Spec-Driven Multi-Agent Corpus Generation for Over-the-Air Wi-Fi Fuzzing

Jean D'Elboux Diogo, Nilton Moura, Thatyanne Prado, Diego Cardoso Borda, Cláudia Werner

[1]Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brazil

{jdelboux,nmoura,prado,diegocbcastro,werner}@cos.ufrj.br

Abstract. *Over-the-air (OTA) Wi-Fi fuzzers such as owfuzz test real IEEE 802.11 devices, but their reach is limited by a hand-written, static corpus that covers only a fraction of the 802.11 frame and information-element (IE) space. We present ongoing work on an autonomous* multi-agent *system that reads the protocol specification and builds the fuzzing corpus automatically. A Spec-Reader, a Synthesizer, and an LLM-reasoned Feedback/Triage agent cooperate through artifact contracts and drive owfuzz, an established OTA fuzzer, against real hardware. In a preliminary OTA study, correcting a mutation-fidelity defect let the corpus populate all 15 targeted IEs and deliver its size and length mutations on the air, widening the injected frame length from at most 66 to 348 bytes across roughly 24,000 frames per campaign. The same corpus, unchanged, then drove a different device class and band (a connected 5 GHz client), indicating the agent layer is device- and band-independent. Against owfuzz's own hand-written corpus, the agent corpus trades raw frame volume for structural validity and stateful depth, completing a credential-gated four-way handshake the native corpus never reaches; a full coverage head-to-head is ongoing.*

1. Introduction

The owfuzz tool [Cao et al. 2023] fuzzes IEEE 802.11 management, control, and data frames over the air, and has found real vulnerabilities in commodity devices. The hand-written test corpus is the limiting factor, since it's fixed, and it exercises only a small part of the 802.11 frame and information-element (IE) space. Grammar- and model-based Wi-Fi fuzzers [Kampourakis et al. 2022, Garbelini et al. 2022] share the problem, because their grammars and protocol models are hand-built too. We began by trying to extend owfuzz, and soon saw that the hard part is building the corpus, which means turning the prose of the standard into frames that are structurally valid, adversarial, and broad enough to cover the specification.

Our answer is an autonomous multi-agent system that derives the corpus from the specification *text* and refines it using feedback from the real device, rather than depending on a hand-written grammar (the gap from prior work is detailed in Section 2). In the architecture, only the transmission backend is Wi-Fi-specific, so the design could in principle extend to a general engine that reads a protocol specification and produces an OTA corpus for it. We have not yet demonstrated that generality beyond Wi-Fi, so we present it as a goal and give a first Wi-Fi instantiation on owfuzz.

So far, our contributions are:

- An autonomous multi-agent architecture (Spec-Reader, Synthesizer, Feedback/Triage) that builds an 802.11 fuzzing corpus from specification text, with no hand-written grammar, and refines it from black-box, over-the-air feedback (Section 3).
- A spec-derived mutation-targeting strategy that augments the host fuzzer's existing mutation engine.
- A preliminary over-the-air evaluation on a commodity access point and a connected client that exercises the full pipeline, checks corpus fidelity, and gives a first head-to-head against owfuzz's own hand-written corpus (Section 4); a complete coverage comparison is in progress.

2. Background and Related Work

We position our system between two lines of prior work. The first fuzzes Wi-Fi over the air but with hand-built test material. The second uses LLMs to automate test generation but does not target over-the-air Wi-Fi.

OTA Wi-Fi fuzzing. Several tools fuzz IEEE 802.11 over the air. The owfuzz tool [Cao et al. 2023] injects management, control, and data frames from a hand-written corpus. Greyhound [Garbelini et al. 2022] runs a directed greybox fuzzer against a hand-built Wi-Fi protocol model. WPAxFuzz [Kampourakis et al. 2022] covers all 802.11 frame types using hand-authored grammars, and the commercial Defensics suite generates tests from a built-in model of the 802.11 specification. Defensics is closest in spirit, but its model is vendor-built and proprietary rather than derived from specification text, and the closed suite was unavailable for comparison; we therefore baseline against owfuzz, whose hand-written corpus is open and shares our OTA substrate. Schepers and Vanhoef [Schepers et al. 2021] provide a framework for testing and fuzzing Wi-Fi devices. In every case the grammar, model, or corpus is hand-written, and generation does not adapt to device behavior.

LLM-guided protocol fuzzing. A separate line of work uses LLMs to produce protocol test inputs. ChatAFL [Meng et al. 2024] extracts a machine-readable grammar and diversifies messages; MultiFuzz [Maklad et al. 2025] is a retrieval-augmented multi-agent system that reasons over RFC documents; LIPFuzzer [Zhong et al. 2025] feeds protocol documentation to an LLM to pick mutation fields for IoT protocols such as Modbus, MQTT, and CoAP; and other recent work applies LLMs to model-based protocol fuzzing [Huang et al. 2025]. These target socket- or emulation-based protocols (RTSP, DNS, MQTT, HTTP), lean on code coverage or in-process feedback, and none reach over-the-air Wi-Fi on real hardware.

Gap. Spec- and grammar-based OTA Wi-Fi fuzzing exists but runs on *hand-authored* grammars and models [Kampourakis et al. 2022, Garbelini et al. 2022], while the LLM-based fuzzers automate generation for socket and IoT protocols, leaving over-the-air Wi-Fi untouched. We are not aware of a prior system that does all three of: (i) build the Wi-Fi frame/IE corpus *automatically* from specification *text*, (ii) through collaborating agents, and (iii) *adapt* it from black-box, over-the-air feedback on real devices.

Organization. Section 3 describes the architecture; Section 4 reports the over-the-air evaluation and the comparison against owfuzz's corpus; Section 5 gives the research

agenda; and Section 6 concludes.

3. Approach

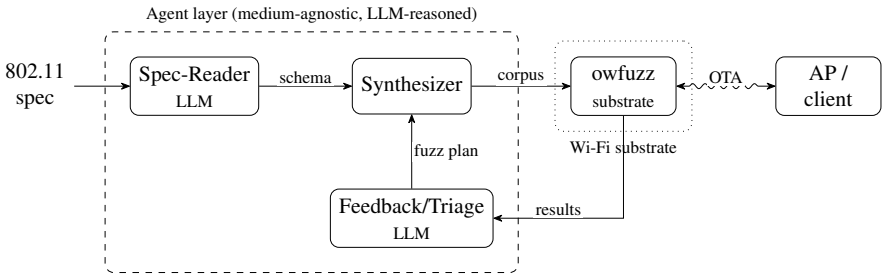

Figure 1. Decoupled multi-agent architecture. The medium-agnostic agent layer (Spec-Reader, Synthesizer, Feedback/Triage) communicates through file artifacts (schema, corpus, results, fuzz plan) and drives an interchangeable Wi-Fi substrate (owfuzz) over the air; the fuzz plan closes the feedback loop.

3.1. Design overview

Figure 1 shows the architecture. We separate two functions earlier OTA fuzzers combine: *generating* test inputs (a medium-agnostic *agent layer* that builds and adapts the corpus) and *transmitting* them (the replaceable *execution substrate*, owfuzz, which injects frames and watches the target). The agents communicate only through files (a frame/IE *schema*, a *corpus*, a machine-readable *results* log, and a *fuzz plan*), so each can be tested and replaced on its own. The Spec-Reader and Feedback/Triage agents are LLM-reasoned (schema-constrained), the Synthesizer is deterministic, and owfuzz is the OTA substrate.

Agentic framing. The system is an instance of *agentic AI* rather than a belief-desire-intention (BDI) multi-agent system, and we are explicit about where autonomy lives. The two LLM-driven roles, the Spec-Reader and the Feedback/Triage agent, are the autonomous agents. Each *perceives* an artifact (specification text; a results log), *reasons* over it with an LLM under a schema constraint, and *acts* by writing a new artifact, and no fixed script dictates how. The Feedback/Triage agent is the clearest case, reading a campaign's outcome and deciding how to spend the next round's budget, which changes a peer's behavior (it dropped the Probe Request family and reweighted the Synthesizer's operators, Section 4). The Synthesizer is *not* an autonomous agent but a deterministic transformation component the agents drive through the shared artifacts (Section 3.3). Coordination follows a *blackboard* model. Unlike MultiFuzz's retrieval agents over RFC text [Maklad et al. 2025], this loop is closed by a physical target over the air.

A good 802.11 corpus must be at once *structurally valid* (or the target silently drops it), *semantically adversarial*, and *spec-complete* across subtypes, the information-element (IE) catalog, and protocol states. Hand-written corpora trade breadth for validity; our agents get all three by building frames from the specification rather than from captured traffic, which shows only what the device already does.

3.2. Spec-Reader agent

The Spec-Reader reads specification text and produces, for each management frame, a structured schema comprising the frame-control subtype, the fixed-field bytes, and the ordered IEs. Each IE carries its element identifier, whether it is mandatory, optional,

or conditional, a baseline value, and the constraints that govern it, along with a short rationale. We constrain the model to a JSON schema the next stage expects and cache the specification text as a stable prompt prefix. A validation step then rejects malformed bytes and repairs values that wrongly embed element framing before the schema reaches the Synthesizer.

3.3. Synthesizer

The Synthesizer is a deterministic program (about 370 lines of Python, with no LLM call and no randomness) that compiles the schema into the corpus through a fixed library of template-based, spec-aware operators. For each frame it emits a structurally valid *baseline* from the schema's fields, then a family of *adversarial-but-parseable* variants built by those operators. Each one breaks a single structural rule while keeping the frame parseable, so the target still reaches the code under test. They enlarge a length field beyond the value present (a parser over-read), oversize a value, duplicate an element, drop a mandatory element, zero an element's length, inject a reserved element identifier, and, for frames carrying fixed pre-IE fields, corrupt those fixed fields (algorithm number, transaction sequence, status code, capability flags, listen interval). It emits frames directly in the host fuzzer's replay format (a 24-byte MAC header, fixed fields, length-prefixed elements), so the substrate needs no translation.

3.4. Feedback/Triage agent and the closed loop

After each campaign, the Feedback/Triage agent reads the results log and corpus labels and labels each frame's effect as dropped (a validity failure), accepted (state advanced), anomalous, or target-silent (a likely crash), then writes a *fuzz plan* that points the next round at the frames, elements, and operators that paid off. The orchestrator feeds the plan back to the Synthesizer, closing the loop. The agent works from *black-box, over-the-air* signals (response or silence, and target liveness on real hardware), since commodity Wi-Fi devices do not expose code coverage. The division of labor is deliberate: the agents decide *what to target* (frames, elements, states), while the host fuzzer's mutation engine does the low-level byte mutation, augmenting a proven fuzzer.

3.5. Integration with the owfuzz substrate

We extend owfuzz with the additions in Table 1, keeping the agent/substrate boundary thin. The agents see only a corpus-in, results-out contract, so they drive any substrate honoring it, and a new OTA protocol needs only a new substrate and a specification to read. Two limits are structural. Transport is fixed to 802.11 monitor injection, and without a credential, fuzzing reaches only the pre-authentication parser surface that owfuzz already targets.

4. Preliminary Evaluation

We evaluate our prototype against three research questions. **RQ1** asks whether the agent-generated corpus achieves broader *spec-valid and stateful* coverage (frame subtypes, IEs, and protocol states reached) than owfuzz's hand-written baseline. **RQ2** asks whether the end-to-end pipeline surfaces anomalies on a real device and makes progress toward reproducing a known vulnerability. **RQ3** asks whether the multi-agent feedback loop improves coverage or anomaly discovery across rounds.

Table 1. Extensions to the owfuzz substrate and the capability each unlocks. The agent layer is unchanged across all of them.

Extension	Capability unlocked	Evidence
Corpus input + results log + OTA liveness	Closes the agent loop by replaying an external corpus and measuring per-frame outcome and crashes with no IP access	all runs
Frame-construction registry	Pluggable per-type construction	extensibility
State-tagged stateful injection	Fuzzed frames land at the matching handshake state of a live exchange	runs 1–4
Data-defined flow engine (`-F`)	Drives spec-derived multi-step handshakes not hardcoded in C	RQ2
Credential-assisted signing (`-K`)	Crosses the crypto gate into post-auth parsers (a coverage technique)	RQ2, §5
CCMP encrypted-data fuzzing	Reaches decrypt and post-decrypt paths	RQ2
Target-side RX hook (optional)	Delivery and crash ground truth where the device can be instrumented	§5

Each experiment maps to one question, so the campaigns can be read against a fixed purpose. The four over-the-air campaigns (Table 2) establish corpus fidelity and cross-target portability; the agent-vs-native comparison on the same AP (Table 3) answers RQ1; the credential-assisted four-way handshake and the CCMP replay answer RQ2; and the v1→v2 feedback rounds on the same AP bear on RQ3.

Setup. We ran four over-the-air campaigns with owfuzz in frame-replay mode. The first two (v1, v2) hit a commodity access point on 2.4 GHz, ablating the pipeline before and after a mutation-fidelity fix; the third replays the v2 corpus unchanged against a connected client station on 5 GHz (channel 100), isolating cross-target portability; the fourth (v2+FF) adds fixed-field operators that corrupt the mandatory pre-IE fields, reaching the driver's state-machine path rather than the IE loop. The pipeline covers management frames (Beacon, Probe Request, Authentication, Association Request) and runs $R{=}2$ feedback rounds (the v1→v2 refinement below is one). The campaign windows differ (the AP runs ran longer than the 60 s client runs), so the raw frame counts in Table 2 are not directly comparable across columns; we read the table through its structural metrics (IE population, on-air length range, mutation classes) rather than through volume. The Spec-Reader and Feedback/Triage agents use Claude Opus 4.7 (`claude-opus-4-7`) with schema-constrained output, default decoding, and the specification cached as a stable prompt prefix; the Synthesizer is deterministic. We withhold device and exploit specifics pending coordinated disclosure.

Metrics. Code coverage is unavailable on commodity devices, so we report black-box metrics observable over the air. These are *specification coverage* (frame subtypes, IE identifiers, and protocol states exercised); *frame survival* (whether injected frames are parsed and acted on without the target faulting); *anomalies* (non-compliant responses and crashes, inferred from loss of beacon liveness and cross-checked against the kernel log where available); and *cost*, reported qualitatively (model, rounds, token budget; Setup). Frames-to-first-anomaly and anomaly rate are not yet quantified, as no on-device anomaly occurred; a known-vulnerability reproduction is the focus of ongoing campaigns.

Results. Table 2 ablates the agent pipeline on a hardened commodity access point. The first corpus (v1) exposed a generation bug. Told to ground every field strictly in the spec text, the Spec-Reader produced structurally complete but value-empty IEs (only 3

Table 2. Preliminary results across four over-the-air campaigns. Campaigns 1–2 ablate corpus generation (same 2.4 GHz AP); campaign 3 shows cross-target portability (same v2 corpus, different device class and band); campaign 4 adds fixed-field mutations. A first head-to-head against owfuzz's hand-written corpus is under RQ1; a full coverage comparison is ongoing.

Metric	v1 (AP)	v2 (AP)	v2 (STA)	v2+FF (STA)
IE values populated	3/15	15/15	15/15	15/15
Unique frame vectors	28	18	18	20
On-air length range (B)	28–66	30–348	30–348	30–327
Frames replayed (OTA)	23,672	24,738	8,093†	7,773†
IE length/size mutations	no	yes	yes	yes
Fixed-field mutations	no	no	no	yes
Target band / mode	2.4 GHz/AP	2.4 GHz/AP	5 GHz/STA	5 GHz/STA
Crashes (liveness loss)	0	0	0	0

†60 s replay window; AP campaigns ran longer.

of 15 carried a payload), so the size- and length-scaling operators became no-ops, leaving the oversized and length-overflow frames identical to their baselines. Tightening the value contract and decoupling those operators from the baseline length (v2) gave every IE a representative payload, widening the on-air length range from 28–66 B to 30–348 B. Across both runs (≈24k frames each) the AP kept beaconing with zero crashes and no kernel oops/panic markers in its log. The smaller v2 vector count is a deliberate budget reallocation: the feedback agent dropped the low-surface Probe Request family for the operators that paid off (see RQ3). These runs establish corpus fidelity; RQ2 turns to vulnerability reproduction.

A third campaign replayed the v2 corpus unchanged against a connected client station on 5 GHz (channel 100), changing only the owfuzz target flags; it stayed alive across 8,093 frames, showing the corpus carries to a different device class and band. A fourth (v2+FF) added fixed-field operators that corrupt the mandatory pre-IE fields of Authentication and Association Request frames (algorithm number, transaction sequence, status code, capability flags, listen interval), reaching the driver's 802.11 state-machine code, a different path from the IE-parsing loop; all 7,773 frames survived on the Linux client. Both targets run widely deployed, maintained stacks, so a clean no-fault result is the honest baseline; extending to weaker embedded clients is the immediate next step (Section 5).

RQ1. Agent corpus vs. owfuzz's hand-written corpus. Table 3 compares the two on the same AP. owfuzz's native generator is broader in *raw* reach (any IE id and subtype) but sprays statelessly, with almost all of it Deauthentication/Disassociation that never advances a session. The agent corpus trades that breadth for structural validity. Every frame is spec-grounded, all 15 targeted IEs carry spec-valid payloads, and it drives the target through authentication, association, and the four-way handshake (RQ2). So the answer is split. The agent corpus is broader in spec-valid, stateful coverage while narrower in raw count, and a quantified per-parser comparison is still ongoing.

RQ2. Stateful depth and a downstream fault. The clearest evidence the corpus reaches code a stateless one cannot is a credential-assisted WPA2 four-way handshake. Driving owfuzz interactively with an agent-generated EAPOL flow and the network cre-

Table 3. RQ1 coverage on the same AP. owfuzz's native generator wins *raw* reach (it can emit any IE id and subtype) but sprays statelessly; the agent corpus wins *spec-valid, stateful* coverage.

Coverage dimension (same AP)	Native	Agent
Distinct IE ids emitted (raw)	256	15
Distinct mgmt subtypes exercised[a]	2	3
IEs populated with spec-valid values	0[b]	15/15
Protocol states reached	0	Auth→Assoc→4-way[c]

[a]Of native's 43,886 frames in 40 s, 99.8% were Deauth/Disassoc. [b]Native emits ids 0–255 with arbitrary, non-spec bytes. [c]Via the credential-assisted flow (RQ2); native is stateless.

dential at runtime, our station completed authentication and association and sent a *fuzzed, MIC-valid* Message 2; the AP replied with Message 3 88 times, each reply confirming it validated the forged-MIC M2 and advanced into the post-MIC key-data parser. Completing the handshake this way is a coverage technique on one's own network, reaching a parser the four campaigns of Table 2 do not. Separately, the engine's encrypted output crashed a downstream decoder. Replaying CCMP frames through `airdecap-ng` 1.7 reproducibly segfaults on any frame whose decrypted plaintext is shorter than the 8-byte LLC/SNAP header. So RQ2 is partly answered. The pipeline reaches anomaly-bearing parsers and produced one downstream crash, but has not yet faulted the wireless device itself, which our ongoing campaigns pursue on weaker targets.

RQ3. Does the feedback loop help? We do not yet claim a quantified answer. The v1→v2 transition bundles two changes, a mutation-fidelity bug fix and the feedback agent's budget reallocation (dropping Probe Request, reweighting operators), so it cannot separate the loop's effect from the fix. A clean test holds the schema fixed and compares one round with the fuzz plan disabled against one round with it enabled, on spec coverage and anomaly rate; our metric harness for this is in place, but the controlled campaign is pending hardware time. We therefore report RQ3 as *open* in this preliminary study, with only the qualitative signal that the agent reallocated budget away from the low-surface Probe Request family, and we do not attribute a coverage gain to the loop.

Threats to validity. Our systematic campaigns (Table 2) run on two commodity devices, an access point and a connected client, and stay within management frames; the credential-assisted runs of RQ2 additionally reach the EAPOL handshake and the encrypted data path on the access point. This is a small, mature device set, so wider coverage is underway. We limit LLM nondeterminism with a pinned model (Setup), schema-constrained output, and logged prompts and outputs; the model exposes no temperature control, so runs reproduce structurally but are not bit-identical. Being black-box, we infer crashes from loss of OTA liveness (cross-checked against the kernel log where available), which can miss silent faults; target-side instrumentation is in progress.

Availability. To support reproducibility we will release the non-sensitive parts of the pipeline. These are the agent source (Spec-Reader, Synthesizer, Feedback/Triage, and orchestrator), the prompts and JSON schemas, the generated sample corpora, the metric and experiment scripts, and the owfuzz substrate extensions of Table 1, at `https://github.com/dukptkey/owfuzzai`. The target's identity, the baseline frames captured from it, and exploit specifics stay withheld pending coordinated disclosure, and will be added to the release once disclosure completes.

5. Research Agenda

The agenda follows from one rule: the model proposes *structure*, code supplies *constants*, instrumentation *confirms* delivery.

Automated grounding. A specification gives structure but omits target-specific constants (byte offsets, on-air encapsulation, accepted baseline frames), which a short manual step supplies in our prototype. For the WPA2 four-way handshake, for example, we read one captured exchange to fix the EAPOL key-information byte offset and the association-request template the access point accepts before the flow could run. This step should become a fourth *Grounder* agent that reads such constants from instrumentation. The Synthesizer can likewise become constraint-driven, enumerating boundary and violation cases from the Spec-Reader's per-IE constraints to widen coverage deterministically.

Firmware observability and scope. On the offloaded silicon we tested, the pre-authentication handshake runs in firmware below the host receive path, so white-box visibility needs firmware-level hooks; broader multi-device evaluation is underway.

6. Conclusion

We presented ongoing work on a spec-driven, multi-agent corpus generator for over-the-air Wi-Fi fuzzing on owfuzz, replacing hand-written corpora with spec-grounded, feedback-adapted ones aiming beyond Wi-Fi.

References

Cao, H., Huang, L., Hu, S., Shi, S., and Liu, Y. (2023). Owfuzz: Discovering wi-fi flaws in modern devices through over-the-air fuzzing. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 263–273. DOI: 10.1145/3558482.3590174.

Garbelini, M. E., Wang, C., and Chattopadhyay, S. (2022). GREYHOUND: Directed greybox wi-fi fuzzing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 19(2):817–834. DOI: 10.1109/TDSC.2020.3014624.

Huang, C., Wang, D., and Zhou, Z. Q. (2025). LLM-assisted model-based fuzzing of protocol implementations. arXiv preprint arXiv:2508.01750.

Kampourakis, V., Chatzoglou, E., Kambourakis, G., Dolmes, A., and Zaroliagis, C. (2022). WPAxFuzz: Sniffing out vulnerabilities in Wi-Fi implementations. *Cryptography (MDPI)*, 6(4):53. DOI: 10.3390/cryptography6040053.

Maklad, Y., Wael, F., Hamdi, A., Elsersy, W., and Shaban, K. (2025). MultiFuzz: A dense retrieval-based multi-agent system for network protocol fuzzing. arXiv preprint arXiv:2508.14300.

Meng, R., Mirchev, M., Böhme, M., and Roychoudhury, A. (2024). Large language model guided protocol fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*.

Schepers, D., Vanhoef, M., and Ranganathan, A. (2021). A framework to test and fuzz Wi-Fi devices. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 368–370. DOI: 10.1145/3448300.3468261.

Zhong, M., Zeng, Z., Guo, Y., Zhao, D., Zhang, B., Li, S., Peng, H., and Ding, Z. (2025). Intelligent test case generation method for fuzzing iot protocols based on LLM. *Automated Software Engineering*. DOI: 10.1007/s10515-025-00557-x.