=ON	zlib-ng	System package by default; ZLIB_USE_EXTERNAL=ON can build zlib-ng from Git or TGZ. zlib compatibility mode is detected.
SZIP filter	HDF5_ENABLE_SZIP_SUPPORT=ON	libaec with libsz compatibility, or another package selected by LIBAEC_PACKAGE_NAME	System package by default; SZIP_USE_EXTERNAL=ON can build libaec from Git or TGZ. HDF5_USE_LIBAEC_STATIC=ON prefers static libraries.
HDF5 filter plugins project	HDF5_ENABLE_PLUGIN_SUPPORT=ON	Installed h5pl/HDF5 filter plugins package, or the hdf5_plugins project	PLUGIN_USE_EXTERNAL=ON can build the plugin project from Git or TGZ. Plugin support is disabled for the 1.6 API.

Table 45 – HDF5 VFD and VOL connector dependency choices

Feature	Build options	Dependency	Notes
Direct VFD	HDF5_ENABLE_DIRECT_VFD=ON	POSIX O_DIRECT and posix_memalign() support	No third-party library. Configuration fails if the platform support is missing.
Mirror VFD	HDF5_ENABLE_MIRROR_VFD=ON	POSIX socket/fork headers and functions	No third-party library. Requires netinet/in.h, netdb.h, arpa/inet.h, sys/socket.h, and fork().

ROS3 VFD	`HDF5_ENABLE_ROS3_VFD=ON`	`aws-c-s3` package	CMake	HDF5 does not fetch this library. Building `aws-c-s3` from source can also require AWS C libraries such as `aws-c-common`, `aws-checksums`, `aws-c-cal`, `aws-c-io`, `aws-c-compression`, `aws-c-http`, `aws-c-sdkutils`, `aws-c-auth`, and, on Linux, `aws-lc` and `s2n-tls`. ROS3 docker-proxy tests additionally require `docker` and the AWS CLI.
HDFS VFD	`HDF5_ENABLE_HDFS=ON`	JNI/JVM and Hadoop `libhdfs`		`HADOOP_HOME` is used to locate `hdfs.h`, `libhdfs`, and the Hadoop version command. Non-MSVC builds also add `-pthread`.
Subfiling VFD and IOC VFD	`HDF5_ENABLE_PARALLEL=ON`, `HDF5_ENABLE_SUBFILING_VFD=ON`	MPI C with `MPI_Comm_split_type`, plus Threads		Not available on Windows. IOC VFD is built when subfiling is enabled.
External VOL connectors	`HDF5_VOL_ALLOW_EXTERNAL=GIT` or `LOCAL_DIR` with `HDF5_VOL_URLnn` or `HDF5_VOL_PATHnn`	Connector-defined		Up to ten CMake-based VOL connectors can be added. HDF5 patches connector CMake code to use the in-tree HDF5 build, but each connector may bring its own dependencies.

Table 46 – Tools, Tests, and Security Features

Feature	Build options	Dependency	Notes
Signed plugin verification	HDF5_REQUIRE_SIGNED_PLUGINS=ON	OpenSSL libcrypto 1.1.0 or newer	Adds OpenSSL::Crypto to HDF5. HDF5_LOCK_PLUGIN_KEYSTORE=ON also requires HDF5_PLUGIN_KEYSTORE_DIR.
h5sign tool	HDF5_REQUIRE_SIGNED_PLUGINS=ON, HDF5_BUILD_TOOLS=ON	OpenSSL libcrypto	Used to sign HDF5 plugins.
Signed-plugin tests	HDF5_REQUIRE_SIGNED_PLUGINS=ON, BUILD_TESTING=ON	openssl executable	Tests generate RSA keys with the OpenSSL command-line tool.
Parallel tools utilities	HDF5_ENABLE_PARALLEL=ON, HDF5_BUILD_PARALLEL_TOOLS=ON	MFU, CIRCLE, DTCMP	CMake looks for headers and libraries, using MFU_ROOT as a hint.
API test driver	BUILD_TESTING=ON, HDF5_TEST_API_ENABLE_DRIVER=ON	KWSys	The test driver fetches KWSys from a tarball or from TGZPATH when KWSYS_USE_LOCALCONTENT=ON.
Shell-script tests	BUILD_TESTING=ON	PowerShell or Bash	CMake uses pwsh/powershell when available; otherwise Unix builds look for bash.
ROS3 VFD tests	HDF5_ENABLE_ROS3_VFD=ON, HDF5_ENABLE_ROS3_VFD_DOCKER_PROXY=ON	Docker daemon and AWS CLI	Used to run S3 proxy tests.

Table 47 – Developer and Packaging Dependencies

Configuration	Build options	Dependency
Analyzer tooling	`HDF5_ENABLE_ANALYZER_TOOLS=ON`	`clang-tidy`, `include-what-you-use`, and `cppcheck` are probed and used by the analyzer macros.
Source formatting targets	`HDF5_ENABLE_FORMATTERS=ON`	`clang-format` and `cmake-format`.
Code coverage with Clang/IntelLLVM	`HDF5_ENABLE_COVERAGE=ON`, `CODE_COVERAGE=ON`	`llvm-cov` and `llvm-profdata`; AppleClang uses `xcrun llvm-cov` and `xcrun llvm-profdata`.
Code coverage with GCC	`HDF5_ENABLE_COVERAGE=ON`, `CODE_COVERAGE=ON`	`lcov` and `genhtml`; fallback instrumentation can also link `gcov`.
Sanitizers	`HDF5_ENABLE_SANITIZERS=ON`, `HDF5_USE_SANITIZER=<kind>`	Compiler sanitizer runtime support. Supported values depend on compiler and platform.
AFL support module	Explicit inclusion of `config/sanitizer/afl-fuzzing.cmake`	`afl-cc` and `afl-c++`. This module is available but is not included by the top-level HDF5 build by default.
Dependency graph support module	Explicit inclusion of `config/sanitizer/dependency-graph.cmake`	Graphviz `dot`. This module is available but is not included by the top-level HDF5 build by default.
Windows installers	CPack on Windows	NSIS and WiX are probed when available.
Linux packages	CPack on Unix-like systems	`dpkg-shlibdeps` enables DEB generation; `rpmbuild` enables RPM generation for non-parallel builds.

Table 48 – External Source Defaults

Dependency	Default version/tag	Default source

zlib	1.3.2 / v1.3.2	https://github.com/madler/zlib.git or https://github.com/madler/zlib/releases/download/v1.3.2/zlib-1.3.2.tar.gz
zlib-ng	2.3.3	https://github.com/zlib-ng/zlib-ng.git or https://github.com/zlib-ng/zlib-ng/archive/refs/tags/2.3.3.tar.gz
libaec	1.1.6 / v1.1.6	https://github.com/MathisRosenhauer/libaec.git or https://github.com/MathisRosenhauer/libaec/releases/download/v1.1.6/libaec-1.1.6.tar.gz
HDF5 filter plugins	master	https://github.com/HDFGroup/hdf5_plugins.git or https://github.com/HDFGroup/hdf5_plugins/releases/download/snapshot/hdf5_plugins-master.tar.gz
KWSys API-test helper	master tarball	https://gitlab.kitware.com/utils/kwsys/-/archive/master/kwsys-master.tar.gz
h5py Python examples	master	https://github.com/h5py/h5py.git
bitshuffle	master or 0.5.2 tarball	https://github.com/kiyo-masui/bitshuffle.git or https://github.com/kiyo-masui/bitshuffle/archive/refs/tags/bitshuffle-0.5.2.tar.gz
Blosc	1.21.6	https://github.com/Blosc/c-blosc.git or https://github.com/Blosc/c-blosc/archive/refs/tags/c-blosc-1.21.6.tar.gz
Blosc zlib helper	develop or 1.3.1 tarball	https://github.com/madler/zlib.git or https://github.com/madler/zlib/releases/download/v1.3.1/zlib-1.3.1.tar.gz
Blosc2	2.17.1	https://github.com/Blosc/c-blosc2.git or https://github.com/Blosc/c-blosc2/archive/refs/tags/c-blosc2-2.17.1.tar.gz

Blosc2 zlib helper	`develop` or `1.3.1` tarball	https://github.com/madler/zlib.git or https://github.com/madler/zlib/releases/download/v1.3.1/zlib-1.3.1.tar.gz
bzip2	`bzip2-1.0.8`	https://github.com/libarchive/bzip2.git or https://github.com/libarchive/bzip2/archive/refs/tags/bzip2-bzip2-1.0.8.tar.gz
fpzip	`develop` or `1.3.0` tarball	https://github.com/LLNL/fpzip.git or https://github.com/LLNL/fpzip/releases/download/1.3.0/fpzip-1.3.0.tar.gz
JPEG	`jpeg-9e`	https://github.com/libjpeg-turbo/libjpeg-turbo.git or https://www.ijg.org/files/jpegsrc.v9e.tar.gz
LZ4	`dev` or `1.10.0` tarball	https://github.com/lz4/lz4.git or https://github.com/lz4/lz4/releases/download/v1.10.0/lz4-1.10.0.tar.gz
LZF	`3.6` tarball	http://dist.schmorp.de/liblzf/liblzf-3.6.tar.gz
SZ	`master` or `2.1.12.5` tarball	https://github.com/szcompressor/SZ.git or https://github.com/szcompressor/SZ/releases/download/v2.1.12.5/SZ-2.1.12.5.tar.gz
ZFP	`develop` or `1.0.1` tarball	https://github.com/LLNL/zfp.git or https://github.com/LLNL/zfp/releases/download/1.0.0/zfp-1.0.1.tar.gz
Zstandard	`1.5.7`	https://github.com/facebook/zstd.git or https://github.com/facebook/zstd/releases/download/v1.5.7/zstd-1.5.7.tar.gz

F. Resources and Supporting Tooling

The artifacts below are referenced by recommendations in this report as existing starting points. Listing a project here identifies a resource; it is not an assessment of that project, and none was audited.

HDF5 Safety, Security & Privacy (SSP) Special Interest Group [46].
Governance, process, and artifact home for the work this report feeds. It is the natural venue for the roadmap adoption decision described in Appendix C, and for publishing the `CORE-S12` metrics dashboard.

HDF5 CVE test suite [40].
An existing collection of vulnerability-triggering inputs. It is the obvious seed for the permanent corpus proposed in `CORE-S6`, which additionally requires per-case expected-outcome metadata and execution in both assertion-enabled and `-DNDEBUG` builds. It also contributed to the CVE enumeration in Appendix D.

`h5lens`: a machine-readable specification of the HDF5 file format [30].
Directly relevant to three recommendations. It is a candidate source of truth for the invariant registry in `CORE-S1`; a generator for the positive and negative fixtures required by `IND-S1`; and the origin of issue #87, the assert-masking observation tracked as `CORE-SEC-003`.

HDF5 extension skeletons [41].
Templates for VOL connectors, filters, and VFDs. A natural place to embed the privacy-documentation template required by `FILTER-S5` and the trust and search-path defaults required by `VOL-S3`, `FILTER-S1`, and `VFD-S4`, so that new extensions inherit them rather than retrofitting them.

HDF5 plugins [43].
The filter-plugin project whose options govern the dependency set discussed in Section 6 and Appendix E. It is the scope boundary for the plugin manifest and allowlist proposed in `CORE-S10`.

G. Glossary

This appendix explains specialized vocabulary used in the report for readers who do not work in security, privacy, software assurance, or HDF5 development. Definitions describe how a term is used *in this report*; they do not add a finding, change a rating, or replace the controlling scope and decision rules in Sections 1.3–1.8. Product and project names are generally omitted unless the name itself is used as technical shorthand.

G.1. Abbreviations

Short form	Expansion	Meaning in this report
ABI	Application Binary Interface	The low-level contract by which compiled software calls a library, including calling conventions, data layout, and exported symbols. Two builds can expose similar source APIs but still be ABI-incompatible.
AFL	American Fuzzy Lop	A coverage-guided fuzzing tool family. The build appendix mentions AFL support as one available test configuration, not as evidence that every parser path was fuzzed.
AI/ML	Artificial Intelligence / Machine Learning	Application domains discussed here chiefly because model loaders and model artifacts can cross an HDF5 trust boundary.
API	Application Programming Interface	The functions, types, and rules through which software uses a library or service. The HDF5 C API is distinct from the HDF5 on-disk format.
ASan	AddressSanitizer	A runtime testing tool that detects many invalid memory accesses, including some buffer overflows and uses of freed memory.
CI / CI/CD	Continuous Integration / Continuous Integration and Delivery	Automated building and testing of changes; CD adds automated packaging or delivery stages. Passing CI is evidence only for the configurations and tests that actually ran.

Short form	Expansion	Meaning in this report
CLI	Command-Line Interface	A program operated from a shell, such as `h5dump` or `h5repack`.
CPU	Central Processing Unit	The processor resource that can be exhausted by excessive or non-terminating work.
CVE	Common Vulnerabilities and Exposures	A public identifier for a reported vulnerability record. A CVE identifier does not by itself establish root cause, affected HDF5 versions, reachability in a particular application, or this report's risk rating.
CVSS	Common Vulnerability Scoring System	A standardized vulnerability-severity scoring method. This report does not convert CVSS scores into its impact/exposure ratings.
CWE	Common Weakness Enumeration	A catalog of software weakness classes. For example, `CWE-20` denotes improper input validation; a CWE describes a class, not a particular affected release.
CWD	Current Working Directory	The process directory used to resolve many relative paths. Implicit CWD lookup can make an external-file or plugin search depend on who launched the process and from where.
DNS	Domain Name System	The network service that resolves names to addresses. DNS queries and logs can expose which external data services a workflow contacts.
DoS	Denial of Service	Loss of availability caused by a crash, hang, excessive work, exhausted resources, or another condition that prevents useful service.
ENOSPC	No space left on device	The operating-system error reported when a storage target cannot accept more data. The report also uses it as shorthand for HDF5's broader out-of-space write, flush, and close problem.
FFM / JDK	Foreign Function and Memory API / Java Development Kit	The JDK supplies the Java compiler and runtime tools; its FFM API lets Java code call native libraries and access foreign memory. HDF5 can optionally generate Java FFM bindings.

Short form	Expansion	Meaning in this report
GHSA	GitHub Security Advisory	A security-advisory record and identifier published through GitHub. Like a CVE, it must be checked for exact scope, evidence, and applicability.
HDF / HDF5	Hierarchical Data Format / Hierarchical Data Format version 5	In this report, HDF5 refers to the version-5 format and data model and, only where specifically stated, the core C library, official CLI tools, extension classes, and build or release configuration. HDF4 and older HDF technology are different formats and software.
H5II / H5PW	HDF5 Independent Implementation / HDF5 Python Wrapper	Names of two SSP SIG mechanism models used in the independent-implementation and binding chapters. Their numbered families classify mechanisms rather than assigning this report's risk tiers.
HDFS	Hadoop Distributed File System	A distributed storage system that can be reached through an optional HDF5 VFD and therefore adds build, storage, and deployment dependencies.
HPC	High-Performance Computing	Computing environments such as clusters and supercomputers, often involving parallel I/O, MPI, shared storage, and tightly matched software builds.
HSDS	Highly Scalable Data Service	An HDF5-oriented service shown for ecosystem context but expressly outside the assessment scope.
HTTP	Hypertext Transfer Protocol	A network protocol used to retrieve resources. HTTP requests, range requests, proxies, and access logs can expose data locations and access patterns.
ID	Identifier	A name or code used to distinguish a record, object, process resource, or other entity. Report identifiers and HDF5 object identifiers belong to different namespaces and must be interpreted in context.

Short form	Expansion	Meaning in this report
I/O	Input / Output	Reading or writing data between a process and files, memory, devices, or network services.
JNI / JVM	Java Native Interface / Java Virtual Machine	JNI connects Java code to native libraries; the JVM executes Java bytecode. Their boundary matters when Java-facing software reaches native HDF5 code.
LSan	LeakSanitizer	A runtime testing tool that detects allocated memory that becomes unreachable without being released.
LRU	Least Recently Used	A cache-management policy that preferentially evicts entries not used recently. The report mentions LRU rank in one metadata-cache invariant case.
MPI / MPI-I/O	Message Passing Interface / MPI input and output	Standards used for parallel programs and coordinated parallel file access. Parallel HDF5 builds depend on compatible MPI behavior and binaries.
N/A	Not applicable	A report status for an item that is not a risk-bearing finding, such as a recommendation or program theme. It does not mean that a risk was evaluated as zero.
`NDEBUG` / `-DNDEBUG`	Assertions disabled	The C/C++ convention and compiler definition that remove ordinary `assert(...)` checks. This commonly occurs in release builds.
NE	Not evaluated	A rating status used when impact or exposure lacks enough evidence. NE never means Low risk.
OSS	Open-Source Software	Software whose source is available under an open-source license. Openness supports inspection and reuse but is not, by itself, a security guarantee.
OSS-Fuzz	Continuous fuzzing service for open-source software	A service that repeatedly runs project fuzz targets at scale. Its results cover the submitted targets and configurations rather than every possible HDF5 path.

Short form	Expansion	Meaning in this report
POSIX	Portable Operating System Interface	A family of operating-system interface standards. POSIX calls and semantics underlie some HDF5 file drivers and candidate storage-reservation controls.
PR	Pull Request	A proposed source change submitted for review and automated checks before merging. A PR test gate provides fast feedback but not long-running assurance.
RCE	Remote Code Execution	Execution of attacker-chosen code across a remote trust boundary. The broader phrase *arbitrary code execution* does not necessarily imply a remote attacker.
REST	Representational State Transfer	A common style of network API that some connectors or external services may use.
ROS3	Read-Only S3	The HDF5 Virtual File Driver for read-only access to objects through an S3 or S3-compatible interface. Its network, credential, and service behavior remain part of the deployment boundary.
SBOM	Software Bill of Materials	A machine-readable inventory of software components and versions in a build or package. It supports applicability analysis but does not prove that a vulnerability is reachable or fixed.
S3	Amazon Simple Storage Service, or an S3-compatible interface	An object-storage API used by cloud services and by HDF5's ROS3 VFD for read-only access. The governing provider need not be Amazon when an S3-compatible service is used.
SDLC	Software Development Life Cycle	The processes used to design, implement, test, release, operate, and maintain software.
SIG	Special Interest Group	A community group focused on a defined subject. The SSP SIG is the HDF5 Safety, Security & Privacy Special Interest Group.
SIGFPE	Floating-point exception signal	An operating-system signal commonly raised by invalid integer or floating-point arithmetic, including the division-by-zero case discussed in the report.

Short form	Expansion	Meaning in this report
SLA	Service-Level Agreement	A documented operating commitment, such as a response target. The proposed roadmap calls for SLA child records to make clocks and evidence accountable; those records do not create a risk rating or show that the proposal was adopted.
SLSA	Supply-chain Levels for Software Artifacts	A framework for improving and describing the integrity and provenance of software builds and releases.
SSP	Safety, Security & Privacy	The three assessment dimensions and the name of the HDF5 SSP SIG. Their meanings in this report are separately defined below.
SWMR	Single Writer / Multiple Reader	An HDF5 access mode that permits one writer and multiple readers to access a file concurrently under SWMR's required open, flush, and refresh protocol and filesystem semantics. It is not a general crash-recovery or multi-writer guarantee.
`T0` / TBD	Clock start / To be determined	`T0` is the recorded start date for a finding-and-scope or work-package clock. Depending on its stated trigger, it is affected-scope or equivalent-exposure identification, or roadmap adoption; the record must say which. TBD marks a field for which a required decision has not yet been made.
UBSan	Undefined Behavior Sanitizer	A runtime testing tool that detects selected C/C++ operations whose behavior the language does not define, such as some invalid arithmetic.
UAF	Use After Free	A memory-lifetime error in which code accesses an object after releasing its storage. The full term is defined in the concepts below.
URL	Uniform Resource Locator	An address for a network resource. URLs in metadata, logs, or connector configuration can themselves reveal sensitive context.

Short form	Expansion	Meaning in this report
VDS	Virtual Dataset	An HDF5 dataset whose logical view maps to regions of source datasets, possibly in other files. Reading it can therefore cause external-file access.
VFD	Virtual File Driver	The HDF5 layer that maps logical file reads and writes to physical storage, such as a local file, memory, multiple files, MPI-I/O, or object storage.
VLEN	Variable Length	An HDF5 datatype whose elements have variable size. VLEN data require dynamic allocation and careful lifetime and resource handling.
VOL	Virtual Object Layer	The HDF5 layer through which a connector implements object operations. A VOL connector can present remote, generated, or non-HDF5 data through the HDF5 API.

G.2. Report identifiers and notation

Identifiers are labels, not scores. A recommendation number does not imply priority, and a roadmap phase does not change a finding's risk. The letter `I` is context-sensitive: `I1–I4` mean impact levels in a risk record, while `I1–I9` in the independent-implementation chapter name mechanism families. The surrounding table or model determines which meaning applies.

Pattern or label	Meaning
`CORE`, `APP`, `EXT`, `FMT`, `IND`, `LONG`, `SUP`	Report component prefixes for the core library or tools, downstream applications, extensions, format governance, independent implementations, long-term accessibility, and the software supply chain.
`SEC`, `SAFE`/`SAF`, `PRIV`, `TOOL`	Report dimension or surface labels for security, safety, privacy, and an official tool.
`<component>-<dimension>-nnn`	A report finding or evidence-record identifier, for example `CORE-SEC-001`. The register states whether the record is formally rated or remains in the evidence backlog.

Pattern or label	Meaning
<component>-Snn	A recommendation identifier, for example APP-S4. Variants such as FILTER-S1, VFD-S4, and CORE-PRIV-S2 identify a more specific track. The number is an identifier only.
RM-nX	A phased-roadmap work package. Its phase and sequence describe delivery planning, not severity or inherent risk.
I1–I4 and E1–E4	The report's ordinal impact and exposure levels. They are combined through the matrix in Table 3; they are not multiplied as numbers.
P1–P7	Privacy-exposure mechanism families: semantic metadata; structural or on-disk leakage; raw-data patterns; unintended inclusion during processing; aggregation, correlation, or inference; persistence beyond the intended lifetime; and misplaced trust in privacy by format. They are categories, not risk ratings.
W1–W10	Wrapper or binding mechanism families from the SSP SIG H5PW model. They cover native compromise, code in data, dynamic loading, lifecycle mismatch, resource amplification, ABI or packaging drift, artifact exposure, trust-boundary confusion, misleading safety signals, and external-reference traversal.
I1–I9 in the H5II model	Independent-implementation mechanism families covering parser/specification divergence through trust-boundary confusion. These are unrelated to the report's four impact levels despite reusing the letter I.
ATK-nnn, HAZ-nnn, EXP-nnn	SSP SIG threat, safety-hazard, and privacy-exposure registry identifiers. They are separate program artifacts, not findings in this report.
EXT, OPS, TCD, SCD in SSP tag fields	SSP classification tags for extensions, operations, toolchains, and distribution, respectively. They classify a mechanism and carry no report-wide risk rating.
HDF5-SHINES-FAMILY-nn	An SSP SIG policy-control identifier. A crosswalk to a report recommendation does not establish adoption or control satisfaction.

Pattern or label	Meaning
APP, ASSURE, CORE, EXT, GOV, INTEROP, PRIV, REL in owner fields	Accountability role labels for application, assurance, core engineering, extension, governance, interoperability, privacy, and release work. They name roles rather than currently assigned people or organizations.

G.3. Quick distinctions

Two groups of terms recur in conclusions and recommendations but answer different questions. These tables are reading aids; the full definitions and the controlling rules remain in the cited report sections.

G.3.1. Risk and response terms

Term	Question it answers	Use in this report
Impact	How serious is the greatest credible consequence?	Ordinal level I1–I4, supported by evidence for the bounded scenario.
Exposure	How readily and commonly can the scenario arise?	Ordinal level E1–E4, based on reachability, preconditions, deployment prevalence, and recurrence; not a numerical probability.
Inherent risk and risk tier	What does the report rate before optional or proposed controls?	The impact/exposure matrix yields Low, Moderate, High, or Critical for a bounded scenario. The two ordinal levels are not multiplied.
Rating status	Is the evidence sufficient to issue that tier?	Rated and Provisional records carry a tier; NE means a required input is not evaluated sufficiently, and N/A marks a non-risk-bearing item.
Current residual risk	What remains after controls already verified in the stated deployment?	It can differ from inherent risk only when evidence shows that existing controls apply and work.

Term	Question it answers	Use in this report
Target residual risk	What might remain if proposed controls are completed?	A projection, not achieved risk reduction; acceptance evidence must pass before it can become current residual risk.
Response urgency	How quickly does the report recommend treatment?	A report-assigned route and target horizon based on remaining risk, time to harm, recurrence or exploitation, and risk tolerance; it is not an adopted stakeholder deadline.
SSP SIG severity and likelihood	How does a separate SSP registry score a mechanism?	Separate 1–5 inputs that the SSP SIG multiplies. Its mechanism scores cannot be converted into or merged with this report's scenario ratings.

G.3.2. Format interpretation terms

Term	What it establishes
Conformance	The bytes or implementation satisfy the normative requirements of the governing specification.
Reader acceptance	One particular reader, version, build, and profile opens or processes a file. It does not alone establish conformance.
Compatibility	A stated producer, artifact, reader, and configuration work together in practice. It is an observed relationship, not a universal property.
Backward / forward compatibility	Respectively, newer software handles older artifacts, or older software handles newer artifacts within a declared feature set. Neither licenses malformed metadata.
Interoperability	Different tools, versions, or implementations exchange and interpret data as intended across a stated scope.
Deployment safety	Processing the artifact under a deployment's authority, limits, and policy does not produce the adverse behavior in question. A file can conform and still be unsafe for a particular deployment.

G.4. Terms and concepts

PDF search is usually the quickest lookup method. The entries are also grouped in four navigable ranges: A–C, D–H, I–P, and Q–Z. A slash in a headword groups terms that are most usefully distinguished together; common secondary terms are repeated or cross-referenced where alphabetical lookup would otherwise be difficult.

G.4.1. A–C

Accountability roles.
The accountable owner answers for an outcome; the implementer performs the work; the verifier independently reproduces acceptance evidence; the approver authorizes rollout; and the risk acceptor decides whether verified remaining risk may be retained. The roadmap separates these roles for High and Critical scope.

Acceptance criteria / acceptance evidence.
The observable condition, test result, document, or operating record required to show that a recommendation was implemented and works over its stated scope. Completing an activity is not acceptance evidence unless its required outcome is also demonstrated.

Access control.
Rules and mechanisms that decide which identities may read, change, create, or delete a resource. HDF5 applications normally rely on the operating system, storage service, or surrounding application for identity and access control.

Adversary / attacker / malicious actor.
A person or system intentionally trying to cause an unwanted result. Safety failures and privacy harms in this report can also occur without an adversary.

Adverse-event scenario.
A bounded statement connecting an initiating actor or non-adversarial event, an assessment object, exposure and preconditions, a condition, and a credible

harm to named assets, people, or operations. This—not a technology or defect name alone—is what the report rates.

Affected scope / affected population.
The specific versions, builds, configurations, deployments, files, users, or workflows to which a scenario applies. A project name or CVE count is not a substitute for establishing this population.

Aggregation, correlation, and inference.
Combining or analyzing data and metadata so that the result reveals information not evident from any one item. This can create privacy harm even when every individual input was disclosed legitimately.

Allocation / resource budget.
Allocation reserves memory, storage, or another finite resource. A resource budget is an enforced upper bound on aggregate or per-object use, such as logical bytes, objects, traversal depth, CPU time, or decompression expansion.

Allowlist.
An explicit list of permitted items, such as approved plugin identities or external directories. Items not on the list are denied. An allowlist is useful only if its storage and update path are themselves protected.

Ambient authority.
Permissions already held by a process—for example filesystem, network, credential, or plugin-loading access—that are available without a fresh, resource-specific authorization decision. An untrusted file can be dangerous when its metadata directs a process to exercise this authority.

Application boundary.
The point where an application decides how an HDF5 capability will be used and with what permissions. A defect at this boundary is not automatically a defect in the HDF5 parser merely because HDF5 carried the instruction.

Archive / archival profile.
An archive preserves information for later use. An archival profile states the formats, features, dependencies, and validation rules accepted for that purpose; preserving primary file bytes alone may be insufficient.

Artifact.
A durable output used as evidence or input. Depending on context this can mean a source or release package, a test fixture, an SBOM, a signed manifest, a log or cache created by a workflow, or a report and decision record.

Assertion / always-active check.
An assertion is a developer check written with facilities such as `assert(...)` and often removed from release builds. An always-active runtime check remains present in supported production configurations. A condition derived from untrusted file bytes must not depend on an assertion alone.

Assessment horizon.
The period over which a scenario is evaluated. Time is material for risks such as a filter becoming unavailable over a decade; ratings with materially different horizons should not be directly ordered without qualification.

Assessment object.
The exact technology, version, file population, component, or deployment being examined. A formal rating attaches to this bounded object and scenario, not to "HDF5" in the abstract.

Assessment surface / scope level.
A surface is a component, input, interface, or boundary considered while looking for risk. A *primary assessment object* is directly assessed; a *boundary case study* examines only a defined interaction with a primary object; and a *contextual reference* illustrates the landscape without creating an audit conclusion about that product. Appearing in a diagram, example, or citation does not mean the entire product was audited.

Assurance.
Justified confidence, based on stated evidence, that a property holds over a defined scope. Tests, review, reproducible builds, and independent verification can increase assurance, but assurance is not an absolute guarantee and does not by itself mean certification or an assurance opinion.

Asset.
Something of value that the assessment seeks to protect, such as scientific

data, metadata, credentials, software integrity, service availability, research validity, privacy, or long-lived records.

Attack mechanism.
A capability or sequence an attacker can use to produce harm. A documented and correctly implemented feature can be an attack mechanism when a surrounding application grants it inappropriate authority.

Attack surface.
The set of inputs, interfaces, features, tools, and extension points through which an adverse condition might be reached. More surface does not by itself establish a vulnerability or a risk tier.

Attestation / certification / assurance opinion.
Formal statements made under defined criteria and evidence requirements. This technical audit is not any of these and does not certify an HDF5 release, application, file, or deployment.

Audit.
A documented technical examination in this report. It does not mean a compliance audit, certification, attestation, penetration test, or exhaustive line-by-line source review.

Authentication / authorization.
Authentication establishes which identity is making a request; authorization decides what that identity may do. Successfully authenticating a connector or plugin does not authorize every file-selected action it can perform.

Authority.
The practical ability to act on a resource, such as reading a local file, opening a network connection, allocating memory, loading code, or using a credential. The report assigns a control failure to the boundary that supplies and governs the relevant authority.

Availability.
The property that data and services remain accessible and usable when needed. Confidentiality, integrity, and availability are the conventional three core security objectives.

Backport.
Applying a correction developed for a newer code line to an older maintained release. A backport is not assumed to contain the fix until the exact release artifact and path are verified.

Backward / forward compatibility.
Backward compatibility lets newer software use older artifacts; forward compatibility lets older software handle artifacts from newer producers within a defined feature set. Neither requires accepting malformed metadata, and historical reader acceptance does not prove format conformance.

Baseline.
The exact version, commit, build, configuration, or deployment state against which an observation is made. This historical ecosystem assessment has no single baseline that represents every conclusion.

Binary / source / release artifact.
Source is human-readable program material; a binary is compiled executable code; a release artifact is a published package containing either or both. Integrity and provenance must identify which exact artifact was assessed.

Binding / wrapper.
Software that exposes HDF5 to another language or programming model, such as Python, Java, or .NET. It translates types, memory ownership, errors, and lifecycle rules, but it does not automatically make every HDF5 capability safe for an application.

Bounded decode.
Reading a defined number of bytes through a cursor or equivalent boundary so a decoder cannot silently run past the available record. The report's preferred sequence is decode, validate, construct, and only then separately authorize activation.

Buffer overflow / out-of-bounds read or write / buffer over-read.
An overflow or out-of-bounds write stores data beyond the memory reserved for a buffer; an out-of-bounds read or over-read reads beyond it. Either can

crash a process, disclose memory, corrupt state, or in some conditions enable code execution.

Build configuration / build profile.
The selected compiler, options, dependencies, features, and debug or release settings used to produce a binary. Different HDF5 builds of the same nominal version may have materially different behavior and dependencies.

Cache / temporary artifact.
A cache retains data to avoid recomputation or retrieval; a temporary artifact is an intermediate file, buffer, snapshot, or log. Either may persist longer than intended and expose data outside the primary HDF5 file.

Callback.
A function or handler supplied for another component to invoke later, often when an event or extension point is reached. A callback executes code in the calling process, so its origin, lifetime, inputs, and authority are part of the security boundary.

Canonical / minimal / nonminimal encoding.
A canonical encoding is the one representation selected when a rule permits several byte representations of the same value. A minimal encoding uses the shortest sufficient representation; a nonminimal encoding uses more space. A nonminimal encoding can still conform unless the specification expressly requires minimal or canonical form.

Capability.
An ability a component exposes or holds, such as opening an external path, contacting a service, allocating memory, or loading executable code. A capability may be intentional and correctly implemented yet still require explicit authorization and limits at a trust boundary; it is not synonymous with a vulnerability.

Census.
A systematic inventory of installed versions, linkage, features, files, or deployments. The roadmap census supplies affected-scope evidence; discovering scope is not itself treatment.

Checked arithmetic.
Addition, multiplication, range calculation, or conversion that detects overflow, underflow, division by zero, truncation, or an invalid bound before using the result for allocation, copying, or traversal.

Checkpoint.
A known recovery point recorded during a long-running write or computation. Checkpoint discipline limits how much work or data must be reconstructed after a failure but is not the same as a file-format transaction guarantee.

Checksum / cryptographic hash / digital signature.
A checksum helps detect accidental change. A cryptographic hash produces a substitution-resistant digest, but detects malicious replacement only when the expected digest comes through a trusted or authenticated channel. A verified digital signature can provide integrity and origin authentication when the verification key is reliably associated with the signer. None of these mechanisms proves that the checked code is safe.

Chunk / chunked storage / chunk index.
A chunk is a separately stored block of a dataset. Chunking supports partial I/O, compression, and extendible datasets. A chunk index is the on-disk structure that maps logical chunk coordinates to stored blocks.

Ciphertext / plaintext.
Plaintext is information in a directly readable form; ciphertext is the output of encryption. Data become plaintext again during authorized processing, where applications, plugins, caches, and logs can expose them.

Closure: defect, class, recommendation, and risk.
Defect closure shows that one affected case was treated. Class closure shows that the recurring family and sibling paths are structurally covered. Recommendation closure requires its acceptance evidence, while risk closure also requires an accountable decision about what remains. These claims are not interchangeable.

Codec.
Software that encodes and decodes a data representation, such as a compres-

sion filter. Long-term readability depends on retaining either a working codec or enough specification and test evidence to reproduce it.

Command-line tool.
A standalone program invoked from a shell. HDF5 inspection and conversion tools are security and privacy surfaces because they parse files, render output, load extensions, and can create persistent copies.

Compatibility.
The practical ability of particular producers, files, readers, and configurations to work together. Compatibility is an observed relationship and is distinct from both conformance to the specification and deployment safety.

Compensating control.
A control used when the preferred or structural control is unavailable, such as isolating an affected parser until an upgrade can be deployed. Its coverage and effectiveness must be tested rather than assumed.

Condition / failure mode / consequence.
The condition is the state that permits an event, the failure mode is how the component departs from intended behavior, and the consequence is the resulting effect. Separating them prevents a visible crash from being mistaken for the earliest root cause.

Confidentiality.
The property that information is not disclosed to unauthorized recipients. Privacy is broader: inference, retention, correlation, or unwanted secondary use can cause privacy harm even without a confidentiality breach.

Conformance.
Compliance with the normative requirements of a specification. A file can be accepted by one reader yet be non-conforming, or be conforming and rejected by a reader that imposes an extra condition.

Confused deputy.
A system with legitimate authority that is induced to use that authority on behalf of a less-privileged requester. In this report, a file-processing service

that follows an attacker-selected local path with the service's permissions is the central example.

Connector / connector stack.
An extension implementing an HDF5 abstraction, especially a VOL connector. Multiple connectors may be layered as a stack, so data origin, caching, network use, and side effects may span more than one component.

Containment / structural prevention.
Containment promptly limits a known exposure, for example by disabling a feature, isolating a process, or monitoring capacity. Structural prevention changes the design, implementation, or process so the recurring defect class is less likely to return. One does not substitute for the other's deadline.

Control.
A technical, operational, or governance measure intended to prevent, detect, limit, or recover from an unwanted event. A proposed control receives credit in residual risk only after relevant acceptance evidence exists.

Controlled rejection.
Stopping on invalid or disallowed input with a bounded, documented diagnostic, without a crash, uncontrolled allocation, unintended side effect, or silent repair. Controlled rejection is often the safe result for unsupported features.

Core VFD.
The specifically named in-memory Virtual File Driver, optionally with a backing file written at close. In that phrase, *Core* does not mean the HDF5 core library.

Corpus / fixture / test vector.
A corpus is a maintained collection of test inputs. A fixture is one input with a stated expected result; a test vector pairs known input and output. Positive fixtures should be accepted, negative fixtures should be rejected, and contested fixtures preserve competing interpretations until adjudication.

Coverage.
Evidence about which declared code, structures, invariants, configurations,

or cases a test or control exercised. A percentage is meaningful only with a versioned denominator and does not, by itself, prove defect or class closure.

Crash consistency.
The guarantees about durable file state when a writer crashes, is killed, or loses power during an update. A flush call, SWMR mode, or successful earlier write does not by itself define those guarantees.

Credential / secret.
Information that grants authority, such as a token, password, private key, or cloud credential. Logs, environment variables, crash dumps, connectors, and plugins can expose secrets even when HDF5 payload data are protected.

Current working directory / search path.
The current working directory is the process's default relative-path base. A search path is the ordered set of locations examined for an external file or loadable module. User-writable or implicit locations can let untrusted parties control what is opened or executed.

G.4.2. D–H

Data integrity.
The property that data remain correct, complete, and unaltered except through authorized changes. In this report it includes avoiding silent wrong data, corruption, and scientifically invalid interpretation.

Data lifecycle.
The stages through which data pass, including creation, processing, storage, sharing, transfer, reuse, archival, and deletion. Security and privacy controls must account for artifacts created at every relevant stage.

Data minimization / tokenization.
Minimization avoids or removes information that is not needed. Tokenization replaces a sensitive value with a less revealing reference. A field is not actually minimized if it disappears from the API view but remains recoverable from raw bytes, free space, logs, or copies.

Data remanence / secure deletion.
Data remanence is information that remains recoverable after ordinary deletion or replacement, including in free blocks, histories, replicas, snapshots, and backups. Secure deletion is a storage-specific process intended to make the defined information unrecoverable across the stated copies and layers.

Dataset.
An HDF5 object containing an array of data values together with a datatype and dataspace, plus optional attributes and storage properties.

Dataspace.
A dataspace defines a dataset's or attribute's extent: scalar, null, or simple, with rank and current or maximum dimensions. A dataspace handle can also carry a selection used for dataset I/O; that selection is not normally part of the stored dataset or attribute definition. A declared logical extent can be much larger than the physically stored payload.

Datatype / committed datatype.
The description of how values are represented and interpreted. A committed datatype is stored as a named HDF5 object that other objects can reference; compound, variable-length, enumeration, and nested types add parsing complexity.

Decode / encode.
Decode turns stored bytes into an in-memory representation; encode performs the reverse. The report requires bounded decode followed by record-, object-, and graph-level validation before construction; invalid state must not reach later encode or flush paths.

Default deny.
A policy under which a capability is unavailable unless it is explicitly permitted. For untrusted HDF5 input, external resources and executable plugins are examples of capabilities suited to a default-deny policy.

Defect / weakness / vulnerability.
A defect is a specific implementation or specification error. A weakness is a broader condition or recurring class that can contribute to harm. A vulnera-

bility is a weakness that can be reached and used under stated conditions to violate a security property. The words are related but are not interchangeable, and none alone assigns a report risk tier.

Defense in depth.
Using independent layers of protection so one failure does not immediately produce harm, such as combining input validation, resource limits, process isolation, and least privilege.

Denial of service / resource exhaustion / amplification.
Denial of service makes a system unavailable. Resource exhaustion consumes finite memory, CPU, storage, output, or handles. Amplification occurs when a small input induces disproportionately large work or allocation.

Dependency / upstream / downstream / transitive dependency.
A dependency is software or infrastructure required by another component. Upstream dependencies are used to build or run HDF5; downstream projects use or package HDF5. A transitive dependency is reached indirectly through another dependency, which can make the actual HDF5 build invisible to an end user.

Deployment.
A concrete installed system: its versions, build options, enabled features, configuration, permissions, users, data sources, storage, and surrounding controls. Risk conclusions often depend more narrowly on a deployment than on a product name.

Deserialization.
Reconstructing application objects or behavior from stored data. HDF5 decoding can be safe while a higher-level model loader unsafely reconstructs executable objects from HDF5 content.

Differential testing / differential conformance testing.
Giving the same inputs to multiple builds or independent implementations and comparing outcomes. A difference exposes a question; it does not by itself prove which implementation or specification is wrong.

Digital signature.
See *Checksum / cryptographic hash / digital signature*. A signature is useful only when verification succeeds and the verification key is reliably associated with the claimed signer.

Disposition / ruling.
An authoritative resolution of a reported issue or specification question, including which interpretation governs and which artifacts or implementations must change. An issue label, implementation behavior, or proposed patch is not a ruling unless the responsible authority records it as such.

Documented observation / reasoned inference.
A documented observation is supported directly by a cited artifact, with source-reported evidence distinguished from independent reproduction. A reasoned inference draws a conclusion from evidence plus stated assumptions and uncertainty. Neither is a recommendation or a formal rating by itself.

Durability.
The property that data acknowledged as written survive the failures covered by the stated guarantee. Durability differs from an in-memory value being visible to a reader and must be evaluated against the storage stack and recovery model.

Dynamic loading / plugin loading.
Loading executable code while a process is running, often from a shared library found on a search path. The loaded code executes with the process's authority, so its identity, origin, path, configuration, and privileges matter.

Effort / implementation dependency / sequence.
Effort estimates the scale of delivery work; a dependency is a concrete prerequisite; sequence records an implementation order and its rationale. These planning fields may shape delivery but never lower or otherwise change inherent risk or response urgency.

Embargo.
A temporary restriction on vulnerability information while affected parties

coordinate analysis, fixes, and disclosure. Embargoed local repair can close a known defect without proving that the recurring class has been eliminated.

Encryption / key management.
Encryption uses a key to make protected information unreadable at a stated boundary. Key management covers generation, custody, access, backup, rotation, recovery, and destruction. Encryption that leaves metadata or keys exposed, or whose keys cannot be recovered, does not satisfy the report's broader privacy and durability needs.

Error path / cleanup / unwinding.
The code followed after an operation fails, including releasing partially created objects and restoring state. Malformed input often exposes lifetime defects in these paths even when the original decode detects an error.

Evidence action / evidence backlog.
A bounded investigation needed because a scenario lacks enough information to rate or route. A backlog record is not a formal risk tier and must never be read as Low risk.

Evidence confidence.
The report's judgment about how directly and completely the evidence supports a scenario and its rating assumptions. High, Moderate, or Low confidence is reported separately from impact, exposure, and risk tier.

Evidence cutoff / subsequent-event update.
The cutoff is the latest date on which substantive evidence is eligible for this revision. A later retrieval can document an older artifact, but a new measurement, fix, or decision after the cutoff is a subsequent event unless the report expressly incorporates it.

Exception / variance.
An approved, scoped departure from a control, gate, route, or deadline. The roadmap requires it to name an accountable authority, rationale, interim control, expiration, and remaining-risk decision; an undocumented deviation is not an exception record.

Exploitability / reachability.
Reachability asks whether a real input can enter the relevant code path in a specified build and deployment. Exploitability asks whether the resulting behavior can be used to cause a security consequence. A source pattern or crash alone may establish neither for supported releases.

Exposure (risk axis).
The E1–E4 axis expressing how readily and commonly a stated scenario arises, considering reachability, preconditions, deployment prevalence, and recurrence. It is ordinal and is not a numerical probability. *Privacy exposure* is a different use of the word: information becoming observable or usable in a way that may cause privacy harm.

Extension.
A component that adds behavior outside the baseline library path, including a filter, VOL connector, or VFD. An extension may be built in or loaded as a plugin; the words *extension* and *dynamically loaded plugin* are therefore not synonyms.

External link / external raw storage / external reference.
An external link names an object in another HDF5 file. External raw storage places a dataset's raw bytes in non-HDF5 files. VDS mappings can also name external source files. This report uses *external reference* or *external resource* as an umbrella for file-controlled access beyond the initial file; it is not limited to the HDF5 object-reference datatype.

Fail closed / fail open.
A fail-closed system denies or safely stops when validation, policy, or a dependency cannot be established. A fail-open system continues, guesses, or silently ignores the condition. Unsupported or unknown untrusted-file features should normally fail closed.

Fault injection.
Deliberately causing failures—such as out-of-space errors, failed writes, or process termination—at controlled points to measure durable state, error reporting, and recovery behavior.

File-derived value / file-controlled metadata.
A size, count, address, path, shape, filter setting, link, flag, or other value obtained from the input file. Until validated and authorized, it must be treated as controlled by the file's originator rather than trusted program state.

File disclosure / data exfiltration.
File disclosure is unauthorized reading or return of file contents. Exfiltration is their unauthorized transfer out of the protected boundary. The demonstrated external-storage incident in this report is rated for the disclosure actually evidenced, not for an unreported code-execution path.

File format / data model / library.
The file format specifies the on-disk representation; the data model describes logical objects and relationships; the library is software that reads, writes, and manipulates them. A defect or capability in one is not automatically a defect in the others.

File locking.
Coordination intended to prevent incompatible concurrent access to a file. Its availability and semantics depend on the HDF5 build, configuration, filesystem, and access pattern; it is not a substitute for crash recovery.

Fill value.
The value HDF5 returns for dataset elements that have not been explicitly written, subject to the dataset's creation and allocation properties. A fill value is part of dataset behavior and should not be confused with bytes that happen to remain in unused storage.

Filter / filter pipeline / filter plugin.
A filter transforms dataset chunks during writing and reverses or interprets the transformation during reading. A pipeline is an ordered sequence of filters. A plugin supplies filter code dynamically; losing that dependency can make data unreadable.

Finding / formal finding.
A documented adverse-event scenario supported strongly enough to enter

the authoritative findings register. Its identifier, scope, evidence, rating, and response route are distinct from the recommendations that may address it.

Flush.
An operation that pushes buffered HDF5 state toward a lower storage layer. What becomes durable after a flush depends on the documented library, driver, operating-system, and storage guarantees; the word alone does not promise crash consistency.

Format bounds / library-version bounds.
HDF5 properties that constrain which on-disk format versions and features a writer may emit. They are not the same as the installed library's version and do not alone prove reader compatibility.

Fuzzing / mutation.
Automated testing that generates or alters many inputs to find crashes, resource problems, or invariant violations. Whole-file fuzzing mutates bytes broadly; structure-aware or invariant-guided fuzzing targets meaningful HDF5 records and boundary values.

Graph / traversal.
HDF5 files contain linked structures that can be viewed as a graph. Traversal follows those relationships. Cycles, excessive depth, repeated work, or external targets require explicit progress rules and budgets.

Group.
An HDF5 container object whose links organize datasets, committed datatypes, and other groups into a hierarchy, much like directories organize filesystem names.

Guard / dominating check.
A guard is a check that prevents unsafe execution when its condition fails. A check dominates an operation when every path to that operation must pass through the check. Cataloging a check matters because a nearby assertion may not protect all release-build paths to the unsafe use.

Hard link / soft link.
An HDF5 hard link is a group entry that names an object in the same file;

more than one hard link can name the same object. A soft link stores a path to be resolved later and can be dangling. An external link additionally names another file and crosses an external-resource boundary.

Hazard.
A condition with the potential to cause a non-adversarial safety loss. The SSP SIG hazard registry and this report use different rating methods, so a hazard identifier or its registry score is not a report finding or tier.

HDF Group, The.
The nonprofit organization that develops and supports the reference HDF5 software and stewards the format. It is distinct from the broader ecosystem of downstream and independent projects.

HDF5 interface and subsystem prefixes.
Prefixes such as `H5C`, `H5O`, `H5T`, and `H5Z` identify metadata-cache, object, datatype, and filter interfaces or subsystems. Some occur in public API names as well as internal symbols; the surrounding finding names the relevant path.

HDF5 object / attribute / link / reference.
Groups, datasets, and committed datatypes are principal HDF5 objects. An attribute is a named metadata item with its own datatype and dataspace, attached to an object. A hard link gives an object a name in a group; soft and external links instead store targets resolved during traversal and may dangle. An object reference identifies an object, while a region reference also identifies a selection within a dataset.

Heap.
In ordinary programming, a heap is the memory area used for dynamic allocation. In the HDF5 format, local, global, and fractal heaps are also named on-disk metadata structures. The surrounding phrase determines which meaning applies.

Hostile / malicious / untrusted input.
Input for which the processor cannot rely on the originator's intentions or care. *Untrusted* includes accidentally malformed third-party files; *malicious* means

intentionally crafted. Safe parsing should not depend on knowing which one arrived.

G.4.3. I–P

Immutable file identity.
See *Open handle / immutable file identity*. It is the property used to show that validation and later processing apply to the same bytes, rather than merely to the same path name.

Impact.
The I1–I4 axis describing the greatest credible consequence supported by the evidence, from limited through severe or systemic. It is not an unconstrained worst case and is not multiplied numerically by exposure.

Independent copy.
A separately stored copy whose failure modes and administrative control are not the same as the working file. For unregenerable data, a duplicate on the same full device or in the same failure domain is not the independent recovery control intended by the roadmap.

Independent implementation.
A separately developed reader or writer that interprets the HDF5 format without simply calling the reference C library for the relevant work. It can reveal specification or reference-library errors, but it has its own parser, validator, writer, and dependency risks.

Inherent risk.
Risk for the stated scenario and deployment before optional, proposed, or organization-specific controls receive credit. Only mandatory controls intrinsic to the identified implementation, version, and default configuration are included.

Input validation.
Checking that input has an allowed type, range, size, relationship, and structure

before it influences trusted operations. Local field checks are not enough when an invariant spans an object or reference graph.

Integer overflow / divide by zero.
Integer overflow occurs when an arithmetic result cannot be represented in the destination integer type; division by zero has no valid arithmetic result. File-derived dimensions, counts, and sizes must be checked before either result can influence allocation, address calculation, or copying.

Integrity.
For data, see *Data integrity*. In a software supply chain, integrity is evidence that an artifact has not changed since an identified producer published or signed it. Either meaning must state its object and boundary; integrity is not the same as provenance or safety.

Interoperability.
The ability of different tools, versions, and implementations to exchange and interpret data as intended. It depends on conformance, supported features, extensions, configuration, and compatibility policy.

Interpretability.
The ability to recover the intended meaning of preserved data, not merely its bytes. It can depend on schema, provenance, format bounds, external resources, filters, connectors, drivers, and a reader that implements the relevant rules.

Invariant.
A condition that must always hold for a structure or operation to be valid, such as a count fitting its buffer or two metadata fields agreeing. An invariant registry ties each condition to enforcement, tests, fuzz targets, findings, and migration status.

Isolation / process isolation.
Separating file processing from sensitive resources, other processes, or the host so a failure has limited reach. Containers, restricted workers, operating-system sandboxes, and separate accounts are possible mechanisms; their actual boundaries must be verified.

Journal / write-ahead log.
A journal records update information used to recover consistent state. A write-ahead log records the intended change before applying it to the main data. These are candidate crash-consistency mechanisms, not guarantees HDF5 is assumed to provide today.

Least privilege.
Giving a process only the filesystem, network, credential, code-loading, and other authority required for its task, for only as long as needed. It limits the consequence when an input or component misbehaves.

Legacy.
An older format, feature, file, tool, or behavior retained for compatibility. Legacy status does not mean unsafe, but historical tolerance of invalid input must not silently become a permanent compatibility requirement.

Library / reference library / `libhdf5`.
A library is reusable program code. `libhdf5` is the principal HDF5 C library; *reference library* emphasizes its central role in implementing the format. Reference status does not make its behavior automatically normative when it conflicts with the specification.

Likelihood / severity in SSP SIG registries.
Likelihood and severity are separate 1–5 inputs in the SSP SIG mechanism registries, where they are multiplied and banded. They are not this report's exposure and impact levels, and their product cannot be converted into this report's scenario-based risk tiers.

Linkage: static, dynamic, system, bundled, or vendored.
Static linkage copies library code into a binary at build time; dynamic linkage loads a shared library at run time. A system library comes from the operating environment; bundled or vendored code is shipped within another project. The linkage determines which HDF5 code a user actually runs and how it is updated.

Logical size / physical size.
Logical size is the amount of data described after interpreting dimensions and

types; physical size is the storage currently occupied. Compression, sparse layout, fill values, external storage, and VDS mappings can make them differ by orders of magnitude.

Lossless / lossy compression.
Lossless compression reconstructs the original values exactly. Lossy compression intentionally discards information according to precision, rate, or error settings. Those settings and the resulting sizes can reveal data characteristics even when values are not directly shown.

Machine-readable specification.
A structured description that software can process to generate validators or fixtures. It makes choices testable but becomes normative only through an explicit governance decision and cannot resolve ambiguity by itself.

Malformed / invalid / inconsistent file.
A file whose bytes violate required structural, arithmetic, or relational conditions. *Malformed* does not imply malicious intent. A valid but policy-disallowed file is different: it may conform to the format yet still be unsafe for a particular deployment.

Managed runtime / native code.
A managed runtime, such as a JVM or .NET runtime, supplies services such as automatic memory management. Native code executes directly under platform conventions, commonly through C or C++. A binding can cross from managed code into native HDF5 code and therefore inherit both models' failure modes.

Manifest.
A machine-readable list of expected artifacts, identities, versions, capabilities, or dependencies. A protected plugin manifest can support an allowlist; a corpus manifest can state each fixture's provenance and expected outcome.

Materialization / activation.
Materialization constructs an in-memory object from validated data. Activation causes effects such as publishing it to a cache, registering an identifier, loading a plugin, decompressing data, invoking a callback, or opening an external file. Successful parsing must not automatically authorize activation.

Memory corruption / memory safety.
Memory corruption changes or accesses memory outside the program's intended rules. Memory safety prevents classes such as out-of-bounds access, use-after-free, double-free, and invalid lifetime use. A memory-safe language reduces many such defects but does not prevent resource exhaustion, logic errors, unsafe foreign code, or policy failures.

Memory leak.
Allocated memory that is no longer usable but has not been released. Repeated leaks can exhaust a process even when no individual access crosses a memory boundary.

Metadata / semantic metadata / structural metadata.
Metadata describe data. Semantic metadata convey meaning, such as names, attributes, units, provenance, or enumeration labels. Structural metadata describe organization and storage, such as hierarchy, shapes, layouts, chunk indexes, filters, addresses, and references. Either can be sensitive.

Metadata cache / cache image.
The metadata cache retains decoded HDF5 metadata in memory. A cache image is a serialized representation used to accelerate a later open. Both manage interrelated structures whose corrupted sizes, ranks, or lifetimes can affect parser and recovery paths.

Migration / repair / regeneration.
Migration transforms data to a supported representation. Repair changes an existing artifact to restore a defined valid state. Regeneration recreates it from an authoritative source. Each must be deterministic, preserve provenance, and state how checksums and meaning change.

Mitigation / treatment.
An action intended to reduce likelihood, exposure, or impact, or to avoid, transfer, contain, or recover from a scenario. A proposed mitigation is not treated as effective until acceptance evidence supports it.

Model artifact / model loader.
A stored machine-learning model, weights file, or related package and the

software that reconstructs it. An HDF5 container can hold application-level instructions or objects whose security semantics are defined by the loader, not by HDF5 alone.

Named VFD families.
SEC2 and STDIO use ordinary local-file mechanisms; Core uses memory; Family, Split, Multi, and Subfiling distribute one logical file across physical files; MPI-I/O supports parallel access; ROS3 reads object storage; and Mirror, Log, and Onion add replication, logging, or history. These names identify storage behavior, not general meanings of words such as "core" or "family."

Normative / informative / implementation-defined.
Normative text states requirements for conformance. Informative text explains without imposing a requirement. Implementation-defined behavior is deliberately left to an implementation but should be documented with its compatibility consequences.

Null dereference.
Attempting to access memory through a null pointer. The usual result is a crash, but the precise consequence depends on the code and platform.

Object header / superblock / B-tree / free-space metadata.
Internal on-disk structures used to describe an HDF5 file. The superblock identifies global file properties; object headers hold messages describing objects; B-trees index records; free-space metadata tracks reusable regions. These structures are parser inputs even though most users never manipulate them directly.

Object store.
A storage service that addresses data as named objects rather than ordinary filesystem byte streams. S3-compatible stores are examples; their credentials, network requests, copies, logs, and consistency behavior become part of the deployment boundary.

On-disk.
Stored in the serialized file representation rather than existing only in program

memory. On-disk metadata remain input-controlled when a file comes from an untrusted source.

Open handle / immutable file identity.
An open handle is a process reference to an already opened file or resource. Binding validation and later use to the same handle, or to a cryptographically or otherwise reliably verified immutable identity, prevents a path from being swapped to different bytes between the two steps.

Open or processing profile.
The report proposes named bundles of policy choices and resource limits for opening or processing a file, with design targets such as legacy, *trusted-fast*, *untrusted-strict*, or forensic. These names are not current HDF5 guarantees; a profile becomes useful only when the relevant APIs and runtime paths enforce it.

Oracle.
A source of expected answers used in testing, such as a validator that labels a fixture valid or invalid. An oracle can expose a disagreement but must itself be reviewed; its answer is not automatically the governing specification.

Parser.
Code that reads bytes and constructs meaning from their structure. Because HDF5 is a complex binary format, its library and tools contain many interacting parsers for metadata and data descriptions.

Path traversal / path confinement.
Path traversal uses components such as `..` or absolute paths to escape an intended directory. Confinement ensures resolved external resources remain under an explicitly permitted root after normalization and resolution.

Payload.
The primary data values a user intends to store, as distinguished from metadata, structure, indexes, logs, caches, and other context. Payload-only protection can therefore leave substantial information exposed.

Penetration test / exploit development / source review.
A penetration test actively attempts to compromise a system. Exploit de-

velopment turns a weakness into a reliable attack technique. Source review examines code for defects or assurance gaps. This report is a targeted technical examination and does not claim exhaustive performance of any of these activities.

Plugin.
Code loaded to extend a program without rebuilding it. HDF5 filter, VOL, and some VFD extensions can be plugins. A plugin is executable code, not inert file data, and normally runs with the host process's authority.

Poisoned or suspect file state.
A proposed durable indication that a write failure may have left a file unsafe to trust. Clearing such a state would require a defined diagnostic and recovery procedure rather than an ordinary reopen.

Precondition.
A fact that must hold for a scenario to occur, such as a feature being enabled, the process having access to a target, or an affected version being deployed. Preconditions shape exposure; they are not implementation-effort factors.

Preflight validator / enforcement coupling.
A preflight validator inspects a file before expensive allocation, deserialization, plugin activation, or external access. Its result is advisory unless it is bound to the same immutable file identity or open handle and the same policy is rechecked at side-effect boundaries.

Preservation object.
Everything that must be retained to make archived information accessible and interpretable. It may be a self-contained HDF5 file, or it may also require external stores, codecs, connector behavior, configuration, schemas, test vectors, credentials policy, and a reproducible reader environment.

Primary artifact / primary evidence.
The source closest to the fact being established, such as the specification, affected source and fix, test result, vendor advisory, or first-party incident record. A primary source can still be incomplete or wrong and must be bounded to what it actually shows.

Primary assessment object / boundary case study / contextual reference.
The primary object is directly assessed. A boundary case study assesses only a defined interaction with that object. A contextual reference explains the surrounding ecosystem without making an audit conclusion about the referenced product. See also *Assessment surface / scope level*.

Privacy.
Protection against adverse consequences from disclosure, retention, inference, aggregation, correlation, surveillance, or secondary use of information. Privacy harm can occur during authorized, technically correct operation and is not limited to a security breach or legal compliance question.

Privilege escalation.
Gaining authority beyond what the initiating user or process was intended to have. Loading code through a path writable by a lower-privileged user can let that code execute with a higher-privileged process's permissions.

Profile.
A named bundle of assumptions or settings. The report uses the word for security/open policy, build configuration, producer/consumer compatibility, deployment scope, and archival requirements. These profiles answer different questions and are not interchangeable.

Property list.
An HDF5 API object that carries configuration for operations such as opening a file, creating a dataset, selecting a driver, or loading external resources. Several proposed security controls are expressed as properties so policy can be bound to the relevant operation.

Provenance.
Evidence of where data or software came from, who or what produced it, which versions and transformations were involved, and how custody changed. A valid signature can support provenance but does not describe every transformation or prove correctness.

G.4.4. Q–Z

Rank / dimension / shape / extent.
Rank is the number of axes in an HDF5 dataspace. Dimensions give the size along those axes; their ordered values form the shape or current extent. These file-declared values must be checked before a consumer allocates an array.

Rating status: Rated, Provisional, NE, and N/A.
Rated means the evidence supports the impact, exposure, and tier. Provisional means those can be estimated but material uncertainty remains. NE means a required rating input is insufficient, and N/A means the item is not risk-bearing. Only Rated and Provisional records carry report-wide tiers.

Reader acceptance.
The observed fact that one reader opens or processes a file. Acceptance does not by itself prove specification conformance, internal consistency, safe deployment, or acceptance by another version or implementation.

Reader / writer / producer / consumer.
A writer or producer creates an HDF5 artifact; a reader or consumer interprets it. The terms can refer to libraries, tools, applications, or independently implemented software, and each exact version and profile matters.

Recommendation.
A proposed control outcome identified separately from a finding. Its number does not encode risk, urgency, effort, dependency, or implementation sequence.

Recovery / recoverability.
The ability and documented procedure to return data or service to a known, usable state after failure. Reopening a file is not sufficient evidence if it silently returns incorrect data.

Recurrence / prevalence.
Recurrence is repeated observation of a mechanism or event. Prevalence is how widely it exists in the relevant installed population. Several CVEs can show a recurring class without establishing current deployment prevalence.

Reference implementation.
The implementation maintained as the principal realization of a specification. It is important evidence and often defines practical compatibility, but its behavior does not override a clear normative requirement without an authoritative specification decision.

Regression / regression test.
A regression is the return of a previously corrected defect or an unintended loss of supported behavior. A regression test is a permanent, preferably small test intended to detect that return. Stricter rejection of invalid input is not automatically a regression.

Re-identification / secondary use.
Re-identification connects data thought to be anonymous or de-identified to a person or entity. Secondary use applies information for a purpose beyond the one for which it was collected or shared. Both can create privacy harm without breaking file encryption or access controls.

Release build / debug build.
A release build is configured for supported distribution and commonly disables assertions or adds optimization. A debug or assertion-enabled build favors diagnostics. Security-relevant behavior must be tested in the configurations users actually receive, including paired assertion-enabled and `-DNDEBUG` builds. A *supported release* is instead a shipped version within the project's support policy; the two phrases are not interchangeable.

Reproducible build / reproducible environment.
A reproducible build can be recreated from recorded source, tools, dependencies, and instructions with a defined matching result. A reproducible environment preserves enough of the runtime stack to execute or reimplement a dependency later. Either claim must state what is reproduced and how equality is tested.

Residual risk: current and target.
Current residual risk is what remains after verified existing controls. Target residual risk is a projection for proposed controls and is not achieved until acceptance criteria pass. Neither replaces the inherent-risk rating.

Resource budget.
See *Allocation / resource budget*. A useful budget is an enforced bound over the relevant aggregate work or resource use, not merely a per-field size check.

Response urgency / response clock.
The report-assigned treatment lane—Immediate, Near-term, Planned, or Monitor—and its target horizon. Urgency considers current residual risk, time to harm, exploitation or recurrence, and risk tolerance; dependencies and effort may shape delivery but do not lower it. These are assessment recommendations, not stakeholder-approved commitments until adopted and assigned.

Risk / risk rubric / risk tier.
Risk is the possibility and consequence of a stated adverse scenario. The rubric is this report's decision method; it combines ordinal impact and exposure through a matrix to yield Low, Moderate, High, or Critical. A tier belongs to a bounded scenario, not to a CVE count, recommendation, technology, or project. A range such as Moderate–High remains a range even when response is conservatively routed at its upper bound.

Risk acceptance / risk owner / risk tolerance.
Risk acceptance is an accountable decision to retain verified remaining risk. The risk owner has authority to make or escalate that decision. Risk tolerance states how much risk an organization permits; silence, an expired exception, or an unfinished mitigation is not acceptance.

Root cause / symptom / consequence.
The root cause is the earliest evidenced failure that explains an event, such as accepting an invalid file-derived size. A crash or buffer overflow may be a symptom or consequence. Fixing one symptom does not prove elimination of the root-cause class or sibling paths.

Runtime check.
A condition evaluated while the deployed program runs. A security-relevant runtime check must be present in supported release builds and return a controlled error when it fails.

Safety.
Operational robustness and protection of integrity, availability, recoverability, interoperability, and long-term accessibility without assuming a malicious actor. It does not mean regulated physical or functional safety in this report.

Sandbox.
An enforced environment that restricts what a process can read, write, execute, or contact. A temporary directory alone is not a sandbox; the actual filesystem, network, credential, and process boundaries must deny access.

Sanitizer.
Instrumentation that detects selected runtime errors during tests. ASan, UBSan, and LSan cover different classes. A sanitizer-clean corpus is useful evidence but cannot prove absence of all defects or policy failures.

Scope tag.
A label identifying where a CVE or condition belongs, such as core parser, official tool, plugin loading, external resource, or downstream deserialization. It supports triage but assigns neither a rating nor priority.

Search path.
See *Current working directory / search path*. The order and ownership of locations searched for external files or loadable code can determine which resource the process actually uses.

Security.
Protection against an adversary compromising confidentiality, integrity, or availability, or causing a process to exercise authority contrary to the operator's intent. A security consequence can arise from a parser defect or from unsafe use of an intentional capability.

Security advisory.
A published notice about a security issue, often stating affected versions, impact, and treatment. Advisory identifiers such as a CVE or GHSA help locate records; the underlying evidence and applicability still require review.

Security profile.
An enforceable bundle of feature policy and limits appropriate to a trust con-

text. The proposed untrusted-HDF5 profile denies unnecessary external and executable capabilities and bounds parsing, traversal, output, and allocation.

Segmentation fault / crash.
A segmentation fault is process termination commonly caused by an invalid memory access. A crash is any unexpected process termination. Either is an observable failure, not by itself proof of root cause or reliable exploitability.

Selection / hyperslab / point selection.
A selection identifies which elements of a dataspace participate in an I/O operation. A hyperslab describes regular blocks with offsets, strides, counts, and block sizes; a point selection lists individual coordinates. Selections can drive large work or allocations and therefore require bounds checking.

Self-describing data.
Data that carry structural information needed to decode them, such as datatypes, dataspaces, names, attributes, and relationships. HDF5 can also carry semantic metadata, but *self-describing* does not guarantee that all context needed to interpret the scientific meaning is present. The included metadata can itself disclose sensitive context.

Sensitive information.
Information whose disclosure, inference, retention, or use can harm people, organizations, research, operations, or obligations. Sensitivity is contextual and can reside in metadata, structure, access patterns, and derived results as well as payload values.

Serialization.
Turning an in-memory value or object into bytes for storage or transfer; the reverse operation is deserialization. See also *Decode / encode* and *Deserialization*.

Side effect.
An externally observable action beyond constructing a value in memory, such as opening another file, loading code, making a network request, writing a cache, allocating a large buffer, or invoking a callback. Validation does not by itself authorize a side effect.

Software license.
The legal terms governing use, modification, and distribution of software. The report's GPLv3-or-later versus permissive BSD-style discussion is a reuse and provenance constraint, not a security rating; legal review may be required before incorporating code or generated artifacts.

Specification / normative ambiguity / adjudication.
A specification defines the format's requirements. A normative ambiguity exists when those requirements permit materially competing interpretations. Adjudication is the authoritative decision that identifies the correct rule, the artifact to change, or an explicitly implementation-defined boundary.

Stack.
In ordinary code execution, the stack holds call frames and many local values; deep recursion can exhaust it, and a stack buffer can be overrun. In extension chapters, a *connector stack* instead means multiple layered VOL connectors.

Static analysis / dynamic testing.
Static analysis examines source or binaries without executing the target program; dynamic testing runs it with selected inputs. They find overlapping but different problems, and evidence from one should not be described as if it came from the other.

Storage exhaustion.
Running out of space or quota while writing, flushing, or closing a file. An operation can report failure yet still leave uncertain durable state, which is why monitoring, headroom, error signaling, and recovery are separate controls.

Storage layout.
The principal dataset layouts are compact (raw data in the object header), contiguous (one continuous region in the HDF5 address space), chunked (separately indexed blocks), and virtual (mappings to source datasets). External raw storage is a form of contiguous layout that places bytes in one or more external files. Layout affects performance, compatibility, and security policy.

Supply chain.
The people, source, dependencies, build systems, automation, packages, dis-

tributors, and downstream consumers through which software is produced and delivered. Supply-chain integrity does not by itself establish parser safety or deployment compatibility.

Supported release / affected version.
A supported release is a shipped version still covered by the maintainer's support policy. An affected version is one for which evidence establishes the relevant condition. Development-tree behavior, an unspecified older version, and a supported affected release are different claims.

Threat model.
A structured description of assets, actors, trust boundaries, capabilities, assumptions, attack mechanisms, and consequences. It helps enumerate scenarios but does not automatically assign this report's risk ratings.

Tranche.
A bounded, independently deliverable subset of a larger migration or assurance program, such as one family of on-disk structures. Tranches support staged implementation; their order does not create a risk rating.

Triage.
The initial classification and routing of a report or record: determining scope, affected versions, duplicates, reachability, owner, and next evidence or treatment action. Triage is not the same as remediation or risk acceptance.

Trust boundary / trusted computing base.
A trust boundary is where data or control passes between parties or components with different authority or assumptions. The trusted computing base is the set of hardware, software, configuration, and people whose correct behavior the security property depends on. In-process plugins normally join it.

Trusted.
A context-dependent claim that may refer to an input source, verified provenance, code admitted to the trusted computing base, or the proposed *trusted-fast* processing profile. It does not automatically mean conforming, harmless, confidential, or free of defects.

Type confusion.
Using a value or memory object as though it had an incompatible type. This can produce incorrect interpretation, invalid memory access, or corruption even when the underlying bytes are within an allocated region.

Undefined behavior.
A C/C++ operation for which the language standard imposes no required result. It can appear to work in one build and fail, corrupt memory, or be optimized in surprising ways in another.

Uninitialized value.
Memory or a variable used before the program has assigned a valid value. Its contents are indeterminate and can affect addresses, sizes, control flow, or output.

Use-after-free / double-free.
Use-after-free accesses an object after its storage has been released. Double-free releases the same storage twice. Both are memory-lifetime errors that can cause crashes, corruption, or, in some conditions, code execution.

User block.
An optional area at the beginning of an HDF5 file reserved for application content. It can contain arbitrary bytes, including scripts, XML, identifiers, or other sensitive information not visible in an ordinary dataset listing.

Valid.
An overloaded word that must be qualified. Bytes can be syntactically decodable, internally consistent, specification-conforming, accepted by a particular reader, and authorized under an application's policy—or only some of these. None of the first four alone proves safe deployment.

Validation scope: record, object, and graph.
Record-local validation checks one decoded structure. Object-wide validation checks relationships across all records forming an object. Graph validation checks references, cycles, aggregate limits, and relationships across objects. A file can pass the first scope and fail a wider one.

Vendored library.
A copy of a dependency included and maintained inside another project's source or package. Users may not realize they are running it, and system-library updates may not replace it.

Version-and-path gap.
The report's term for missing evidence about which versions contain a defect, which real entry paths reach it, and which versions or commits fix it. Until that gap is closed, a CVE history cannot be treated as an installed-population applicability list.

Vulnerability.
See *Defect / weakness / vulnerability*. A bug, assertion site, feature, or CVE mention is not enough by itself to establish a vulnerability in every version or deployment.

Whole-file / payload-only protection.
Payload-only protection covers primary data values but may leave names, attributes, shapes, filter settings, links, and internal structure readable. Whole-file or equivalent all-metadata protection covers the serialized file at the stated boundary, but still does not prevent post-decryption inference, authorized misuse, logs, caches, key loss, or uncontrolled copies.

Work package / roadmap phase.
A planned bundle of implementation and evidence work with an owner, dependency, effort, target, and acceptance condition. Its phase is a delivery construct and does not assign or alter a finding's risk tier.

Workflow artifact.
An output created around the primary data, such as a log, redirected command output, cache, temporary file, converted copy, snapshot, crash dump, or derived dataset. These artifacts can change the security, privacy, and preservation boundary.