# Securing the HDF5 Supply Chain via Safe-OSE

**M. Scot Breitenfeld**

The HDF Group

Improving Scientific Software Conference 2026

This material is based upon work supported by the National Science Foundation under Federal Award No. 2534078.

# HDF5: Critical Data Infrastructure

## What is HDF5?

A technology suite for storing and managing large, complex data — from simple tables to multidimensional arrays.

## Where is it used?

HPC, AI/ML, cloud computing, long-term archival, and the exchange of data across platforms.

**700+**
GitHub dependents

**128**
CVEs reported

**500+**
Google Scholar results

**25+**
years in production

## Sectors

| | | | | | |
|---|---|---|---|---|---|
| **National Security** | **Open Science** | **Healthcare** | **Finance** | **AI / ML** | **IoT** |

# Safety, Security, and Privacy (SSP)


**Three interconnected domains**

**Safety** — Unintentional defects: crashes, corruption, race conditions

**Security** — Intentional exploitation: malicious plugins, supply-chain attacks

**Privacy** — Data exposure: metadata leakage, anonymization gaps

*Overlaps define compound threats: Vulnerabilities, Breaches, Exposures, and Catastrophes*

# Vulnerability Landscape: Seven Categories

**FMT File Format**
Malformed file attacks, ambiguous specs, no integrity verification

**LIB Library**
Memory safety, insecure defaults, race conditions, undefined behavior

**EXT Extensions**
Unsafe plugins (VOL, VFD, filters), covert channels -> hijacking

**TCD Toolchain / Deps**
Vulnerable packages, scripting flaws, unpinned builds

**OPS Operational**
Insecure sharing, logging leaks, missing access controls

**PRV Privacy**
Metadata leakage, insufficient anonymization, traceability

**SCD Supply Chain**
Unsigned binaries, trojanized deps, compromised repos

*Vulnerabilities interact across layers — e.g., a malformed file (FMT) triggers a buffer overflow (LIB) in a plugin (EXT) loaded from an unverified build (SCD).*

# Technical & Socio-Technical Attack Vectors

## Technical Vectors

- Malformed File Attacks (CVE-2021-37501)
- Privilege Escalation via Plugins (CVE-2022-26061)
- Side-Channel Exploits (timing, memory layout)
- Race Conditions & Memory Corruption (the bread and butter of C/C++ security bugs)

## Socio-Technical Vectors

- Social Engineering (trojanized tools)
- AI-Generated Code Risks (PR #5210)
- Insider Threats & Misconfigurations
- Compliance Failures (no audit trails)
  - No access controls
  - Data exposed through ftp or Cloud buckets

# NSF-Safe-OSE: Initiative Overview

## AUDIT PHASE

Months 1–9

- Launch: scope, methodology, advisory group
- Assessment: systematic audit of all 7 categories
- Evaluation: risk register, mitigation roadmap
- Community workshop & bug bounty

## MITIGATION PHASE

Months 10–24

- Track A: Library & file format hardening
- Track B: Ecosystem best practices
- Continuous SSP monitoring & CI/CD
- Hardened releases & security playbook

*Lasting security requires socio-technical infrastructure — clear guidelines, transparent processes, and pathways for diverse contributors.*

# Segmented Risk Approach

**Sector-Specific**
Gov/Defense, Academia, Commercial, Healthcare, IoT — each with different regulatory, compliance, and threat profiles

**Use Pattern**
Data producers, consumers, extension developers, maintainers, integrators — each with distinct attack surfaces

**Infrastructure Scale**
Desktop to HPC clusters to embedded/air-gapped — 'blast radius' varies with scale and software diversity

**Data Sensitivity**
Short-lived test data vs. regulated archival vs. operational real-time — guides proactive vs. reactive mitigation

*This segmentation maps user types to threat profiles, prioritizes high-risk mitigations, and tailors community outreach.*

# Audit Phase: From Threat Modeling to Action

## Launch

- SSP Audit Charter
- Community Advisory Group
- Ecosystem Inventory
- Custom Audit Framework

  (OWASP, MITRE, NIST)

## Assessment

- Category-specific risk reports (7)
- STRIDE threat modeling
- Preliminary risk register
- Bug bounty program
- Community workshop
- Community survey

## Evaluation

- Final audit report
- Mitigation roadmap
- Methodology review
- SSP audit archive
- Governance recommendations

Adapted from OWASP SAMM, MITRE ATT&CK, NIST 800-218, and OpenSSF Scorecard

# SSP Survey: What the Community Told Us (n=33)

**45%**
rate supply-chain threats Major/Critical

**39%**
rate malformed file attacks Major/Critical

**52%**
load third-party plugins regularly

**30%**
saw corruption after crash/power loss

33 respondents • 19 developers • 12 private • 9 gov • 12 N. America • 11 Europe

Full results: github.com/HDFGroup/hdf-clinic | Survey still open: forms.gle/nD6hmRUd9h5H2KM19

### What This Tells Us

- Prioritize supply-chain visibility (SBOM, CVE communication) and defensive parsing for untrusted files
- Build a "safe open" posture: read-only + place resource limits as defaults for untrusted inputs
- Design for offline/air-gapped verification — 64% require it
- Strong appetite for built-in integrity checks: 55% would adopt immediately
- 79% accept up to 10% overhead for security features

# Mitigation Track A: Library & Format Hardening

**Unsafe Code Refactoring**
Input validation, buffer overflow fixes, format-level checksums

**Policy-Driven Plugin Manager**
Signature verification, version constraints, execution isolation

**Fuzz & Test Expansion**
Edge-case testing, fuzz file format parsing, AddressSanitizer integration

**Memory-Safe Language Strategy**
Identify critical modules for Rust/C# migration or wrapping

**SAST / DAST in CI/CD**
Static & dynamic security testing, GitHub security advisories

**Machine-Readable Format Spec**
Kaitai/poke-based spec, enabling automated verification tools

*Goal: Hardened HDF5 releases (e.g., 2.x.x-hardened) with embedded safer-by-default behaviors*

# Ruminations 1

## A New Breed of Tools

- A machine-readable spec. brings CI/CD to the format
- Easier to create audit tools, e.g., discover "hidden objects"
- Create repair tools: Think of GNU poke as an editor for binaries

## Independent Implementations

- Share only the file format spec.
- Ensure everyone is on the same page!
- Examples: C#, Java, Python, Node.js, Rust, etc.
- Current spec. is a Word/HTML doc. (Yikes!)
- Automate compliance checks by having a machine-readable specification
- GNU poke - HDF5 pickle
  - Expect the first version in late Q2/Q3

# Mitigation Track B: Ecosystem Hardening

**Data Integrity & Privacy**

Encryption strategies, mitigation documentation, best-practice guides

**Secure-by-Default Templates**

Reference implementations for VOLs, VFDs, and filters with security patterns baked in

**Maintainer Collaboration**

Patch integrations with h5py, PyTables, rhdf5, HDF-Java; AI contribution review policies

**Package & Distribution Security**

Signed packages, reproducible builds, CVE tracking across Conda, PyPI, Linux distros

**HDF5 Security Playbook**

Comprehensive guide for integrators with webinars and training

**SBOM & Vulnerability Feed**

Ecosystem-wide Software Bill of Materials and shared vulnerability tracking

*Goal: A proactive security culture across the entire HDF5 distribution ecosystem*

# Toward Signed Plugins & Policy-Driven Loading

### Today: Simplistic Plugin Manager

- Binary enable/disable per plugin type
- Search path manipulation via env vars
- No cryptographic signature checking
- No version or compatibility constraints
- No execution isolation or sandboxing

### Future: Policy-Driven Framework

- YAML/JSON policy files at runtime
- Version and inter-plugin constraints
- Operation-type restrictions (allow/deny)
- System / user / process-level defaults
- Cryptographic signature verification

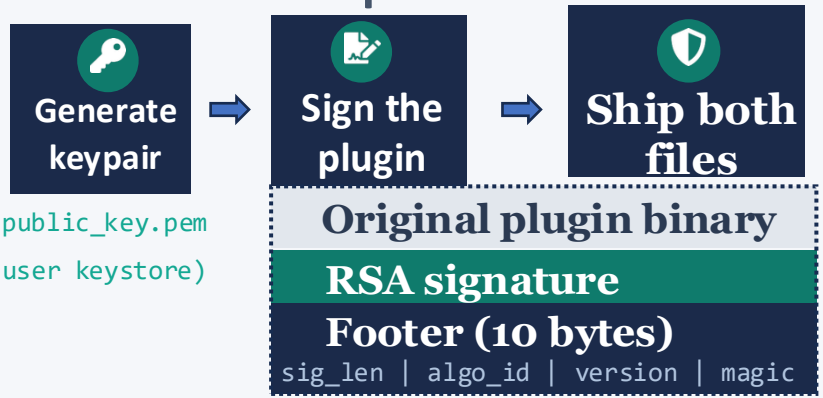
*Defaults will not break existing apps — they give users choices.*

# Cryptographic verification for the HDF5 plugin ecosystem

## Why sign plugins

- Tamper detection
  - Detects if a plugin binary has been modified after it was signed by the developer.
- Authenticity
  - Verifies the plugin actually comes from a trusted source before loading arbitrary code.
- Revocation
  - Block compromised plugins by adding their signature hash to a revocation list.

## For filter developers




## For plugin users

**Set env variable**

```
export HDF5_PLUGIN_KEYSTORE=~/.hdf5/keystore
```

# Vulnerability Disclosure & Emerging Policy

## Vulnerability Disclosure Policy

- Published at ssp.hdfgroup.org
- Defines reporting, triage, and response timelines
- Coordinated disclosure with MITRE CVE
- Community feedback loop for policy refinement
- Transparent communication strategy

## Emerging Policy Challenges

- Software Bill of Materials (SBOM) generation
- Provenance and reproducible builds
- AI-generated code review policies
- Secure dynamic plugin loading
- FIPS, NIST, ITAR compliance mapping

ssp.hdfgroup.org

# SBOM & OpenSSF: A Practical Starting Point

### Software Bill of Materials

- Enumerate every dependency — direct and transitive
- CycloneDX format for machine-readable SBOMs
- Attach SBOMs to every release artifact
- Automated CVE scanning against SBOM contents
- Extend to ecosystem: h5py, PyTables, filters
- Provenance chain: source → build → binary → user

### OpenSSF: Where We Start

- Best Practices Badge
- Scorecard: automated security health checks
- SLSA framework for build provenance levels
- Sigstore for signing releases and plugins
- Graph for Understanding Artifact Composition (GUAC) for aggregating SBOM + vulnerability data into queryable graph
- Allstar for enforcing repo security policies

*If you don't know what's in your build, you can't secure it.*

# Reusable Strategies for Scientific Software

1. **Risk Triage for Performance-Sensitive Code**
   Balancing security hardening with strict performance requirements — automated regression benchmarks gate every mitigation.

2. **Legacy Compatibility as a Threat Surface**
   Forward/backward compatibility reduces upgrade incentives. Document and communicate the SSP cost of staying on old versions.

3. **Reproducibility-Aware Security**
   Signed builds, SBOMs, and pinned dependencies protect both scientific reproducibility and supply-chain integrity.

4. **Community-Driven Threat Modeling**
   Segment users by sector and use pattern. Co-develop threat scenarios — cloud, HPC, IoT, regulated data — with the people who know them best.

*These patterns apply beyond HDF5 — to any long-lived scientific infrastructure with broad downstream adoption.*

# Call to Action: Collaborate With Us

**Co-develop threat scenarios**

Cloud-based, HPC, IoT, and regulated data environments

**Validate risk priorities**

Help us rank vulnerabilities by real-world impact in your domain

**Partner on trusted distribution**

Signed packages, reproducible builds, SBOM adoption

**Join the SSP Working Group**

Shape governance, policies, and long-term security culture

**Give us your feedback**

Scan the QR code and take the SSP survey

# Thank You

## Questions & Discussion

**M. Scot Breitenfeld**

The HDF Group

ssp.hdfgroup.org

hdfgroup.org

**Give us your feedback**

Scan the QR code and take the SSP survey

This material is based upon work supported by the 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.