Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

When Predicate-Negation Controls Are Vacuous: A Reproducible PX4 Autopilot Case Study of an Added Command-Rejection Guard

Satish Kumar Mishra

satishmsr37@gmail.com

Netaji Subhas University of Technology (NSUT)

Manisha Khulbe

Netaji Subhas University of Technology (NSUT)

Research Article

Keywords: Flight control software, Unmanned aerial vehicle, Systems integration, PX4 autopilot, Regression testing, Test oracle

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11703294/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

When Predicate-Negation Controls Are Vacuous: A Reproducible PX4 Autopilot Case Study of an Added Command-Rejection Guard

Satish Kumar Mishra[1*] · **Manisha Khulbe**[1]

[1]Department of Electronics and Communication Engineering, Netaji Subhas University of Technology (NSUT), New Delhi 110078, India

[*]**Corresponding author(s).** E-mail(s): satishmsr37@gmail.com; **Contributing authors:** manisha.khulbe@nsut.ac.in

Abstract

In modular flight and robotic software, ground-control links often route dozens of telecommand types through one shared ingestion routine before dispatching each command to its subsystem. When a maintenance patch inserts a command-agnostic early-return validation check at that entry point, testing only inputs meant to trigger or negate the check can miss an interface-level integration failure: valid status and configuration requests get rejected alongside malformed navigation commands. Under a single-insertion handler model, any test built by negating the added guard condition executes the exact same instructions before and after the patch and is therefore vacuous for checking guard scope. GuardScope addresses this gap through a conservative static walk of the production C++ diff. Given an explicit field-to-payload map, it isolates dispatch branches that never read the guarded payload as exterior invariance controls and abstains whenever the full command structure is passed to external helpers. Evaluated on a historical patch in unmodified PX4 autopilot software-in-the-loop simulation, GuardScope identifies six exterior controls. All six keep acknowledgement code 0 under the global integer frame, whereas all 12 requests across two non-global frames shift from 0 to 9 across repeat and clean-clone runs. A 169-command differential sweep and eight-year PX4 and ArduPilot mining bound both baseline recall and idiom prevalence.

Keywords Flight control software · Unmanned aerial vehicle · Systems integration · PX4 autopilot · Regression testing · Test oracle

1 Introduction

Autonomous aircraft and spacecraft software stacks separate packet ingestion from subsystem execution: a single front-end command receiver parses incoming telecommands and forwards them to navigation, mission, storage, and health-reporting modules. In unmanned aerial vehicle (UAV) flight stacks built around MAVLink, ground control stations send both coordinate-bearing flight commands and non-positional status queries through this shared link [1–3]. Empirical defect studies of PX4 and ArduPilot show that missing bounds and faulty parameter checks account for a large share of field autopilot bugs [2, 3]. Maintainers therefore patch command handlers regularly to harden input validation. Standard regression testing checks the intended block first: an invalid packet that used to pass should be rejected after the patch [4, 5]. From a systems-integration standpoint, however, an unguarded check placed at a shared entry funnel risks breaking the ground-to-vehicle interface for unrelated subsystems. Blocking valid capability or home-position requests on an active link leaves operator consoles blind to vehicle state. The verification question during patch review is not merely whether pre-patch and post-patch binaries differ, but which command interfaces *ought to preserve identical behaviour* [6].

This paper pairs a simple structural observation with a reproducible software-engineering case study on the PX4 autopilot [1]. Suppose a commit inserts `if (p) { reject; return; }` at the top of a shared handler while leaving the rest of the routine unchanged. Every packet satisfying $\neg p$ then

traverses the exact same instructions in both revisions. Tests satisfying $\neg p$ still check that the downstream body was not broken by mistake, yet by construction they cannot reveal whether p sweeps in too many command types. Catching scope overreach requires holding p true while varying the command identifier (ID) across dispatch branches that never read the payload field constrained by p. Applied to a historical PX4 MAVLink patch, our conservative static branch analyser (GuardScope) isolates six such frame-independent commands for an unguarded coordinate-frame check. Every one of those six commands exposes unintended rejection on the public command acknowledgement (ACK) interface `COMMAND_ACK`.

Because the PX4 commit pair examined here was selected after its revert was already recorded in version history, we frame the evaluation strictly as a bounded post-hoc case study rather than a predictive benchmark. The paper makes four contributions: (i) a formal handler model showing why predicate-negation tests are structurally vacuous for newly added, command-agnostic early-return guards under a single-insertion edit; (ii) GuardScope, a conservative C++ dispatch walk driven by an explicit field map with abstention rules for whole-structure forwarding; (iii) deterministic paired software-in-the-loop (SITL) replay on unmodified PX4 binaries (commits `48525073` and `8a8496d5`), verified across repeat batches and clean repository clones; and (iv) a 169-command process-isolated differential baseline, a protocol-schema comparison against MAVLink `common.xml`, and an eight-year commit screen across PX4 and ArduPilot.

2 Related work

Regression selection and equivalence proofs. Classic regression test selection narrows an existing suite down to tests that reach modified statements [4, 5]. Regression verification tackles the problem formally by attempting to prove input-output equivalence between two closely related program revisions [7]. Related work on industrial programmable logic controllers combines equivalence checking with engineer-supplied invariants and delta specifications [8]. Proving full functional equivalence across multithreaded C++ flight firmware is rarely practical during day-to-day patch review. GuardScope instead derives a compact, auditable set of branch-level invariance controls and executes them directly against unmodified SITL binaries.

Differential testing and test oracles. Differential testing [9], behavioural regression tools [10], differential symbolic execution [11], and shadow symbolic execution [12] search for inputs on which two software versions diverge. Recent language-model pipelines such as DiffTestGen supply call graphs, docstrings, and coverage traces to a model to generate version-splitting tests [13]. Finding an input where two revisions disagree solves only half of the review problem; an engineer still needs a test oracle to separate intended new rejections from unintended collateral regressions [6]. Just as metamorphic testing [14] checks invariance relations across linked inputs, GuardScope targets oracle labelling by identifying command branches whose public ACK code has no source-level reason to change when the guard fires.

Autopilot defect studies and fuzzing. Repository inspections of PX4 and ArduPilot confirm that parameter-validation flaws recur throughout unmanned flight-control development [2]. Runtime fuzzers such as RVFuzzer [3] and PGFuzz [15] mutate configuration parameters and monitor temporal safety formulas during simulated flight to uncover validation bugs. Meanwhile, the ROBUST study of Robot Operating System packages documents the practical engineering effort needed to reproduce historical robotics defects reliably [16]. Closed-loop fuzzers explore wide state spaces over multi-hour campaigns; our work addresses fast, deterministic command-interface verification during code review.

3 System and problem formulation

3.1 Paired handler model

Let P^- and P^+ denote the pre-patch and post-patch autopilot binaries, let $x = (c, q)$ denote an incoming telecommand packet with command ID c and payload fields q, and let $y(P, x)$ denote the public ACK code emitted by binary P. Consider a patch in which P^+ differs from P^- solely by placing an early rejection guard ahead of an unchanged downstream handler body H:

$$P^+(x) = r(x) \text{ if } p(x) \text{ holds}; \quad P^+(x) = H(x) \text{ if } \neg p(x) \text{ holds}; \quad \text{and} \quad P^-(x) = H(x). \tag{1}$$

Suppose evaluating $\neg p(x)$ introduces no side effects. By Eq. (1), $P^+(x) = P^-(x)$ for every packet satisfying $\neg p(x)$. Any regression test constructed by negating p verifies that $H(x)$ still executes as before, yet it cannot detect an over-broad condition inside p. Scope overreach becomes observable only when $p(x)$ holds on a command c whose dispatch branch in H never reads the payload field restricted by p.

Our empirical study examines the PX4 MAVLink command receiver [1]. Between parent commit `48525073aa6f6e96e06e3740d14247f3201cae97` and merge commit `8a8496d57e0d5140d9210c255d18435cac1de57a`, a patch inserted the following check at the entry of `MavlinkReceiver::handle_message_command_int`:

```
if (cmd_mavlink.frame != MAV_FRAME_GLOBAL_INT) { acknowledge(...);
return; }
```

Nothing in that `if` condition inspects `cmd_mavlink.command`, even though the routine translates the packet into a `vehicle_command` structure and forwards it to a shared multi-command dispatcher. Throughout this paper, the observed output y is the public MAVLink `COMMAND_ACK` result code; all empirical claims remain bounded to this command-acknowledgement interface rather than closed-loop flight dynamics.

3.2 Conservative exterior-control construction

GuardScope takes only the pre-patch and post-patch C++ source files together with a declared field map Q(`frame`) = {x, y, z}. Commit messages, pull-request discussions, and existing test suites are never read. The analyser traces how fields in Q copy into `vehicle_command.param5` through `param7` and then inspects top-level `command == MAV_CMD_*` branches inside the target dispatcher (Fig. 1). A branch is emitted as an exterior control only if it neither reads a tracked coordinate slot nor forwards the whole command structure to an external helper. Whole-structure forwarding is marked unresolved because a callee could read coordinate slots downstream, and nested command checks inside generic forwarding blocks abstain for the same reason.

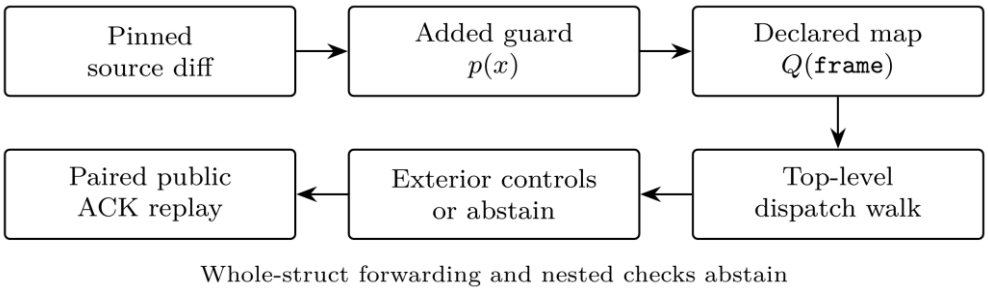


Fig. 1 Pipeline of the GuardScope static walk; the field map Q is supplied explicitly and branches with unresolved struct forwarding abstain (created in TikZ)

3.3 Algorithm and abstention rules

Six deterministic rules govern when GuardScope emits exterior controls and when it abstains:

1. Locate newly added early-return blocks matching if (predicate) { ... acknowledge(...) ... return; }. Unless the diff introduces a single block of this shape, abstain.

2. Reject any predicate that already inspects the command selector (.command), since such a guard already bounds its own command scope.

3. Extract packet fields referenced inside the predicate. Exactly one field must match a declared key in Q; otherwise abstain so domain assumptions remain explicit.

4. Identify the enclosing ingestion routine and resolve its dispatcher call within the same translation unit. If the target routine cannot be resolved or contains fewer than two command branches, abstain.

5. Propagate local assignments from fields in Q into the translated internal command structure (for example, x assigned to param5).

6. Classify each depth-one command == MAV_CMD_* branch: reading a tracked field marks it payload-consuming; passing the whole structure to a callee marks it unresolved; doing neither marks it an exterior control.

Rules 5 and 6 keep precision high by refusing to guess across helper boundaries. In the PX4 dispatcher studied here, three branches forward the complete `vehicle_command` structure to external routines and one check sits nested inside a generic forwarding block; all four are withheld from the control set. Each emitted exterior control carries a concrete invariance oracle: under Q and the local branch body, adding the coordinate-frame guard should leave the public `COMMAND_ACK` code unchanged when both binaries execute from an identical simulator state.

4 Experimental protocol

4.1 Paired SITL replay

Applying GuardScope to the PX4 commit pair under $Q(\texttt{frame}) = \{x, y, z\}$ partitions the 11-branch dispatcher into six exterior controls, two payload-consuming branches, three unresolved whole-structure forwards, and one nested check. Stock `px4_sitl_default` binaries were compiled at both commits and booted under PX4's built-in simulation-in-hardware (SIH) quadrotor airframe. A compact Python harness sent identical `COMMAND_INT` packets over the User Datagram Protocol (UDP) to each binary, matched incoming `COMMAND_ACK` messages by command ID, retried every 1 s up to a 6 s ceiling if needed, and terminated the simulator cleanly after each batch.

Every exterior control was replayed across three MAVLink coordinate frames with all remaining fields held fixed: `GLOBAL_INT` (the $\neg p$ baseline), `GLOBAL`, and `LOCAL_NED` (both satisfying p). Two audit commands were recorded alongside them without counting toward the six controls: `DO_REPOSITION` as a guard-activating positional check and `LOGGING_START` for the excluded nested branch (Table 1). Each paired trial was labelled *preserved* when pre-patch and post-patch ACK codes matched, *drift* when both replies arrived with different codes, or *unobserved* if either binary timed out. The 24-row batch was executed twice per revision to confirm run-to-run determinism.

Table 1 MAVLink command requests executed in paired SITL replay and observed COMMAND_ACK transitions (pre-patch → post-patch; identical across initial, repeat, and clean-rebuild runs)

MAVLink Command (MAV_CMD_*)	Audit Role	Fixed Payload[a]	GLOBAL_INT	GLOBAL / LOCAL_NED
REQUEST_AUTOPILOT_CAPABILITIES	Derived control	param1 = 1	0 → 0	0 → 9
REQUEST_PROTOCOL_VERSION	Derived control	param1 = 1	0 → 0	0 → 9
GET_HOME_POSITION	Derived control	All zeros	0 → 0	0 → 9
REQUEST_FLIGHT_INFORMATION	Derived control	param1 = 1	0 → 0	0 → 9
REQUEST_STORAGE_INFORMATION	Derived control	param2 = 1	0 → 0	0 → 9
GET_MESSAGE_INTERVAL	Derived control	param1 = 1	0 → 0	0 → 9
LOGGING_START	Nested audit	All zeros (excl.)	0 → 0	0 → 9
DO_REPOSITION	Activation check	Positional (excl.)	0 → 0	0 → 9

[a]All unset payload slots, including coordinate fields x, y, z (param5–param7) on derived controls, are 0. ACK code 0 denotes MAV_CMD_ACK_ACCEPTED; code 9 denotes MAV_CMD_ACK_ERR_COORDINATE_FRAME_NOT_SUPPORTED.

4.2 Clean rebuild, differential baseline, and schema check

Clean rebuild. Both commits were checked out into fresh clones of upstream PX4, compiled from scratch (`px4_sitl_default`), and re-executed with the identical UDP harness to confirm that cached build state played no role.

Process-isolated 169-command sweep. A separate differential script transmitted one zero-payload `COMMAND_INT` packet for every non-deprecated `MAV_CMD` entry in MAVLink `common.xml` across the same three frames using a 2 s receive window. `PREFLIGHT_REBOOT_SHUTDOWN` and `DO_FLIGHTTERMINATION` were omitted so they could not halt the simulator mid-run, leaving 169 commands (507 command–frame rows per commit). A fresh SITL process was booted for every single row so state mutations from earlier commands could not contaminate later replies.

MAVLink XML schema check. To test whether protocol metadata alone could replace *Q*, `common.xml` parameter descriptions for `param5–param7` were scanned for positional labels (latitude, longitude, altitude, *x, y, z*) or units (`degE7`, `m`). Schema labels agreed with GuardScope on all six exterior controls. However, `SET_MESSAGE_INTERVAL` and `REQUEST_MESSAGE` repurpose `param5–param7` for non-positional configuration and response-target fields; a schema-only filter marks both commands non-positional even though their C++ branches read those slots directly. Across all 169 commands, `common.xml` classifies 117 as non-positional and 52 as positional. Interface schemas provide helpful priors, yet they do not replace source-level branch inspection.

4.3 Runtime timing and threats to validity

After SITL boot, the 24-row replay completed in 8.4 s on the pre-patch binary and 3.5 s on the post-patch binary, as early-return rejections avoid downstream dispatch. Strict command-ID matching filtered out asynchronous telemetry traffic. Internal validity is bounded by whole-binary execution:

replaying a merge commit cannot by itself exclude unrelated edits outside the inspected routine, although deterministic pairing and clean rebuilds rule out simulator nondeterminism. Construct validity rests on the explicit map $Q(\texttt{frame}) = \{x, y, z\}$ and single-hop copy tracking, with whole-structure abstentions preventing silent alias misses. External validity is restricted to public `COMMAND_ACK` responses in SITL rather than physical flight trials.

5 Results and baseline comparison

Figure 2 and Table 2 summarise the paired SITL replay outcomes. Pre-patch PX4 returned `COMMAND_ACK` code 0 (accepted) on all 24 command–frame requests. Post-patch PX4 preserved code 0 on every tested command under `GLOBAL_INT` ($\neg p$), whereas all 12 derived exterior-control requests under `GLOBAL` and `LOCAL_NED` (p) shifted from 0 to code 9 (unsupported coordinate frame). Repeat batches and fresh-clone builds reproduced all 24 transitions without a single dropped reply (Table 1 and Fig. 2).

Command / audit role	GLOBAL_INT	GLOBAL	LOCAL_NED
REQUEST_AUTOPILOT_CAPABILITIES (derived)	0→0	0→9	0→9
REQUEST_PROTOCOL_VERSION (derived)	0→0	0→9	0→9
GET_HOME_POSITION (derived)	0→0	0→9	0→9
REQUEST_FLIGHT_INFORMATION (derived)	0→0	0→9	0→9
REQUEST_STORAGE_INFORMATION (derived)	0→0	0→9	0→9
GET_MESSAGE_INTERVAL (derived)	0→0	0→9	0→9
LOGGING_START (nested; excluded)	0→0	0→9	0→9
DO_REPOSITION (activation; excluded)	0→0	0→9	0→9

Unshaded (0→0): $\neg p$ baseline; Dark-shaded (0→9): guard-activating drift; Below rule: excluded audit rows

Fig. 2 Paired acknowledgement grid across pre-patch and post-patch PX4 SITL binaries showing unshaded $\neg p$ baseline cells (0 → 0), dark-shaded guard-activating cells (0 → 9), and excluded audit rows below the horizontal rule (created in TikZ)

Table 2 Aggregate paired public-ACK replay outcomes across the 24-row test protocol

Command–Frame Group	n	Pre-Patch ACK	Post-Patch ACK	Paired Outcome
Derived controls × GLOBAL_INT	6	0	0	6 preserved
Derived controls × GLOBAL, LOCAL_NED	12	0	9	12 drift
Positional activation check (audit)	3	0	0/9	Descriptive
Nested unresolved command (audit)	3	0	0/9	Excluded
Total observed command–frame rows	24	0	0/9	0 missing

Table 3 contrasts GuardScope against the 169-command differential baseline. Post-patch PX4 returned code 9 on every non-global row (338 of 338) while leaving all 169 `GLOBAL_INT` rows unchanged. Raw divergence recall is identical: 143 commands exhibited observed ACK drift, while 52 non-global rows produced no pre-patch ACK within 2 s. However, an unlabelled differential sweep hands the reviewer 143 diverging commands without distinguishing intended positional

rejections from collateral interface breakage [6]. Filtering those 143 commands through `common.xml` still leaves 94 drifting non-positional commands across separate modules, including commands that overload coordinate slots for non-positional settings. GuardScope narrows that output to six invariance controls tied directly to branches in the edited dispatcher. Running the same 507-row sweep inside a single persistent SITL instance without per-row restarts altered 13 of the 1,014 observations (11 pre-patch and 2 post-patch) due to cross-command state carryover, confirming the need for process isolation.

Table 3 Process-isolated differential baseline across 169 MAV_CMD commands (507 command–frame rows per revision)

Differential Baseline Metric	Observed Count
GLOBAL_INT rows differing post-patch (of 169)	0
Non-global rows returning ACK code 9 post-patch (of 338)	338
Non-global rows with observed pre-patch ACK / differing post-patch	286 / 286
Non-global rows with no pre-patch ACK within 2 s window	52
Distinct commands exhibiting ≥ 1 observed ACK drift	143
Non-positional commands per common.xml (drifting / pre-patch observed rows)	117 (188 / 188)
Non-positional commands per common.xml exhibiting ≥ 1 observed ACK drift	94
Positional commands per common.xml (drifting / pre-patch observed rows)	52 (98 / 98)
Source-derived exterior controls isolated by GuardScope (11-branch dispatch)	6

6 Causal and mechanistic analysis

In the studied PX4 patch, $p(x)$ = [`frame != GLOBAL_INT`], so $\neg p(x)$ restricts every test input to `GLOBAL_INT`. All six exterior controls in that column remain at 0 → 0 (Table 2), matching Eq. (1). A regression suite built only from $\neg p$ sanity checks and positive positional checks passes without error even while 12 frame-independent status and configuration requests fail. Exposing the scope defect requires keeping $p(x)$ true and varying the command selector across branches that never touch positional slots.

This vacuity property holds only under the single-insertion structure of Eq. (1). Whenever a patch simultaneously modifies H, mutates persistent vehicle state while evaluating p, or alters message decoding ahead of the guard, $\neg p$ tests are no longer vacuous and remain essential to exercise H. Table 4 details the static breakdown of all 11 branches in the PX4 dispatcher; withholding the three whole-structure forwards and one nested check prevents false regression alarms when helper routines consume coordinates.

Table 4 Static classification of the 11-branch PX4 command dispatcher

Branch Classification	Count	Technical Basis
Exterior control	6	No qualified-field read and no whole-object forwarding
Payload-consuming	2	Branch directly reads a frame-qualified parameter
Unresolved	3	Whole command structure forwarded to external callee
Nested check	1	Nested inside generic forwarding block rather than depth-one

7 Historical prevalence across autopilot stacks

How frequently do open-source flight stacks introduce command-agnostic early-return guards of this form? Four repository screens across PX4 and ArduPilot quantify its prevalence (Table 5). The first screen paired the study patch with a later 2026 PX4 parameter-validation commit that also rejected valid ground-station commands such as takeoff; because that commit used a command-indexed lookup table rather than an unguarded early return, GuardScope abstained by rule. A second screen covering 47 guard-related commits (4 in PX4 and 43 in ArduPilot) produced no additional candidates, reflecting ArduPilot's decomposition of MAVLink handling into per-message routines rather than one shared command funnel. A third slice of 12 earlier PX4 commits predating the motif likewise yielded zero matches.

Table 5 Historical repository screens evaluating prevalence of the command-agnostic rejection-guard idiom

Repository Screen	Outcome	Technical Interpretation
PX4 guard pair (n = 2)	1 candidate; 1 abstention	Emits study case; abstains on table-driven command-indexed check
Two-stack screen (n = 47)	1 candidate; 46 abstentions	43 ArduPilot updates use distinct dispatch structures and abstain
Earlier PX4 slice (n = 12)	0 candidates	Pre-motif commits contain no matching early-return guard addition
8-year PX4 mining (n = 26 pairs)	1 candidate; 25 abstentions	Manual triage confirms at most two instances across 2018–2026

The fourth screen ran a pre-registered, SHA-256-logged git search across eight years of PX4 history (2018 to September 2026) for diffs adding early returns paired with acknowledgement calls. Two core MAVLink command files produced 47 hits across 24 commits, and a broader `src/` handler filter yielded 18 hits across 14 commits. Running GuardScope unmodified across all 26 distinct commit–file pairs returned the study case and 25 abstentions. Manual inspection of every hit confirmed at most two genuine instances of this idiom in eight years, and 12 subsequent PX4 pull requests (July to October 2026) all abstained as well. Over-broad command rejection recurs in flight-software maintenance [2], yet this specific syntactic idiom is rare.

8 Limitations and reproducibility

Three scope boundaries govern our findings. First, the PX4 commit pair was chosen after its revert was already on record, so the evaluation represents a post-hoc case study rather than a blind prospective trial. Second, GuardScope relies on an explicit field map $Q(\texttt{frame}) = \{x, y, z\}$ and single-hop copy tracking rather than interprocedural pointer analysis; abstaining on whole-structure forwards keeps unresolved helper calls out of the emitted control set. Third, replay observes public `COMMAND_ACK` codes on single requests and does not evaluate closed-loop flight trajectories or fuzzer wall-clock budgets [3, 13, 15]. Every quantitative result in this article can be reproduced directly from upstream PX4 at parent commit `48525073aa6f6e96e06e3740d14247f3201cae97` and merge commit `8a8496d57e0d5140d9210c255d18435cac1de57a` using the stock `px4_sitl_default` SIH target, MAVLink `common.xml`, and the packet parameters in Table 1.

9 Conclusion

Shared telecommand ingestion routines create a subtle software-evolution hazard across flight and autonomous systems: an entry-level validation guard written for one payload class can silently sever unrelated ground-to-vehicle command interfaces. Under a single-insertion handler model, tests that negate the added guard condition execute unchanged code paths and are vacuous for detecting scope overreach. By holding the guard predicate true and walking depth-one dispatch branches under a declared field map, GuardScope isolated six frame-independent exterior controls in a historical PX4 MAVLink patch. Deterministic SITL replay confirmed 0 → 0 preservation under `GLOBAL_INT` and 0 → 9 collateral rejection across all 12 non-global control requests, pinpointing source-backed regressions within a 169-command differential sweep while eight-year repository mining established the rarity of the exact syntactic motif. Extending conservative exterior-control extraction to table-driven command validators and modular flight-software buses is a practical direction for future verification tooling.

Statements and Declarations

Funding The authors did not receive support from any organization for the submitted work.

Competing Interests The authors have no competing interests to declare that are relevant to the content of this article.

Author Contributions Satish Kumar Mishra formulated the paired handler model, developed GuardScope and the PX4 software-in-the-loop replay harness, conducted the experiments, and wrote the manuscript. Manisha Khulbe supervised the research methodology, reviewed the experimental protocol and validity bounds, and revised the manuscript. Both authors read and approved the final manuscript.

Data and Code Availability All experimental results reported in this article can be reproduced directly from the public upstream PX4-Autopilot repository (https://github.com/PX4/PX4-Autopilot) at parent commit 48525073aa6f6e96e06e3740d14247f3201cae97 and merge commit 8a8496d57e0d5140d9210c255d18435cac1de57a, together with the public MAVLink protocol specification (https://github.com/mavlink/mavlink). The GuardScope static analysis script, deterministic UDP replay scripts, and raw CSV logs generated during the current study are available from the corresponding author on reasonable request.

References

1. Meier L, Honegger D, Pollefeys M (2015) PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, Seattle, pp 6235–6240. https://doi.org/10.1109/ICRA.2015.7140074

2. Wang D, Li S, Xiao G, Liu Y, Sui Y (2021) An exploratory study of autopilot software bugs in unmanned aerial vehicles. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM, New York, pp 20–31. https://doi.org/10.1145/3468264.3468559

3. Kim T, Kim CH, Rhee J, Fei F, Tu Z, Walkup G, Zhang X, Deng X, Xu D (2019) RVFuzzer: Finding input validation bugs in robotic vehicles through control-guided testing. In: Proceedings of the 28th USENIX Security Symposium. USENIX Association, Santa Clara, pp 425–442

4. Rothermel G, Harrold MJ (1997) A safe, efficient regression test selection technique. ACM Trans Softw Eng Methodol 6(2):173–210. https://doi.org/10.1145/248233.248262

5. Yoo S, Harman M (2012) Regression testing minimization, selection and prioritization: A survey. Softw Test Verif Reliab 22(2):67–120. https://doi.org/10.1002/stvr.430

6. Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S (2015) The oracle problem in software testing: A survey. IEEE Trans Softw Eng 41(5):507–525. https://doi.org/10.1109/TSE.2014.2372785

7. Godlin B, Strichman O (2009) Regression verification. In: Proceedings of the 46th Design Automation Conference (DAC). ACM, New York, pp 466–471. https://doi.org/10.1145/1629911.1630034

8. Ulewicz S, Ulbrich M, Weigl A, Kirsten M, Wiebe F, Beckert B, Vogel-Heuser B (2016) A verification-supported evolution approach to assist software application engineers in industrial factory automation. In: 2016 IEEE International Symposium on Assembly and Manufacturing (ISAM). IEEE, Fort Worth, pp 19–25. https://doi.org/10.1109/ISAM.2016.7750714

9. Evans RB, Savoia A (2007) Differential testing: A new approach to change detection. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM, New York, pp 549–552. https://doi.org/10.1145/1295014.1295049

10. Jin W, Orso A, Xie T (2010) Automated behavioral regression testing. In: Third International Conference on Software Testing, Verification and Validation (ICST). IEEE, Paris, pp 137–146. https://doi.org/10.1109/ICST.2010.31

11. Person S, Dwyer MB, Elbaum SG, Păsăreanu CS (2008) Differential symbolic execution. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). ACM, New York, pp 226–237. https://doi.org/10.1145/1453101.1453131

12. Palikareva H, Kuchta T, Cadar C (2016) Shadow of a doubt: Testing for divergences between software versions. In: Proceedings of the 38th International Conference on Software Engineering (ICSE). ACM, New York, pp 1181–1192. https://doi.org/10.1145/2884781.2884845

13. Hu H, Cadar C, Pradel M (2026) DiffTestGen: Change-directed LLM-based testing for exposing behavioral differences. arXiv preprint arXiv:2607.16024. https://arxiv.org/abs/2607.16024. Accessed 10 Nov 2026

14. Chen TY, Kuo FC, Liu H, Poon PL, Towey D, Tse TH, Zhou ZQ (2018) Metamorphic testing: A review of challenges and opportunities. ACM Comput Surv 51(1):4:1–4:27. https://doi.org/10.1145/3143561

15. Kim H, Ozmen MO, Bianchi A, Celik ZB, Xu D (2021) PGFuzz: Policy-guided fuzzing for robotic vehicles. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). Internet Society, Reston. https://doi.org/10.14722/ndss.2021.24096

16. Timperley CS, van der Hoorn G, Santos A, Deshpande H, Wąsowski A (2024) ROBUST: 221 bugs in the robot operating system. Empir Softw Eng 29(3):57. https://doi.org/10.1007/s10664-024-10440-0