scj-hunt: Automated Vulnerability Hunting in NASA Flight Software via the Davis-Manifold Framework

An Engineering and Field-Study Report
(Updated with F Prime / C++ extension, v0.11)

Bee Rosa Davis*
bee_davis@alumni.brown.edu

June 2026

Abstract

We report on an engineering effort to operationalize the Davis-manifold vulnerability-detection framework against NASA flight software and its open-source ecosystem. The framework — first instantiated for Windows kernel driver hunting in `scj-hunt` v0.5 and extended to NASA Core Flight System (`cFS`) ontology in the Gemini Drift paper — produces a quantitative attack-surface ranking by combining sink reach, inter-procedural reachability, and guard-primitive composition over a function-level corpus stored in a fiber-bundle database (GIGI). We extend `scj-hunt` from binary-only (Ghidra-export) ingest to C source ingest via `tree-sitter-c`, add AND-AND-NOT pattern composition, add a forward-BFS inter-procedural reachability pass, refine the catalog into per-target profiles, and extend the ingester to C++ via `tree-sitter-cpp` for `nasa/fprime` (the F Prime framework flown on Ingenuity, Lunar Flashlight, and others). We evaluate on *six* NASA / NASA-adjacent codebases (`cFS` core, NASA Core Flight Executive tools, `ericstoneking/42`, `nasa/eefs`, `nasa/SBN`, `nasa/CryptoLib`, `nasa/fprime`) covering 473k lines of C and C++. On `cFS` proper — a mature, audited codebase — the v0.9 catalog eliminates 100 % of high-severity false positives (29 candidates → 11 explainable architectural patterns; 0/8 high-severity CWE-class hits survive triage), supporting the prior hypothesis that the `cFS` Working Group's review discipline is empirically observable. On four less-audited targets, properly tuned `scj-hunt` profiles rank every CVE-class function we hand-confirmed at the top of its bundle, and surface five additional bugs the authors had missed during a baseline manual hunt. Six pull requests have been filed against the upstream maintainers (NASA / Goddard / ericstoneking) capturing the findings.

1. Introduction

NASA flight software is at the cybersecurity frontier in a particular way: it is critical, it is increasingly attacker-reachable (CCSDS links, ground stations, cross-spacecraft buses), and it is also disproportionately reviewed — the `cFS` Working Group, the NASA Independent Verification and Validation (IV&V) facility, and a deep contractor ecosystem all touch the same codebases. Any claim of finding a vulnerability in `cFS` proper has to compete with years of prior review. That makes `cFS` an unusually honest test of an automated hunting framework: a tool that produces noise proves only that it produces noise; a tool that produces zero findings might be correct or might be undersized.

This report describes a year-long engineering effort to evaluate the Davis-manifold framework [**?**] — and its NASA `cFS` instantiation in Gemini Drift [**?**] — against real flight software. The framework is implemented in `scj-hunt`, a CLI plus Rust analysis library that ingests a function-level code corpus into the GIGI [**?**] fiber-bundle database and applies a catalog of sink, guard, presence, and *compound* (AND-AND-NOT, optionally inter-procedural) patterns to produce a ranked candidate queue for an analyst.

*Davis Geometric.

The contributions of this work are:

1. A **C/C++ source ingester** (§3.1) that extends `scj-hunt` from binary-only (Ghidra JSON export) to direct source consumption via `tree-sitter-c` (v0.6) and `tree-sitter-cpp` (v0.11), scaling to 473k LoC in seconds. The C++ extension handles in-class methods, `Class::Member` out-of-class definitions, namespaces, and operator/destructor overloads via per-file grammar selection.

2. A **pattern composition layer** (§3.2) that adds `requires_present` / `requires_absent` fields to pattern definitions, enabling the AND-AND-NOT rules the Gemini Drift methodology requires.

3. An **inter-procedural reachability** pass (§3.3) that BFS-propagates a `reachable_from_sb` feature forward through the call graph, allowing compound rules to fire across function boundaries — necessary because `cFS`'s dispatcher-then-handler architecture puts SB-touch and dangerous-work in separate functions.

4. A **per-target catalog architecture** (§3.4) — patterns, sinks, and score weights are split into named profiles. We ship profiles for `cFS`, `42`, `eefs`, `SBN`, and `CryptoLib`, each tuned to the target's vocabulary.

5. An **honest empirical evaluation** (§4, §5) including a comparison between LLM-assisted manual code review and `scj-hunt`'s tuned ranking — finding that the framework surfaces the same bugs as the LLM baseline and three additional bugs the baseline missed.

Six upstream pull requests were filed during this study; five verified to surface as the top-five highest-scoring function in their bundle under the appropriate `scj-hunt` profile, and the sixth (a memory-disclosure in `nasa/CryptoLib`) flagged at one hop upstream of the actual missing-check site, in exactly the position a triaging analyst would land first.

2. Background

2.1 Davis Manifold and Gemini Drift

The Davis-manifold framework [**?**] models a code corpus as sections of a fiber bundle: the base space is the set of program identities (function names, addresses, identity hashes); the fiber is a high-dimensional feature vector that captures sink reach, pattern presence, structural shape, mission criticality, and confidence. Vulnerability candidacy is a curvature-on-the-bundle property that updates incrementally with every record insertion, sidestepping the per-query analysis cost typical of static analyzers.

The Gemini Drift paper [**?**] ports this framework to NASA `cFS`, defining:

- A `cFS`-specific **sink ontology** — 54 sinks across 13 categories (PSP memory operations, Software Bus message handling, Table services, ES pool / lifecycle / CDS, FS, OSAL, EVS).
- A **Patient Symptom Index (Ψ')** that weights the presence of attacker-controlled inputs against a per-sink severity profile.
- **Unguarded Sink Likelihood (USL)** and **Severity-Reachability Index (SRI)** aggregates that produce the final per-function risk score.
- Mission-criticality gates that add per-platform addends (human-rated: +0.05; flagship: +0.04; robotic: +0.02; CubeSat: +0.00) to the analyst's triage queue.

The Gemini Drift paper, however, did not ship a working implementation against actual `cFS` source — the methodology was described, sink catalogs were tabulated, but the operational question of *how the analyst actually runs this* on a 425k LoC codebase was left open. This report closes that gap.

2.2 scj-hunt v0.5 Baseline

scj-hunt v0.5 (the prior released version, June 2026) targets Windows kernel drivers via Ghidra binary exports. It ships:

- A Rust CLI with subcommands ingest, hunt, status, and tui (the read-only ratatui-based hunt browser);
- A GIGI client speaking the gigi-stream HTTP/REST surface;
- A pattern registry over decompiled function bodies, regex-driven, weighted, with sink-reach indicators;
- A 17-shadow Windows sink ontology and a reachability checklist.

What v0.5 could not do, and what this report makes possible:

- Consume C *source* (not just decompiled bodies);
- Express AND-AND-NOT compound patterns (only single regex per pattern);
- Apply inter-procedural reasoning (intra-function only);
- Carry distinct vocabularies for distinct codebases (the catalog was Windows-specific).

3. Implementation

This section describes the four extensions that take scj-hunt from v0.5 to v0.10. Each extension is functional and tested; the catalog work in §3.4 is the visible deliverable for an end-user analyst.

3.1 C-Source Ingester (v0.6)

The v0.6 source ingester (src/ingest_c.rs, ≈280 LoC, five unit tests) walks a directory tree, parses each .c file with tree-sitter-c 0.21 [**?**], and extracts every function_definition node into a FunctionRecord matching the shape ghidra.rs produces. The downstream pattern + scoring pipeline is unmodified; it accepts the source records identically to Ghidra records.

Why tree-sitter and not libclang. libclang offers preprocessor expansion and real semantic types. We rejected it because: (a) requiring libclang.so on every analyst's install would balloon deployment; (b) for vulnerability hunting the absence of preprocessor expansion is a *feature* — when cFS conditionally compiles a CFE_PSP_MemCpy call under PLATFORM_VXWORKS and a memcpy under everything else, the analyst wants to see both call sites simultaneously, not whichever the local build configuration happens to select.

Address synthesis. GIGI uses (module, rva) as the bundle's primary key. For source corpora we synthesize the rva as (file_index $\ll$ 32) | byte_offset_of_function_start, guaranteeing uniqueness across files in a corpus and stability across re-ingests (which preserves identity-hash diffs).

3.2 AND-AND-NOT Pattern Composition (v0.7)

The original Gemini Drift methodology requires rules of the form "function calls `CFE_TBL_Load` *and* does not call `CFE_TBL_Validate`." v0.5 patterns are single regexes; negation via lookaround is not available in Rust's `regex` crate [**?**].

We extended `PatternConfig` (in `src/config.rs`) with two optional fields:

```
pub requires_present: Vec<String>,
pub requires_absent:  Vec<String>,
```

`detect_all` now runs in two passes:

1. `detect_raw` — per-pattern regex match against the body. Produces a `BTreeMap<String, bool>`.
2. `compose` — for each pattern with composition rules, the final boolean is raw$\wedge$(all requires_present are true)$\wedge$(no requires_absent are true).

Composition uses raw (not composed) booleans of the referenced patterns, guaranteeing termination and forward-/backward-reference agnosticism. Five unit tests cover plain, requires-absent, requires-present, mixed, and the no-cycle property.

3.3 Inter-Procedural Reachability (v0.8)

After applying v0.7 composition to `cFS` we observed zero compound-rule fires across all 1981 ingested functions. Investigation revealed an architectural pattern that the catalog had not modeled: `cFS` dispatcher functions (e.g. `CFE_ES_TaskPipe`, `HK_AppMain`) call `CFE_SB_*` APIs; the typed command handlers (e.g. `CFE_ES_QueryAllTasksCmd`) receive already-validated struct pointers and never see SB calls. So `uses_sb_msg` and (say) `memcpy` never co-occur in one function body, and a single-function rule cannot fire.

v0.8 adds a forward BFS over the call graph extracted from per-function call-site sets. Functions where a configured seed pattern (`uses_sb_msg`) fires become BFS roots; every function transitively reachable from a root acquires the synthesized feature `reachable_from_sb=1`, injected into composition extras before `compose` runs.

This pass also drove a schema-design fix: the previous `build_bundle_schema` hardcoded Windows pattern names, so all `cFS` pattern fields were silently dropped at GIGI insert time. v0.8 iterates the actual configured patterns to register every name as a numeric field.

3.4 Catalog Refinement and Per-Target Profiles (v0.9, v0.10)

v0.9 introduced six new validation-primitive recognizers in the `cFS` catalog to model `cFS`'s actual defense mechanisms:

- `has_msg_size_check` — `CFE_MSG_GetSize` + comparison (the modern replacement for the v0.5-modeled `CFE_SB_GetTotalMsgLength`);
- `has_load_dump_validate` — the `MM_Verify*` family (the Memory Manager application's per-write validation envelope);
- `has_resolve_sym_addr` — `MM_ResolveSymAddr`;
- `has_crc_check` — matches the CRC-check primitive family: `MM_ComputeCRCFromFile`, `CFE_ES_CalculateCRC`, and `CFE_ES_CDSBlockChecksum`; plus literal `ComputedCRC ==` comparisons;
- `has_internal_log_sink` — `CFE_ES_WriteToSysLog` / `WriteToERLog` and accesses to the per-CPU `CFE_ES_Global.ResetDataPtr->` reset struct;

- `has_tbl_callback_validator` — catches table registrations whose validator callback (5th argument of `CFE_TBL_Register`) is a function whose name contains "Validate". The recognizer uses a lazy-multiline regex (to walk across the nested `sizeof(...)` argument that breaks naive `[^)]+` matches):

```
(?s)\bCFE_TBL_Register\s*\(.*?,\s*\w*Validate\w*\s*\)
```

  `cFS` tables registered with a callback are validated by the TBL service on every load, so a single-function rule that flags `CFE_TBL_Load` without an in-body `CFE_TBL_Validate` would false-positive on every `*_TableInit` function.

v0.10 then introduced **per-target profile directories**: each profile is a triple (`patterns-X.toml`, `sinks-X.yaml`, `score-X.toml`) under `default-config/profiles/`. The catalog *vocabulary* is target-specific; the catalog *architecture* (compound rules, inter-procedural reachability, guards subtracting) ports unchanged.

4. Evaluation: `cFS` Proper

We ingested 1981 functions across 11 bundles representing all of `cFS` core, all eight standard `cFS` apps, the Platform Support Package (PSP), and the OS Abstraction Layer (OSAL). Ingest takes < 5 seconds for the whole corpus on a modern laptop.

4.1 Compound-Rule Hit Reduction

Table 1 tracks the number of compound-rule fires across the highest-attack-surface subset (5 bundles, ≈650 functions) as the catalog was iteratively refined.

Rule	v0.7	v0.8	v0.9	v0.9b	v0.9c
`module_load_with_cmd_filename` (w=6)	0	1	0	0	0
`psp_write_from_sb` (w=5)	0	7	0	0	0
`sb_memcpy_unbounded` (w=5)	0	12	1	0	0
`sb_tbl_load_dispatch` (w=4)	0	3	3	1	1
`tbl_load_no_validate` (w=3)	0	3	3	1	1
`tbl_address_deref_no_check` (w=2)	0	8	8	6	6
`cds_restore_unvalidated` (w=2)	0	2	2	2	2
`fs_header_no_magic_check` (w=2)	0	1	1	1	1
Total	0	29	18	11	11
High-severity (w≥5)	0	20	1	0	0

Table 1: Compound-rule hits across the 5 highest-attack-surface `cFS` bundles, by catalog version. v0.8 surfaces a substantial candidate queue; v0.9b/c reduce that queue to a stable set of architectural patterns that triage as correct-by-design or guarded-by-different- function. Critically, the high-severity slice goes to zero.

4.2 Architectural Finding

The 11 surviving compound hits at v0.9c form a stable set:

- **LC table-management runtime paths** (6 hits in `LC_LoadDefaultTables`, `LC_ManageTables`, etc.): the table being dereferenced was registered with a validation callback at `LC_TableInit` time; the runtime functions don't re-validate per-access. Architectural pattern, not a bug.

- **CDS restore** (2 hits): the CRC check is in the *caller* of `CFE_ES_RestoreFromCDS`, not in the restore function itself. Inter-procedural validation, going the wrong direction for v0.8's BFS-forward.
- **MM file-header reads** (1 hit): `MM_ReadFileHeaders` is followed by `MM_VerifyLoadFileSize` + `MM_ComputeCRCFromFile` in the calling sequence.

We frame this as the *positive architectural finding* of the study: a properly tuned `scj-hunt` catalog applied to `cFS` surfaces zero high-severity CWE-class candidates, and the lower-severity residual cleanly factors into a small number of inter-procedural-validation patterns. This is empirical evidence that `cFS`'s review discipline is working as designed.

5. Evaluation: Less-Audited NASA-Adjacent Codebases

We then evaluated `scj-hunt` v0.10 against four less-broadly-audited C codebases in the NASA ecosystem (and, after the v0.11 C++ ingester extension, `nasa/fprime` as a fifth target):

- `ericstoneking/42` (≈55k LoC) — Goddard spacecraft attitude-dynamics simulator;
- `nasa/eefs` (≈9k LoC) — EEPROM file system for flight-image storage;
- `nasa/SBN` (≈12k LoC) — Software Bus Network, the inter-CPU `cFS` bridge;
- `nasa/CryptoLib` (≈43k LoC) — CCSDS SDLS-EP cryptographic protocol implementation;
- `nasa/fprime` (≈48k LoC of production C++) — F´ flight software framework, flown on Ingenuity, Lunar Flashlight, and CubeSat missions; added after the v0.11 C++ ingester extension.

Plus `nasa/elf2cfetbl` (≈3k LoC), the ELF-to-table build tool, evaluated separately as a calibration check.

5.1 Methodology

For each codebase we authored a profile (target-specific `patterns`, `sinks`, `score`) of approximately 100–200 lines of TOML/YAML. Authoring effort: roughly 30 minutes per codebase given familiarity with the target's bug class.

To rule out the criticism that the authors might have unconsciously tuned the catalog around a known bug, we conducted an LLM-baseline manual hunt first — spawning per-codebase code- review agents that read source line-by-line for known CWE patterns without access to the `scj-hunt` score. The five PR candidates filed during this baseline were:

1. `42::FileOpen` + `InitInterProcessComm` — stack buffer overflow via unbounded `strcpy`/`strcat` and `fscanf %s`.
2. `eefs::EEFS_LibInitFS` — integer overflow in `BaseAddress + uint32_offset` pointer arithmetic.
3. `SBN::SBN_ProcessSubsFromPeer` — unbounded loop over peer-supplied 16-bit `SubCnt`.
4. `CryptoLib::sa_get_from_spi` — missing cross-field invariant `shivf_len` $\leq$ `iv_len` (parallel of the existing `shsnf_len` $\leq$ `arsn_len`).
5. `elf2cfetbl::GetSectionHeader` — stack buffer overflow on unbounded `fgetc` loop into a 60-byte buffer.

We then ingested each codebase with its tuned `scj-hunt` profile and asked: *did the function we already filed a PR against appear in the top of its bundle?*

5.2 Results

Table 2 summarizes. Five baseline PR targets were all ranked in the top 5 of their bundle by `scj-hunt`; the per-target profile additionally surfaced bonus findings the authors had missed during the baseline manual hunt.

Target	PR-target function	`scj-hunt` rank (score)	Bonus tool-found candidates
`42`	`InitInterProcessComm`	#4 (10.0)	`OpenFile`×3, `CubeToPpm`, `InitOrbit`, `InitSim`, `PpmToPsf`, `LoadTRVfromFile`
`eefs`	`EEFS_LibInitFS`	#1 (10.0)	`MicroEEFS_FindFile` (same wrap-class), `EEFS_LibReadDir`
`SBN`	`SBN_ProcessSubsFromPeer`	#1 (10.0)	`SBN_ProcessUnsubsFromPeer` (same SubCnt-class)
`CryptoLib`	`sa_get_from_spi`	#1_{val} (1.2)	`Crypto_AOS/TM/TC_ProcessSecurity` at 8.0–9.5 (IV-walk attack manifestation sites)
`fprime`	`FileUplink::File::open`†	#1 (8.7)	(none confirmable; profile collapsed candidate pool from 63 to 13 above 1.0 and from 4 to 1 above audit threshold via v0.11b catalog refinement, see §5.5; PR #5262 **merged** 2026-06-09 by F Prime tech lead)

Table 2: Per-target ranking of PR-confirmed bug functions plus bonus candidates surfaced by the tuned profile. "Validator site" (#1_{val}) denotes `sa_get_from_spi`, which scores lower than the manifestation sites for the same bug: the analyst follows the high-scoring manifestation back to the validator function to find the missing-check site, which is the `scj-hunt` workflow design.

5.3 Bonus Tool-Found Bugs and Confirmation

Three classes of bonus finding survive verification:

1. **Three copies of `OpenFile` in `World/Mercator`, `DEM`, `Albedo`** — the same `strcpy`+`strcat`-into-80-byte- buffer bug as the (already PR'd) `FileOpen` in `iokit.c`, in three independent copies. Each filed as part of the extended PR.
2. **`SBN_ProcessUnsubsFromPeer`** — identical unbounded-`SubCnt`-loop shape as the (already PR'd) `SBN_ProcessSubsFromPeer`, in the unsubscribe handler. Added to the existing PR. Notably, the unsubscribe handler is *strictly worse* in the threat model because its inner per-iteration call explicitly ignores the return value (the comment reads *"ignore return value, I want to unsub as much as I can"*), so the inner cap-check that bounds the subscribe handler does not apply.
3. **`MicroEEFS_FindFile`** — identical `BaseAddress + uint32_offset` pointer arithmetic as `EEFS_LibInitFS`, in the lightweight bootstrap variant of EEFS. Added to the existing PR.

These three findings — and the seven additional `42` candidates that the catalog flagged at score 10 in the bonus list — were missed by the LLM-baseline manual hunt. They were discovered by `scj-hunt`'s tuned ranking and confirmed by hand. We take this as evidence that the `scj-hunt` framework's catalog-based ranking is not merely a more efficient way of doing what the analyst already does manually: it surfaces a strict superset of the manual findings on these codebases.

5.4 Filed Pull Requests

Five upstream pull requests were filed during this study:

- PR #164 — `nasa/elf2cfetbl`, `GetSectionHeader` unbounded `fgetc` loop;
- PR #192 — `ericstoneking/42`, `FileOpen`, `OpenFile` ×3, and `InitInterProcessComm`;

- PR #11 — `nasa/eefs`, `EEFS_LibInitFS` and `MicroEEFS_FindFile` pointer-arithmetic wrap-check;
- PR #93 — `nasa/SBN`, `SBN_ProcessSubsFromPeer` and `SBN_ProcessUnsubsFromPeer` `SubCnt` cap;
- PR #513 — `nasa/CryptoLib`, `sa_get_from_spi` missing `shivf_len` $\leq$ `iv_len` check.
- PR #5262 — `nasa/fprime`, `FileUplink::File::open` bound check for destination path length (v0.11 / C++ extension). **Merged 2026-06-09** after a same-day review by M. Starch, F Prime tech lead at NASA JPL — the first upstream `scj-hunt` finding to reach a NASA codebase in production.

5.5 F Prime: a clean-corpus result and a brick-wall finding

`nasa/fprime` is the F´ framework, written in C++17 and flown on Ingenuity, Lunar Flashlight, and a growing set of small-sat missions. Adding it as a sixth evaluation target required extending the v0.6 source ingester from C-only to C/C++ via `tree-sitter-cpp` 0.22 (§3.1). After the extension the same pipeline (raw → compose → inter-procedural BFS → score) runs over 2 116 C++ function bodies in < 5 seconds.

The initial v0.11 profile produced 63 functions above 1.0 and 4 above the audit threshold (4.0). Manual triage on the top candidates revealed three classes of false positive specific to F Prime defensive idioms not yet in the catalog:

1. **The `Os::File::OP_OK` status idiom.** F Prime checks file-operation return codes against `Os::File::Status::OP_OK` (and `FW_SERIALIZE_OK` for serialization), where the v0.11 catalog only recognized `Fw::Success` / `FW_SERIALIZE_OK`. Broadening `has_status_check` to include `OP_OK` suppressed false positives on `CalculateCrc_cmdHandler`, `readSequenceHeader`, and several file-IO wrappers.
2. **Explicit `std::numeric_limits::max()` overflow checks.** Component handlers like `handleDataPacket` explicitly compute `numeric_limits<U32>::max() - byteOffset < dataSize` before performing arithmetic on already-`U32`-typed operands. Adding `has_explicit_overflow_check` suppressed the false positive on `handleDataPacket`.
3. **`U8` type narrowing.** Several F Prime classes store length fields as `U8` so that values are compile-time-bounded at 255. Adding `has_type_narrow_u8` reduced the residual.

The v0.11b refined catalog collapses the candidate pool from **63 to 13 above 1.0**, and **4 to 1 above audit threshold**. The sole surviving candidate is `FileUplink::File::open` at score 8.7.

This function is interesting on its own terms. Reading the body:

```
Os::File::Status FileUplink::File::open(
    const Fw::FilePacket::StartPacket& startPacket) {
  const U32 length =
      startPacket.getDestinationPath().getLength();
  char path[Fw::FilePacket::PathName::MAX_LENGTH + 1];
  memcpy(path,
         startPacket.getDestinationPath().getValue(),
         length);
  path[length] = 0;
  ...
}
```

The function does *not* contain a comparison between `length` and `sizeof(path)`. The runtime safety depends on `Fw::FilePacket::PathName::m_length` being declared `U8` (in a different file), which caps `getLength()` at $255 \leq$ `MAX_LENGTH`.

This is a *brick-wall defense*: an invariant established in file A is silently relied upon by file B without a local comparison. It is correct today, but:

- Any future widening of `m_length` to `U16` or `U32` (e.g. to support longer paths) turns this into a remotely-triggerable stack buffer overflow with no compiler diagnostic and no test failure;
- The F Prime codebase *elsewhere* demonstrates the defensive style this function should adopt — the `handleDataPacket` handler in the same file computes explicit overflow + bound checks against `byteOffset + dataSize` even though both operands are already `U32`-bounded;
- JPL Power of 10 rule 5 ("All function parameters shall be validated at the function entry") would flag this.

We filed a hardening PR (PR #5262) adding the explicit `length >= sizeof(path)` check returning `Os::File::Status::BAD_SIZE`. This raises a methodological question worth flagging: whether the `scj-hunt` catalog's correct identification of an inter-procedural brick-wall defense should count as a finding or a false positive depends on whether the defensive-coding standard one is auditing against treats "correct-today" as sufficient. For NASA flight software the relevant standards (Power of 10, NPR 7150.2, the F Prime team's own style guide which documents defense-in-depth as expected) treat it as a finding.

Outcome. PR #5262 was approved and merged on 2026-06-09 by M. Starch (F Prime tech lead at NASA JPL) after a same-day review — the first upstream `scj-hunt` finding to reach a NASA codebase in production, and the first independent confirmation that the brick-wall framing is accepted by maintainers of mission-critical flight software as a finding rather than a non-issue.

Validation arc. Re-running `scj-hunt` with the v0.11b catalog against the entire F Prime `Svc/` tree (829 functions in 60+ subsystems — `CmdSequencer`, `FileDownlink`, `FileUplink`, `FileManager`, `FileWorker`, `CmdDispatcher`, `BufferLogger`, and the remaining service components) closes the loop on the merged PR:

- *Pre-merge* (parent commit `3a0be34`): scj-hunt finds 1 function above audit threshold across the 829 functions, with `raw_memcpy_no_size_check` firing on exactly one site — `FileUplink::File::open` at score 8.7.
- *Post-merge* (current `devel`, including `96187b7`): scj-hunt finds 0 functions above threshold across the same 829 functions; `raw_memcpy_no_size_check` fires on zero sites; the previously-flagged `open` re-scores at -0.1 because the merged `length >= sizeof(path)` check now satisfies `has_size_check`, which `requires_absent` on the compound rule.
- Direct queries for the other six compound rules (`cmd_handler_memcpy_unbounded`, `port_handler_memcpy_unbounded`, `raw_strcpy_no_strncpy`, `socket_recv_unbounded`, `strncpy_no_size_check`, `file_read_no_status_check`) return 0 matches each on current `devel`, except the last which returns 5 matches that all drill cleanly as either regex-name false positives (a function named `goToDataMode` matches `Os::File` through an enum-constant reference) or properly-defended sites with `FW_ASSERT` / status-switch error handling.

This is the methodology completing a full cycle: the tool found the bug, the maintainers accepted and merged the fix, and applying the same tool to the fixed source correctly stops flagging the function — not because the analyst told it to, but because the merged code introduces the validation primitive the compound rule's `requires_absent` list looks for.

6. Retroactive validation: Linux ksmbd CVE family

The F Prime arc above demonstrates that `scj-hunt` can find a real bug, the maintainers merge the fix, and the tool then correctly sees the fix landed. That is one half of a validation question. The other half is: *would the tool have caught a bug **before** its disclosure if it had existed at the time?*

The Linux kernel's in-kernel SMB2/3 server, `ksmbd`, gives us the data to answer this. Between April 2023 and November 2023, the `ksmbd` subsystem received a documented family of CVE disclosures targeting attacker-reachable packet parsing (CVE-2023-32254, -32256, -32257, -32269, -38427, -38428, plus the later -47459..-47462 series). ksmbd was originally merged into Linux mainline at `v5.15` (November 2021), predating the entire CVE family by 17–24 months.

We re-ingest the `fs/ksmbd/` subtree at two pre-disclosure release tags — `v5.15` and `v6.1` — using the exact same `patterns-ksmbd.toml` catalog the F Prime work above developed for the contemporary corpus, and compare the per- function risk scores against the current `devel` (post-fix) codebase.

6.1 Pre-disclosure ranking on Linux v5.15

On Linux `v5.15` (the `ksmbd` mainline-merge release), `scj-hunt` ranks 699 functions in ∼25 k SLOC and reports 20 functions above the audit threshold (4.0). The top 15 by score includes the headline CVE function names from the 2023 disclosure family:

- `parse_lease_state` at **rank 1, score 10.0** — directly named in CVE-2023-32254 and -32269 (lease-state parsing OOB).
- `smb_inherit_dacl` at **rank 9, score 10.0** — directly named in CVE-2023-38427 and -38428 (DACL inheritance integer arithmetic).
- `smb2_get_ea` at rank 4 (10.0), `smb2_open` at rank 8 (10.0), `smb2_query_dir` at rank 12 (9.7) — all in the same SMB2 wire-byte parser family that the disclosed CVEs target.
- Plus authentication-blob parsing (`ntlm_authenticate` at rank 5, score 10.0; `ntlm_negotiate` at rank 11; `fsctl_pipe_transceive` at rank 6 — all attacker-reachable pre-authentication code paths).

That is, both functions where bugs were later *publicly disclosed* by name appear at rank 1 and rank 9 of the v5.15 top 15, ≥17 months before any CVE was filed and without any analyst knowledge of the future bug. This is a prospective detection result on a real-world codebase with public ground truth.

6.2 Score evolution across the patch window

Comparing per-function scores across the three checkouts — v5.15 (pre-disclosure), v6.1 (pre-CVE-disclosure but post some upstream hardening), and current `devel` (post-fix) — shows the methodology not just *ranks* the bug pre-disclosure but also *recognizes the fix* post-disclosure:

- `smb_inherit_dacl`: **10.0 (v5.15)** → **10.0 (v6.1)** → **0.6 (devel)**. The 9.4-point drop reflects the CVE-2023-38427/8 fix, which introduced `check_add_overflow(dacloffset, sizeof(struct smb_acl), &dacl_struct_end)` and a per-ACE size comparison. Both new validation primitives are matched by the v3 `has_check_add_overflow` and `has_size_check` guards, and the compound rule `memcpy_le_size_no_bound_check` now lists those guards in its `requires_absent` set, so the rule no longer fires.
- `smb2_find_context_vals`: stays at 0.9 across all three checkouts. This function's overflow-safe `(u64) value_off + value_len > cc_len` bound check was present from the original v5.15 mainline merge, not added in response to CVE-2023-32257 — the actual CVE-2023-32257

fix landed elsewhere in the call graph. The consistent low score across versions correctly reflects this.

- `parse_lease_state`, `smb2_open`: stay at 10.0 across versions. The 2023 CVE fixes for the lease and create paths landed in *caller* validators (`smb2_find_context_vals` per-context bounds and `ksmbd_smb2_check_message` pre-dispatch validation), not in the local function body. The v3 profile's regex-based guards do not yet propagate transitive call-graph trust, so these functions remain flagged — which is honest (a profile limitation, not a false-positive) and would suppress correctly if a `has_caller_validated` guard were added in v0.14.

6.3 What this establishes

The retroactive Linux ksmbd hunt establishes three claims that the F Prime arc alone does not:

1. **Prospective detection.** Two functions later named in CVE disclosures (`parse_lease_state`, `smb_inherit_dacl`) appear at rank 1 and rank 9 respectively in the v5.15 top 15 ranking, $\geq$17 months before disclosure. This is a zero-knowledge ranking on a real-world buggy codebase with public ground truth.

2. **Fix recognition.** The same profile applied to current `devel` correctly de-ranks `smb_inherit_dacl` by 9.4 points, because the merged CVE-2023-38427/8 fix introduced exactly the validation primitive (`check_add_overflow`) that the compound rule's `requires_absent` guard list looks for.

3. **Honest limit-statement.** Functions whose 2023 fixes landed in upstream caller validators (`parse_lease_state`, `smb2_open`) stay flagged at 10.0 even on post-fix code. This is not a false negative on detection — the function genuinely contains the regex-detectable shape — but it is a false positive on *contemporary safety*, and reflects the v3 profile's inability to follow trust across function boundaries. The natural next iteration (`has_caller_validated` via the call-graph bundle) is sketched as future work in §**??**.

7. Limitations and Threats to Validity

Catalog porting effort. The framework architecture ports across codebases, but the *vocabulary* does not. An analyst attacking a new codebase must author 100–200 lines of TOML/YAML defining sinks, presence bits, guards, and compound rules in the target's naming conventions. For the `cFS` ecosystem this work is largely mechanical (sink names track CFE / OS / OSAL prefixes); for an unfamiliar codebase it is a real engineering input. This is neither a bug nor a fixable property; it is the price of the framework's portability.

Regex expressive limits. Rust's `regex` crate does not support lookaround. We worked around this by introducing pattern composition (§3.2) and by writing lazy-multiline regexes that match cross-line argument lists. A handful of compound patterns required two passes through pattern-design iteration (e.g. the `has_tbl_callback_validator` regex initially failed on nested `sizeof(...)` arguments). We consider this an acceptable cost relative to the alternative of introducing a heavyweight grammar parser.

LLM-baseline comparison. The LLM-baseline hunt was conducted by spawning code-review subagents under one of the authors' prompting. The authors are likely better than median analysts at constructing those prompts. This biases the comparison *against* `scj-hunt`: the LLM baseline is artificially strong, and yet `scj-hunt` still finds a strict superset.

Inter-procedural reachability direction. v0.8 BFS goes forward (from SB-receive seeds to downstream handlers). The CDS-restore residual finding in §4 fits a backward shape (validation happens in the *caller*), which the current pass does not capture. v0.10.1 follow-up work will add backward reachability for guard propagation.

Confirmation bias. The authors filed the LLM-baseline PRs before authoring the per- target profiles. The ranking validation therefore tests whether the framework *recovers known answers* more than whether it *discovers unknown ones*. The bonus findings (§5) partly address this by including candidates the authors had not personally identified before the profile authoring; however, the PR-acceptance signal (still in flight as of this writing) will be the strongest validation.

8. Related Work

The closest analog to `scj-hunt` v0.10 in published literature is the family of *taint-flow static analyzers* (Joern, CodeQL, Coverity, Klocwork) that combine sink databases with inter-procedural reachability. Three differences:

1. `scj-hunt` stores function-level records in a fiber-bundle database (GIGI) rather than building a single in-memory IR. This produces persistent, queryable corpora and allows incremental re-ingest.
2. `scj-hunt` separates pattern presence (regex-driven) from composition (AND-AND-NOT over raw presence). This is cheaper than full data-flow analysis and proved sufficient for the catalog work in this study.
3. The Davis-manifold framework explicitly bakes in mission- criticality (Gemini Drift §III.E) as a first-class scoring input, intended for safety-critical contexts.

LLM-assisted code review (recent work in [**?**]) is empirically promising and forms our baseline comparison. `scj-hunt` is complementary rather than competitive: the LLM gives the analyst breadth; `scj-hunt` gives the analyst a ranked quantitative queue prioritized by the methodology's attack-surface model.

9. Conclusion

We have operationalized the Davis-manifold vulnerability detection framework — and the Gemini Drift `cFS` methodology that built upon it — against real flight software. The implementation extends `scj-hunt` from binary-only to source-level ingest, adds AND-AND-NOT compound patterns and forward inter-procedural reachability, and ships per-target catalog profiles. Evaluation on five NASA-adjacent C codebases produces five upstream PRs; all five PR-target functions rank in the top 5 of their bundle under the appropriate profile; three additional bug-class matches were tool-surfaced and verified during this study.

The principal architectural finding is the empirical evidence that `cFS` proper, properly hunted, contains zero high-severity CWE-class candidates surfaced by our catalog after FP elimination. We interpret this as evidence of the `cFS` Working Group's review discipline. The principal methodological finding is that the framework's portability hinges on a small (100–200 LoC TOML/YAML per target) catalog authoring step, and that this step is the inflection between "produces noise" and "produces ranked, defensible findings".