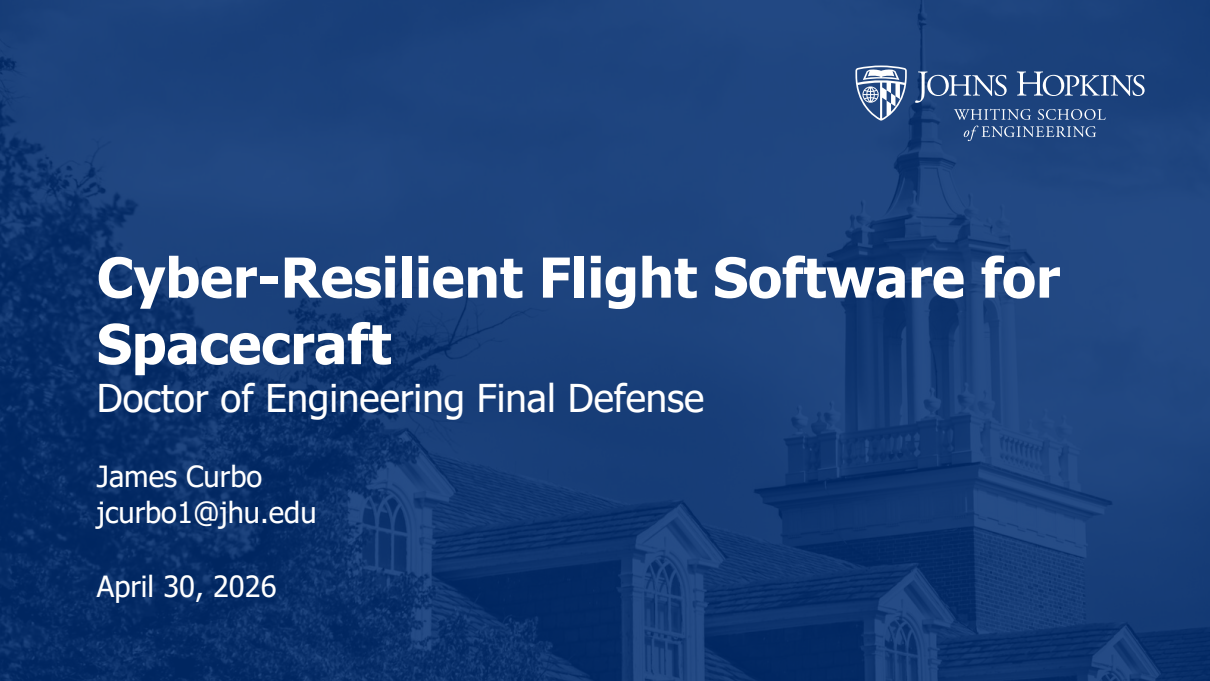


JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Cyber-Resilient Flight Software for Spacecraft

Doctor of Engineering Final Defense

James Curbo
jcurbo1@jhu.edu

April 30, 2026

The Problem

- Increasingly, spacecraft are networked, software-defined, and autonomous
- Compromised flight software can alter commands, modes, telemetry, and fault response
- Heritage FSW architectures emphasize reliability and fault tolerance more than adversarial resilience
- General cybersecurity guidance is difficult to translate into flight-software requirements, architecture, and evidence

Thesis and Research Questions

Cyber resilience can be engineered into flight software as a first-class system property.

- **RQ1**: What attack surfaces and threat scenarios are most consequential for mission integrity in flight software?
- **RQ2**: What minimum secure-by-design principles are necessary, and how can they be translated into formal requirements?
- **RQ3**: How can an architecture embed resilience by construction using explicit trust boundaries, isolation, and enforcement surfaces?
- **RQ4**: How can cyber resilience be demonstrated and measured to evaluate protection effectiveness and mission impact?

Published Work

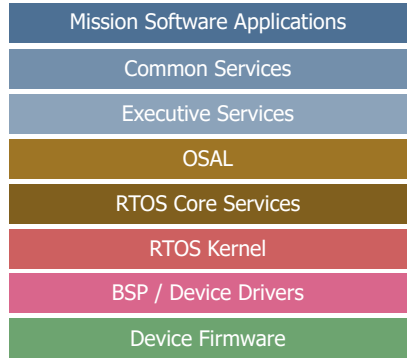
- "A Research Agenda for Space Flight Software Security," IEEE SMC-IT, 2023.
- "Attack Surface Analysis for Spacecraft Flight Software," IEEE SMC-IT, 2024.
- "Minimum Requirements for Space System Cybersecurity," IEEE SMC-IT, 2024.
- "Testable Cyber Requirements for Space Flight Software," IEEE Aerospace Conference, 2025.
- "Alcyone: A Blueprint for Secure Rust Flight Software," IEEE SMC-IT, 2025.
- "Cyber Resilience in Cislunar Space," IEEE SMC-IT, 2025.

RQ1: Threat Landscape

What attack surfaces and threat scenarios are most consequential for mission integrity in flight software?

Heritage FSW: Architectural Gaps

- **Single address space:** one app compromise can reach all apps
- **Implicit internal trust:** buses lack publish-side restrictions
- **Edge-only link auth:** components still need authority checks
- **Memory-unsafe C/C++:** interfaces can become code-execution paths
- **Reliability assumptions:** fault response expects random failure



Threat Mapping with SPARTA

- Space-specific threat taxonomy — ATT&CK for space systems
- Each technique carries a Normalized Risk Score (NRS, 0–25)
- Highest-risk techniques exploit the same gaps: implicit trust, weak isolation, limited onboard authorization

Tactic	Technique	NRS
Initial Access	Malicious commanding via ground station	25
Execution	Exploit code flaws in FSW	25
Execution	Modify on-board values (stored/volatile)	25
Execution	Spoof bus traffic	25
Defense Evasion	Disable fault management	24
Persistence	Software backdoor	24
Lateral Movement	Bus-mediated propagation	24

RQ1 Takeaway

The architectural gaps in heritage FSW — shared address spaces, implicit internal trust, and limited onboard authority enforcement — are not individual bugs. They are structural properties that create compounding attack surfaces.

Bottom-up decomposition reveals risks that top-down threat modeling can miss. These findings define what the requirements must address.

RQ2: Minimum Requirements

What minimum secure-by-design principles are necessary, and how can they be translated into formal requirements?

The Requirements Gap

Space missions often lack tailored flight-software cybersecurity requirements:

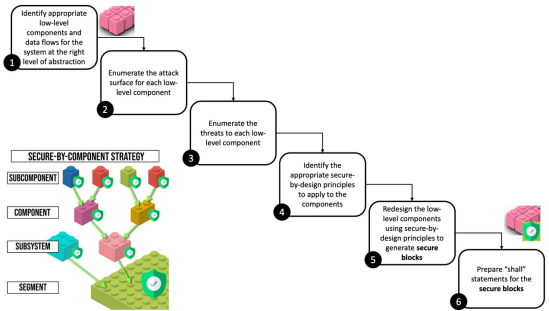
- Boilerplate — projects may inherit generic cyber language without tailoring it to onboard software behavior
- Minimum compliance — standards such as NASA-STD-1006 and CNSS address command authority and link protection, but not the full onboard software design problem
- Control-driven — NIST 800-53 catalog selection says what to protect against, not how the flight software must behave

Generic controls do not by themselves produce testable "shall" statements derived from threat analysis through architecture to verification.

Requirements Derivation

Extends the IEEE P3349
secure-by-component process:

1. Frame with NIST SP 800-160 principles
2. Define mission resilience priority
3. Decompose FSW into components
4. Analyze component attack surfaces
5. Select SPARTA techniques with HAT TRICK
6. Select secure-by-design principles
7. Define secure blocks and "shall" statements
8. Embed requirements into FSW architecture



Source: Viswanathan et al., secure-by-component design strategy.

Four Core Design Principles

1. **Privilege Restriction**
   Per-command authorization; mode-gated command authority
2. **Substantiated Integrity**
   Input validation at every boundary; interface integrity enforcement
3. **Segmentation**
   Process-level isolation; no shared address space
4. **Analytic Monitoring and Adaptive Response**
   Active health monitoring; autonomous fault detection and recovery

Example: Command Reception Requirements

Threat: SPARTA EX-0009 (Exploit code flaws in FSW), NRS 25

- **CR-1** (Substantiated Integrity): The command reception system shall prevent unauthorized data from being written to the command processing subsystem.
  - CR-1-1: Implement RBAC to restrict command entry permissions
  - CR-1-2: Validate all incoming commands against acceptable formats
  - CR-1-3: Log unauthorized write attempts for audit
- **CR-5** (Segmentation): No other subsystem can access unauthorized data written to the command processor.
- **CR-3** (Adaptive Response): Upon detection, report unauthorized data to the responsible entity in real time.

RQ2 Takeaway

The secure-by-component process selects principles and derives requirements from each component's attack surface.

These are more than aspirational goals: they produce testable "shall" statements traceable to specific threats. The question is whether an architecture can enforce them structurally.

RQ3: Architecture by Construction

How can an architecture embed resilience by construction
using explicit trust boundaries, isolation, and enforcement surfaces?

Alcyone: Cyber-Resilient FSW

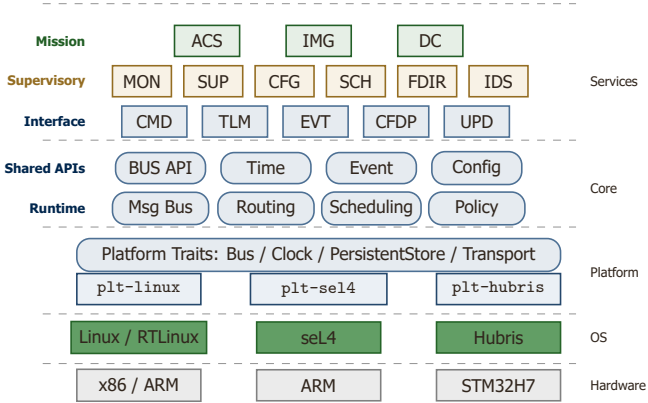
- Resilience as a first-class architectural property, not a bolt-on
- Every enforcement point traces to a specific threat and requirement
- Written in Rust — memory safety, data race freedom, and resource cleanup enforced at compile time
- Isolated services communicating over a zero-trust message bus
- Primary platform is seL4; initially developed on Linux, with hardware experiments using Rust embedded frameworks

Alcyone is the brightest star in the Pleiades cluster. In Greek mythology, Alcyone was a daughter of Atlas who was transformed into a kingfisher — the origin of "halcyon," meaning calm and protected.

From Requirements Paper to Implementation

	Initial paper scope	Final implementation scope
System scope	Four C&DH components	Full Alcyone service architecture and EO mission case study
Requirements	63 main C&DH requirements	89 system and subsystem requirement records
Artifacts	Threat-derived "shall" statements	38 architecture specs, 42 test cases, 68 code modules, and 80 observables
Traceability	Threat technique → requirement	Stakeholder → concern → requirement → spec → code → test

Architecture Overview



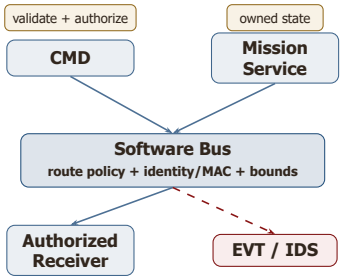
Rust and Platform Enforcement

- Removes many exploit preconditions at compile time, not at runtime
- `#![no_std]` means Rust code builds without the standard library
- `#![deny(unsafe_code)]` on all mission-logic crates; unsafe confined to platform FFI
- seL4 deployment maps services to isolated protection domains; platform traits keep mission logic OS-independent

Exploit Primitive	C/C++ Precondition	Rust Defense
Arbitrary write	Buffer overflow	Bounds-checked indexing
Arbitrary read	Out-of-bounds	Bounds-checked slices
Control-flow hijack	Corrupted fn ptr	No raw function pointers
Use-after-free	Dangling pointer	Ownership + borrow checker
Data race	Unsync. access	Send/Sync trait system
Type confusion	Unchecked cast	No implicit casts
Heap exhaustion	Dynamic allocation	No mission heap + `heapless`

Secure Blocks and Zero-Trust Bus

- Each service has explicit ingress, egress, authority, and fault behavior
- Routine traffic crosses the Software Bus, not direct task calls
- The bus checks sender, topic, destination, identity, and bounds
- Fixed-size messages and bounded queues make resource behavior measurable
- Rejects, drops, and quarantines become evidence for EVT, IDS, and TLM



Secure blocks make enforcement local; the bus makes cross-block trust explicit.

Authority and Supervisory Control

Authority is assigned, not implied

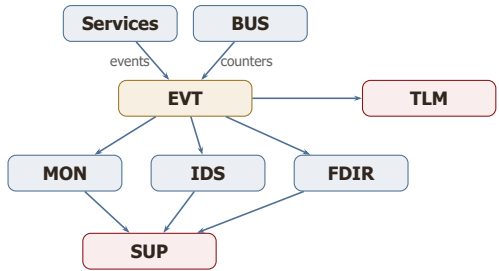
- CMD is command ingress and command authorization
- CFG owns configuration and table-update authority
- SUP owns mode, recovery, and prevention authority
- EVT/TLM preserve evidence and downlink visibility
- Mission services request or report; they do not own global control

Control path

- MON reports health and liveness
- IDS reports adversarial patterns
- FDIR reports fault classifications and recovery candidates
- SUP fuses evidence into bounded actions
- BUS enforces who may publish control messages

Observability and Evidence Flow

- Drops, rejects, denials, health changes, and recovery actions are observable artifacts
- EVT collects service events and bus counters
- MON, IDS, and FDIR consume evidence for health, threat, and fault assessment
- SUP decisions and TLM downlink make the architecture measurable



The same evidence path supports onboard response and experiment measurement.

Formal Methods in Rust

Safe Rust enforces memory safety, data-race freedom, and resource cleanup. Kani and Verus provide evidence for residual design properties.

Kani — bounded model checking

- Exhaustive over bounded inputs
- 131 harnesses, all passing

Verus — SMT-based proof

- Unbounded functional correctness
- Implementation vs. Assume-Guarantee Contracts

Properties checked:

- Authorization soundness
- Mode transition invariants
- Authority monotonicity
- Serialization round-trips
- Capacity bounds

RQ3 Takeaway

Alcyone embeds resilience through secure service blocks, explicit authority boundaries, bus-enforced communication, and supervised recovery control.

Those architectural mechanisms produce observable evidence and should yield measurably more resilient outcomes under adversarial conditions.

RQ4: Validating and Measuring Cyber Resilience

How can cyber resilience be demonstrated and measured to evaluate protection effectiveness and mission impact?

Taurus SWIL Testbed

- SWIL testbed based on NASA's NOS3 for repeatable mission scenarios and adversarial runs
- Closed-loop device simulation and synchronized mission time
- Orchestrates scenarios, attack injection, and data collection
- Orbital dynamics — NASA 42 engine for attitude and orbit state
- Containerized — reproducible end-to-end with a single CLI workflow
- Observability — records telemetry, events, detections, and metrics

Earth-Observation Case Study

- Shared workload for all experiments: a LEO Earth-observation imaging mission
- Mission sequence: plan upload, target slew, image collection, onboard storage, contact window, and downlink
- Representative functions: command processing, attitude control, payload imaging, data collection, storage, telemetry, and CFDP export
- The case study exercises the resilience mechanisms that matter: command authority, sensor trust, bus enforcement, recovery, and data return

Adversarial Testing

- Repeatable scenarios map attacks to SPARTA techniques, requirements, enforcement points, and measured outcomes
- Attack types include:
  - Command integrity — unauthorized, replayed, reordered, malformed
  - Denial / flooding — high-rate streams, queue saturation
  - Sensor manipulation — spoofed or withheld plant data
  - Telemetry corruption — injected or modified values
  - Recovery interference — disrupting fault management and modes

Experiment 1 Overview

- **Method:** controlled paired comparison; same mission, attack timing, and metrics, with architecture as the variable
- **Attacks:** command flood, unauthorized/replayed command, sensor drop, app kill, telemetry manipulation, internal bus spoofing, recovery interference, compound and adversary-chain cases
- **Evaluation criterion:** Alcyone produces earlier evidence, smaller blast radius, explicit rejection, and equal or better mission functionality under the same attacks as cFS

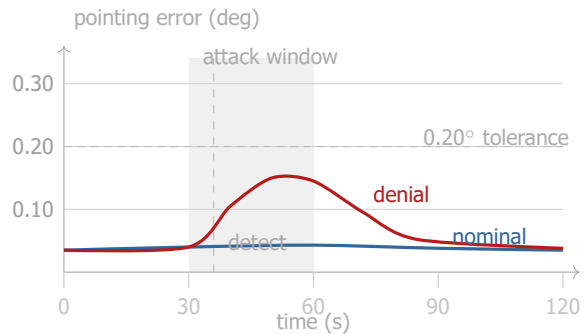
Experiment 1 Results

Attack	Metric	Alcyone	cFS
Command flood	Result	MTTD = 1.8 s	no detection
	R	0.94	0.72
	BR	1	4
Sensor drop	Result	MTTD = 0.087 s	no detection
	R	0.98	0.89
	BR	1	2
Unauthorized command	Result	0 accepted; 0.163 s	3 accepted; no alert
	R	1.00	0.96
	BR	0	1
Replay / reordered	Result	0 accepted; 0.220 s	8 accepted; 6.8 s
	R	1.00	0.91
	BR	0	2
Internal bus spoof	Result	MTTD = 0.47 s	no detection
	R	0.97	0.74
	BR	1	4

Experiment 2 Overview

- **Method:** Quantitative Measurement of Cyber Resilience (QMOCR), adapted from Army Research Lab work; compare nominal and attacked Alcyone pointing-error traces over time
- **Attack:** star tracker denial
- **Computation:** $F(t)$ maps pointing error to functionality;
  $R = \int_0^T F_{attacked}(t)\,dt / \int_0^T F_{nominal}(t)\,dt$
- **Evaluation criterion:** star tracker denial perturbs pointing error, but pointing stays within the 0.20° tolerance and recovers before mission loss

Experiment 2 Results



Resilience ratios by level

Component $R_{comp} = 0.936$
local ACS sensor channel

Subsystem $R_{sub} = 0.980$
ADCS behavior after containment

Mission $R_{miss} = 0.997$
EO pointing objective

Experiment 3 Overview

- **Method:** dose-response sweep; increase attack intensity while keeping the mission and detector definitions fixed
- **Evaluation criterion:** transition regions and reliable thresholds are identified, not just detected/missed labels

Attack variant	Sweep variable
Command flood	Packet rate: 10, 25, 50, 100, 250, 500, 1000, 2000 pps
Unauthorized command	Burst size: 1, 2, 5, 10, 20 commands
Sensor drop	Stale interval: 1, 5, 10, 15, 30, 60 s
Internal bus spoofing	Topic deviation: 1, 5, 10, 25, 50, 100%

Experiment 3 Results

Transition = detection begins; reliable = detection in nearly all trials.

- **Unauthorized command burst:** system-detected at 5 commands
  - Shows authority checks firing before mission impact
- **Sensor drop:** transition 10 s, reliable 15 s
  - Sets the stale-data timeout for sensor trust
- **Internal bus spoofing:** transition 25%, reliable 50%
  - Makes topic deviations an internal-abuse signal

Experiment 4 Overview

- **Method:** phase-timing study; inject the same attack during different mission phases
- **Attack:** unauthorized command burst
- **Evaluation criterion:** the unauthorized command is rejected in each phase; any remaining mission effect is explained by timing

Phase variant	Operational question
Imaging	Target pointing / image capture
Coast	Attitude and stored-data state
Downlink	Data-return timing and continuity

Experiment 4 Results

- Same unauthorized-command burst is exercised in each mission phase
- Imaging phase: 3/3 detections; MTTD = 0.16–0.18 s; R $= 0.992$
- Coast phase: 3/3 detections; MTTD = 0.16–0.18 s; R $= 1.000$
- Downlink phase: 3/3 detections; MTTD = 0.16–0.18 s; R $= 0.996$

Experiment 5 Overview

- **Method:** IDS configuration plus detector contribution study: rerun with detector families isolated or disabled
- **Evaluation criterion:** high detection coverage with acceptable false positives; contribution results show whether the full detector mix is needed

Detector family	Evidence source
Flood / rate	command rate, queue pressure, burstiness
Authority	invalid sender, command, mode, or role
Sensor trust	stale, missing, or out-of-range device data
Telemetry integrity	malformed/corrupted values and continuity anomalies

Experiment 5 Results

Metrics

P_d: detection probability
FPR: false positives per hour
MTTD: median time to detect
Window: detector lookback interval

- **History window 10 s:** $P_d = 0.99–1.00$, FPR = 0.050/hr, MTTD = 3.8 s
  - Stable evidence without slow alerts
- **Rate threshold 5:** $P_d = 0.950$, FPR = 0.050/hr, MTTD = 2.5 s
  - Balances sensitivity and alert load
- **Authority threshold 3:** $P_d = 1.000$, FPR = 0.040/hr, MTTD = 0.160 s
  - Catches command abuse quickly
- **Contribution:** full = 0.98, IDS-off = 0.00, minus-target = 0.20–0.30, only-target = 0.95–1.00
  - Detector mix preserves coverage

RQ4 Takeaway

Across 13 paired Experiment 1 scenarios, Alcyone produced detection evidence in 13/13 cases versus 3/13 for cFS, with mean R 0.94 vs. 0.72 and mean blast radius 1.3 vs. 3.0 mission functions.

The follow-on experiments bound the claim. Detection envelopes have measurable floors; phase-invariant authority detection still produces phase-dependent mission consequence; and no single detector family covers the attack space.

Discussion

Research Questions Answered

- **RQ1**: Attack-surface analysis is critical for flight software when tailored to flight-software structure, mission context, and spacecraft threat patterns
- **RQ2**: Systematic methods can produce formal cybersecurity requirements: specific, verifiable, and traceable to threats and engineering artifacts
- **RQ3**: An architecture can be built to support those requirements structurally through isolation, authority boundaries, bus enforcement, and observability
- **RQ4**: Evaluation validates the approach within the exercised scenarios by measuring cyber resilience qualities and mission impact

Limitations

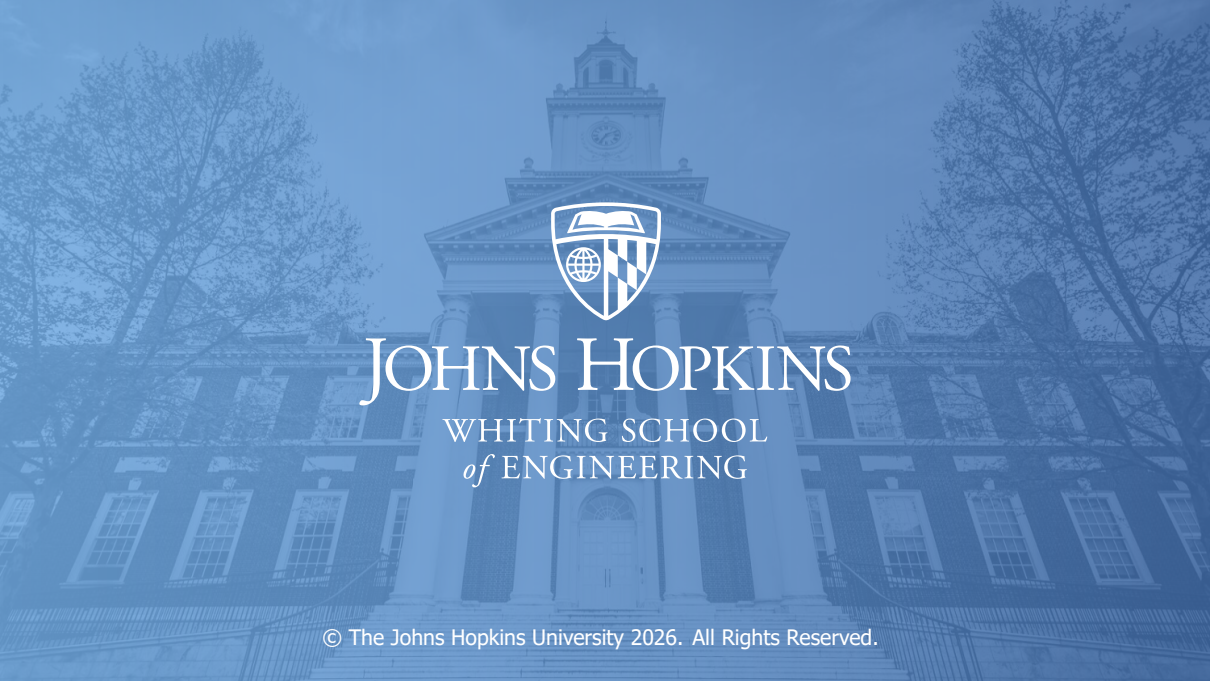
- **SWIL vs. flight hardware** — timing and fault behavior may differ on hardware; results characterize architectural influence, not absolute flight performance
- **Mission scope** — single LEO Earth-observation profile; generalization to other mission classes not yet established
- **Threat scope** — ground-segment compromise, supply-chain threats, and link-layer cryptanalysis are boundary conditions, not subjects of evaluation
- **Empirical boundary** — results are strongest for exercised Taurus scenarios, not untested attack classes
- **Single architecture** — comparison is cFS vs. Alcyone; broader architectural diversity (e.g., F Prime, proprietary FSW) would strengthen claims

Future Work

- **Hardware validation** — port to embedded flight hardware; characterize timing and watchdog behavior under resource constraints
- **Autonomous cyber defense** — lightweight anomaly detection and response policy under flight-processor constraints
- **Post-quantum cryptography** — cryptographic agility across decades-long mission timelines
- **Multi-spacecraft resilience** — extend secure-by-component reasoning to constellations and cross-vehicle trust
- **Formal methods expansion** — increase use of model checking and proof for bus policy, recovery invariants, and supervisory authority

Thesis Restated

Cyber resilience can be engineered as a first-class property of flight software.



© The Johns Hopkins University 2026. All Rights Reserved.