DualFuzz: Detecting Vulnerability in Wi-Fi NICs through Dual-Directional Fuzzing

Yuanliang Chen
KLISS, BNRist, School of Software
Tsinghua University, China

Fuchen Ma*
KLISS, BNRist, School of Software
Tsinghua University, China

Yanyang Zhao
KLISS, BNRist, School of Software
Tsinghua University, China

Yuanyi Li
Shuimu Yulin Technology Co., Ltd
Tsinghua University, China

Yu Jiang*
KLISS, BNRist, School of Software
Tsinghua University, China

***Abstract*—Wi-Fi Network Interface Cards (NICs) are vital for enabling wireless connectivity across a wide range of devices. Ensuring their security is critical, as vulnerabilities can expose entire networks to threats. Fuzzing is a promising technique for detecting such flaws. However, existing Wi-Fi fuzzers typically test transmission and reception separately, overlooking their interactions and resulting in inefficient testing.**

In this work, we present DualFuzz, a dual-directional fuzzing framework designed to simultaneously test both transmission and reception processes in Wi-Fi NICs. First, DualFuzz automatically identifies interaction behaviors within Wi-Fi NICs and constructs a Transmission-Reception Model (TRModel) to characterize Wi-Fi frames that influence these interactions. Leveraging this model, DualFuzz utilizes latency guided fuzzing to efficiently coordinate exploring transmission and reception interaction logics. Finally, we propose liveness and equivalence detectors that enable real-time monitoring to identify abnormal states and uncover potential vulnerabilities in Wi-Fi NICs. We implemented and evaluated DualFuzz on eight widely used Wi-Fi NICs, incorporating chipsets from various manufacturers (e.g., Intel and Realtek). Compared to state-of-the-art Wi-Fi fuzzers like OwFuzz, wpaspy, and Greyhound, DualFuzz detects 75%, 163%, and 250% more vulnerabilities, respectively. In total, it uncovered 21 previously unknown vulnerabilities, 7 of which have been assigned CVEs.

***Index Terms*—Wi-Fi NIC, vulnerability detection, fuzzing**

I. Introduction

Wi-Fi Network Interface Cards (NICs) have become essential components in modern wireless communication systems, significantly influencing mobile connectivity and wireless network performance [54], [20]. By converting data into radio signals and transmitting over networks, Wi-Fi NICs enable efficient communication and maintain reliable, high-speed connectivity for personal and industrial usage. As a result, they are widely used in various scenarios, including smart homes, industrial automation, and medical devices [2], [49], [19].

However, as Wi-FI NICs are applied in an increasing number of scenarios, their security has become a critical concern. Vulnerabilities within these components can be exploited by attackers using maliciously crafted packets, potentially leading to serious consequences such as device crashes, system outages, and unauthorized access [41], [55]. For example, the Kr00k vulnerability (CVE-2019-15126) [12], [10] affected billions of devices using Wi-Fi NICs with chips manufactured by Broadcom and Cypress, allowing attackers to intercept and decrypt sensitive data by exploiting flawed encryption during disconnections. This severe flaw led to significant economic losses due to data breaches, compromised privacy, and heightened security risks for businesses and consumers. Thus, ensuring Wi-Fi NIC security is essential to safeguard networks and connected devices from potential threats.

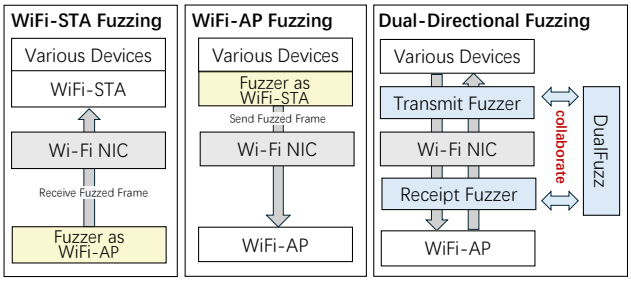

Fig. 1. DualFuzz focuses on the interaction logic between frame transmission and reception by simultaneously exploring the generation of both transmitted and received Wi-Fi frame testcases. In contrast, existing Wi-Fi fuzzers explore the transmission and reception logic separately.

Fuzzing has emerged as a powerful automated vulnerability detection technique for diverse Wi-Fi devices [45]. Many Wi-Fi fuzzers have proposed various fuzzing strategies [36], [9], [50], [17], successfully detecting a significant number of vulnerabilities across different devices. In Wi-Fi network architectures, there are two device roles: Wi-Fi Access Points (APs), such as wireless routers, gateways, or hotspots, which provide wireless connectivity by linking client devices to a wired network; and Wi-Fi Stations (STAs), such as smartphones, tablets, or security cameras, which act as client devices that connect to APs for network access [40], [43], [52]. Based on their testing roles, existing Wi-Fi fuzzers can be categorized into two main types, as shown in Figure 1. The first type treats the fuzzer as a Wi-Fi STA, generating a large number of Wi-Fi frames as test inputs to continuously fuzz Wi-Fi AP devices. The second type positions the fuzzer as a Wi-Fi AP, producing numerous Wi-Fi frames to continually test Wi-Fi STA devices.

*Fuchen Ma and Yu Jiang are the corresponding authors.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00069

However, existing Wi-Fi fuzzers typically focus on unidirectional fuzzing, targeting either Wi-Fi STA or Wi-Fi AP devices separately, thereby neglecting the complex interaction logic between frame transmission and reception within Wi-Fi NICs. In practice, Wi-Fi NICs involve many intricate interactions such as acknowledgment sequences, buffer management, and retransmission protocols, creating interdependent processes between sending and receiving frames. Effectively exploring these complex interaction logics requires high-quality test cases that incorporate both transmission and reception frames with specific execution dependencies. For instance, triggering bug #1 in Table II (detailed in Section VI-B1, Case Study) requires at least four key transmission and reception interactions with well-fuzzed frames. Due to their limitation of unidirectional testing, existing Wi-Fi fuzzers struggle to explore such complex interactions, resulting in lower testing efficiency and limited vulnerability coverage.

To effectively test Wi-Fi NICs and uncover vulnerabilities hidden in complex interaction logic, we need dual-directional fuzzing that collaboratively generates both transmission and reception test frames, as shown in Figure 1. There are three main challenges: (1) **The first challenge** is to identify which Wi-Fi frame behaviors impact the transmission-reception interaction logic of Wi-Fi NICs. These interactions are complex and span multiple stages (e.g., flow control, retransmission, timing synchronization). Thus, precisely determining the critical frames that influence these interactions is challenging yet essential. (2) **The second challenge** is efficiently coordinating the generation of reception and transmission frames. Dual-directional fuzzing needs to explore both input dimensions simultaneously. The vast number of combinations between two dimensions significantly expands the test space, making it hard to explore. (3) **The third challenge** is designing precise detectors to determine whether the NIC behaves correctly. Wi-Fi NIC behavior is dynamic and complex, and even if vulnerabilities are triggered, they are hard to detect – especially in black-box settings with no visibility into internal states.

To address these challenges, we propose DualFuzz, a fuzzing framework that effectively coordinates the testing of frame transmission and reception logic within Wi-Fi NICs. First, according to the IEEE 802.11 standard [24], DualFuzz automatically analyzes and extracts the shared data (e.g., sequence numbers, retransmission counters, acknowledgment states, etc.) accessed by both the transmission and reception frame handling processes. Then, DualFuzz constructs a Transmission-Reception Model (TRModel), modeling all Wi-Fi frames that operate (i.e., read or write) on these shared data items. Second, considering that frame processing latency can reflect the internal complexity of transmission-reception interactions within NICs, DualFuzz introduces a latency-guided fuzzing algorithm to efficiently coordinate the exploration of both transmission and reception frames. Finally, to precisely detect vulnerabilities, DualFuzz introduces two anomaly detectors to monitor the abnormal status of Wi-FI NICs in real-time. A liveness detector is designed to check whether the NIC is online and providing service. An equivalence detector, based on a content equivalence comparator, is employed to determine whether the Wi-Fi NIC's frame processing is correct.

We implemented DualFuzz and evaluated it on eight widely used Wi-Fi NICs from various manufacturers, each equipped with different mainstream chipsets(e.g., Intel [28], Realtek [47], Mediatek [35], Broadcom [8]). Compared with state-of-the-art Wi-Fi fuzzers, OwFuzz [9], wpaspy [50], and Greyhound [17], DualFuzz detects 75%, 163%, and 250% more vulnerabilities, respectively. In total, DualFuzz discovered 21 previously unknown vulnerabilities, 7 of which have been assigned CVEs.

In summary, we make three key contributions:

- We design a dual-directional fuzzing framework and propose a latency guided fuzzing strategy to collaboratively generate test frames for both transmission and reception.
- We propose two bug detectors for precisely identifying anomalies and vulnerabilities in Wi-FI NICs.
- We implement and evaluate DualFuzz on eight widely used Wi-Fi NICs. We will open-source DualFuzz[1]. Currently, it has detected 21 previously unknown vulnerabilities.

II. Background of Wi-Fi NICs

As shown in Figure 2, a typical Wi-Fi NIC is connected to the host, enabling communication with various wireless devices through the transmission and reception of Wi-Fi frames [14]. The architecture of a Wi-Fi NIC is organized into four main layers [38]: The PHY layer handles wireless signal transmission and reception, modulating data into radio waves and demodulating incoming signals into packets [57]. The MAC layer manages medium access, including frame addressing, collision avoidance, and reliable delivery [42]. The Firmware layer, typically closed-source and running on the NIC's MCU, processes frames for tasks like ACK handling, authentication, flow control, and queue scheduling [27], [37]. The Driver layer, residing on the host, coordinates transmission and reception by managing queues and interacting with the firmware to prioritize packets [23].

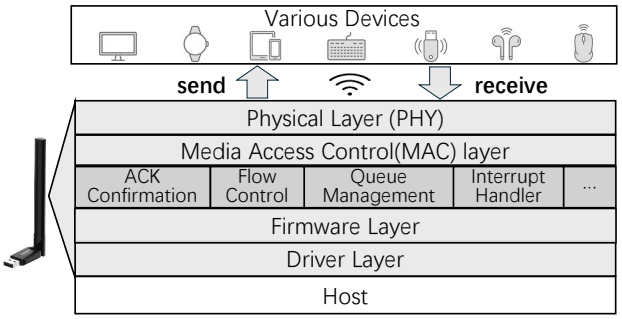

Fig. 2. The architecture of a Wi-Fi NIC consists of four layers: the Physical, MAC, Firmware, and Driver Layer.

The complexity of Wi-Fi NICs primarily arises from intricate dependencies between frame transmission and reception, often involving shared data such as ACK timers, sequence

[1]DualFuzz at: https://anonymous.4open.science/r/DualFuzz-A6F0

numbers, retransmission counters, and acknowledgment statuses. *Acknowledgment Mechanism*: After sending a frame, the sender expects an ACK from the receiver. If none is received within a set timeout, retransmission occurs. While ensuring reliable delivery, this dependency complicates timing coordination and impacts performance [30]. *Flow Control and Buffer Management*: These regulate data flow based on buffer availability. When the receiver's buffer nears capacity, it signals the sender to slow down, preventing overflow and maintaining stable communication [7]. *Queue Management*: Frames are organized into TX and RX queues [25]. Prioritization ensures timely handling of critical frames, but managing latency and overflow under load increases complexity. *Retransmission Logic*: Relies on shared data like sequence numbers and retry counters, requiring tight synchronization between sender and receiver [62].

III. Motivation Example

Threat Model: Throughout this paper, we define a Wi-Fi communication system as a tuple $\Psi = \{STA, AP, WN\}$, where STA represents the station device; AP denotes the access point device interacting with the STA; WN is the Wi-Fi NIC under test. We assume that the attacker compromises either STA or AP, but **not both simultaneously**. From one compromised side, the attacker can inject arbitrary Wi-Fi frames at any time, targeting any stage of the communication process (e.g., association, transmission, teardown). The goal of the attacker is to exploit logic flaws in the dual-directional processing of Wi-Fi frames within the NIC, particularly those involving shared internal states (e.g., ACK tracks, retry counters, buffer queues), and induce system crashes, memory corruptions, or undefined behavior in Wi-Fi NICs.

Although some bugs hidden in deep interaction logic can be triggered and exploited from a single side, as shared states can be gradually and predictably influenced using carefully crafted frame sequences, this often requires precise timing or complex multi-step manipulation, making such bugs difficult to uncover through automated testing. To efficiently explore transmission-reception interactions and systematically uncover vulnerabilities, performing fuzzing on both ends during testing is a practical and effective approach.

One such example is CVE-2022-21745 [13], found in Mediatek Wi-Fi NICs, where incorrect buffer deallocation leads to a dangling pointer. Subsequent improper access to this pointer triggers a use-after-free vulnerability, posing a serious threat to network security. An attacker could exploit this flaw to launch remote DDoS attacks or even data leakage. Figure 3 illustrates the seven key steps to trigger this bug, and Figure 4 presents the core code snippet of this vulnerability.

In a Wi-Fi NIC, QoS Data frames carry prioritized traffic, ACK frames confirm successful reception, SEQ numbers track frame order, the Retry bit indicates retransmissions, and the 'tx_buffer' temporarily holds frames waiting to be sent. As shown in Figure 3, when a QoS Data frame is transmitted, the Wi-Fi NIC stores it in the 'tx_buffer' based on its SEQ number. Upon receiving a corresponding ACK, the NIC frees

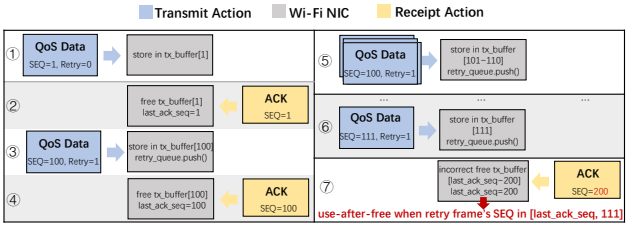

Fig. 3. CVE-2022-21745, an Use-After-Free in Mediatek Wi-Fi Firmware that could lead to NIC crash, data leakage, or remote privilege escalation.

```
1  void retry_frame() {
2     ...
3     uint16_t retry_index =
          retransmission_queue->pop();
4     retransmission_counter[retry_index]++;
5  -- transmit(tx_buffer[retry_index]);
6  ++ if (tx_buffer[retry_index])
       transmit(tx_buffer[retry_index]);
7  }
8  void handle_ack(uint16_t ack_seq) {
9     ...
10    for (uint16_t idx=last_ack_seq+ 1; idx <=
          ack_seq; idx+=1){
11       if (tx_buffer[idx]) {
12          free(tx_buffer[idx]);
13 ++       tx_buffer[idx] = NULL;
14          retransmission_counter[idx] = 0; } }
15    last_ack_seq = ack_seq;
16 }
```

Fig. 4. The core code snippet of CVE-2022-21745. Use dangling pointer tx_buffer(idx) in 'retry_frame()' after it was freed in 'handle_ack()'.

the associated entry from the buffer. If a QoS frame is sent with the Retry bit set, the NIC also places it into the 'retry_queue' for subsequent retransmission. However, when a large number of QoS Data frames are awaiting retransmission and accumulate in the 'retry_queue', receiving an ACK with a large SEQ number (e.g., 200) can cause the Wi-Fi NIC to mistakenly free 'tx_buffer' from last_ack_seq up to the large SEQ. This results in dangling pointers. If the 'retry_frame' function later accesses these freed pointers, it triggers the use-after-free vulnerability, leading to a Wi-Fi NIC crash (if the memory is invalid) or potential data leakage (if the memory has been reallocated to other users). This vulnerability has been fixed by setting the pointer to NULL after it is freed and adding a validation check before reuse (Lines 6 and 13).

Wi-Fi NICs are widely used in laptops, smartphones, IoT, and industrial systems, and their vulnerabilities can compromise entire networks through crashes, unauthorized access, or data leaks. We can draw **three important lessons** from this case: (1) Some vulnerabilities are hidden in complex interactions between transmitted and received frames, and can only be triggered by mutating both directions, such as the transmitted QoS Data frame and the received ACK frame in this case. To address this, DualFuzz adopts a dual-directional fuzzing approach that simultaneously tests both transmission and reception logic. (2) Not all Wi-Fi frames or fields impact transmission-reception interaction. Only specific fields operating on shared data (e.g., sequence numbers, ACK status in this case) are relevant. Blindly mutating all fields reduces efficiency. To solve this problem, DualFuzz first performs static

analysis on the Wi-Fi NIC design specification (i.e., the 802.11 standard) to identify shared data, and then extracts relevant frames and key fields to construct a TRModel that guides targeted mutations. (3) Some vulnerabilities require specific execution dependencies between both frame transmission and reception (the seven key steps in Figure 3) to trigger, which greatly expand the input space. To handle this issue, DualFuzz employs latency guided fuzzing, dynamically selecting high-quality transmission-reception frame combinations to explore as much interaction logic as possible.

IV. Design

Design goal: A practical Dual-Directional Wi-Fi fuzzing framework should have the following properties.

- *General:* DualFuzz is designed to find vulnerabilities in most practical Wi-Fi NICs and supports mainstream Wi-Fi chipset vendors such as Intel, Realtek, MediaTek, and Broadcom. The tool can be quickly deployed to test different types of Wi-Fi NICs with only minor adjustments.
- *Non-intrusive:* Most Wi-Fi NICs are closed-source, neither can DualFuzz directly access their source code nor modify their firmwares or drivers. Therefore, DualFuzz is designed as a **black-box fuzzer** that relies solely on runtime external status (e.g., online/offline, handling time, frame content).
- *Efficient:* DualFuzz is able to frequently exercise the transmission-reception interactions and effectively detect vulnerabilities in real-world Wi-Fi NICs within 24 hours.
- *Accurate:* DualFuzz is designed to have satisfying precision and recall to avoid reporting false positives.

A. DualFuzz Workflow

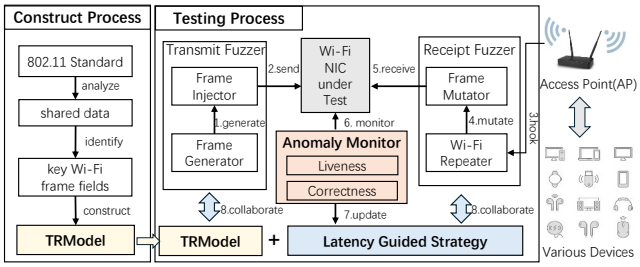

Fig. 5. The workflow of DualFuzz. It includes three key components: (1) TRModel for describing Transmission-Reception Model; (2) Latency Guided Fuzzer for coordinated generation of TX/RX test frames; (3) Anomaly Monitor for identifying bugs in Wi-Fi NICs.

Figure 5 illustrates the workflow of DualFuzz, consisting of two main phases. The first phase is the TRModel construction process. In this phase, DualFuzz first analyzes the IEEE 802.11 standard to automatically extract the shared data involved in the transmission and reception processing (e.g., SEQ, ack, buffer queue). It then identifies the fields in Wi-Fi frames that operate on this shared data, and models them into a TRModel. The second phase is the testing process, which involves the following 8 steps: (1) DualFuzz first generates a transmit Wi-Fi frame based on the TRModel. (2) The Frame Injector sends this test frame (F_{tx}) to the device through the Wi-Fi NIC under test. (3) Subsequently, the response from the device return through the Wi-Fi AP, where a Wi-Fi repeater hooks this frame and forwards it to the Recept Fuzzer. (4) The Frame Mutator is employed to mutate the hooked frame based on the TRModel. (5) The Receipt Fuzzer sends the mutated frames (F_{rx}) back to the Wi-Fi NIC under test for processing. (6) Anomaly Monitor observes and records the runtime status (e.g., connection status, etc.) of the Wi-Fi NIC. (7) Meanwhile, DualFuzz measures the processing latency of both F_{tx} and F_{rx}, and updates the results to the fuzzer for guidance. (8) If a new anomaly is detected or longer processing latency is observed, the current test frame pair $\langle F_{\text{tx}}, F_{\text{rx}} \rangle$ is considered an interesting seed and is prioritized in the seed pool to help collaborate and guide subsequent TX/RX generation. DualFuzz then proceeds to the next iteration (from step 1 to step 8) until the testing process terminates.

B. TRModel Construction

Definition of transmission-reception interaction: In a Wi-Fi NIC, let F_{tx} denote a transmitted Wi-Fi frame, F_{rx} denote a received Wi-Fi frame. Let $S = \{s_1, s_2, \ldots, s_n\}$ represents the set of shared data structures (e.g., sequence number, retry counter, buffer, ACK state) within the Wi-Fi NIC. Let $Access(F, s)$ denote the access relation indicating whether frame F reads or writes shared data item $s \in S$. A transmission-reception interaction is defined as:

$$\text{Interact}(F_{\text{tx}}, F_{\text{rx}}) \iff \exists s \in S,\ \text{Access}(F_{\text{tx}}, s) \wedge \text{Access}(F_{\text{rx}}, s)$$

That is, a transmitted frame F_{tx} and a received frame F_{rx} are interacting if they both access at least one shared data item s.

TRModel Construction: Considering that all Wi-Fi NICs are implemented in compliance with the IEEE 802.11 standard, DualFuzz performs static analysis of the specification to automatically analyze transmission-reception interaction logic and construct TRModel, involving the following four steps:

Step 1. Filtering Sections Describing Key Operational Behaviors in the IEEE 802.11 Standard: DualFuzz first scans the entire specification using a PDF layout parser to extract structured section headings, tables, and paragraphs. We identify sections covering critical NIC behaviors such as retransmission, acknowledgment, buffer management, and frame sequencing(as introduced in Section II). Each subsection is then classified as either transmission or reception logic based on heuristic keyword matching(e.g., "transmit", "send", "enqueue", etc. for TX; "receive", "process", "handle" for RX) and verb-level dependency parsing.

Step 2. Extracting Shared Data and Analyzing Wi-Fi Frame Types: From transmission (TX) subsections, we extract a data set DS_{tx} representing internal state variables updated or accessed during transmission. Similarly, DS_{rx} is extracted from RX logic. Their intersection $S = DS_{\text{tx}} \cap DS_{\text{rx}}$ forms the set of shared state variables (e.g., `seq_num`, `retry_flag`, `ack_state`). We then parse all frame formats in Section 9.4, extracting field names, bit offsets, lengths, and associated frame types, forming the set Set_{type}.

Step 3. Performing Static Semantics Analysis for Each Field: For each field $f \in F$, we retrieve its description and apply dependency parsing (via the tool spaCy [56]) to

determine whether it interacts with any shared variable $s \in S$. We mark it as *Read* (R) if it queries NIC state, or *Write* (W) if it modifies NIC state. For example, it detects that the Sequence Control (SC) field in QoS Data frames updates the transmission buffer index and influences retry logic; the Retry flag affects retransmission counters; the Frame Control (FC) field governs acknowledgment behavior and protection settings; and the Data Size field impacts buffer management. By automatically analyzing these field-to-shared-data interactions, DualFuzz constructs the TRModel, as illustrated in Figure 6.

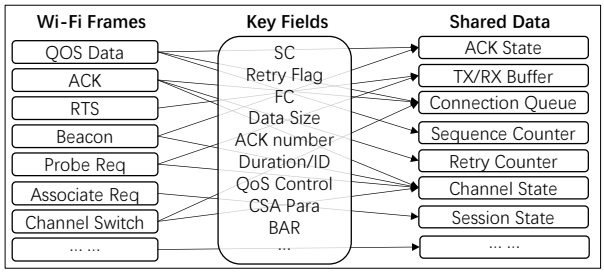

Fig. 6. TRModel for describing key interactions of frame transmission and reception within Wi-Fi NICs.

Step 4. Mapping and Refining the TRModel: Existing Wi-Fi fuzzers use Scapy [6] for packet generation and mutation. We align the constructed TRModel with Scapy's Wi-Fi model, which follows the IEEE 802.11 specification. We perform field-level mapping between the TRModel and Scapy definitions (which retain standard-compliant field names). If inconsistencies or unmapped fields are encountered, they are pruned from the TRModel to ensure correctness and compatibility with fuzzing execution.

Initial Test Frames Generation: Before starting the testing phase, DualFuzz creates initial test frames to drive the fuzzing process. Since not all frame fields affect transmission-reception interaction logic, DualFuzz focuses only on generating and mutating the key fields modeled in the TRModel. It first randomly selects combinations of transmission and reception frame types to generate frame pairs $\langle F_{tx}, F_{rx} \rangle$. For each test frame, DualFuzz applies customized mutation strategies based on the shared data it operates on and initializes key fields accordingly. For example:

- ACK State: DualFuzz maintains a list $list_{acks}$ to track all ACK frames. For each ACK, it introduces a random delay from 0 to max_t+1 seconds, where max_t is the NIC's configured ACK timeout. Additionally, if ACK aggregation [22] is enabled, DualFuzz randomly mutates the number of ACKs aggregated in a single frame from 0 to max_{agg}, the NIC's maximum supported value.
- TX/RX Buffer: To test buffer interaction logic effectively, DualFuzz creates boundary scenarios for data size. For 802.11a/b/g/n [1], with a typical MTU of 1500 bytes, it generates frame sizes between 1499 and 1501 bytes. When AMSDU [53] is used, supporting up to 7935 bytes, it generates sizes near this limit. DualFuzz also crafts concurrent frames near the NIC's maximum parallel capacity max_p.
- Connection Queue: DualFuzz uses a map, map_{con}, to track all Wi-Fi connections. To test boundary conditions, it maintains the number of connections near the NIC's limit max_{con}. Each connection's priority, channel, and duration are randomly assigned, and DualFuzz randomly disconnects and establishes new connections during testing.
- Counters: For counter-type shared data such as the Sequence Counter and Retry Counter, DualFuzz performs targeted and random mutations on related frame fields to simulate edge-case behaviors. For the Sequence Control field, it mutates sequence numbers with rapid increments, large jumps, or wraparounds. For the Retry flag, it randomly toggles the bit and resends frames with the same sequence number to emulate retry scenarios.

Moreover, more detailed mutation designs and implementations are available in our git[1]. Based on this approach, we design tailored mutation strategies for different frame fields in the TRModel according to their semantic meanings. Since these semantics are derived from the IEEE 802.11 standard, which all Wi-Fi NICs follow, the mutation strategies proposed by DualFuzz are generalizable across different Wi-Fi devices.

C. Processing-Latency Guided Fuzzing

Due to its black-box design, DualFuzz cannot access internal runtime data of some Wi-Fi NICs (e.g., code coverage, which is commonly used by fuzzers). Meanwhile, randomly generating transmission-reception frame combinations leads to low testing efficiency. To efficiently coordinate exploration across the two input dimensions (i.e. transmission and reception), DualFuzz adopts a key heuristic: the longer a frame's processing latency, the more likely it has exercised deeper interaction logic within the Wi-Fi NIC, and thus, the higher the chance of triggering hidden bugs. Therefore, for each test frame pair $\langle F_{tx}, F_{rx} \rangle$, DualFuzz measures and records the processing latency of both F_{tx} and F_{rx} on the target NIC, using this feedback to help guide and prioritize subsequent fuzzing. We use $latency_{f_i}$ to denote the processing time of a frame f_i, Avg_T to represent the average processing time of all transmission frames, and Avg_R for the average processing time of all reception frames. We define a test frame pair $\langle F_{tx}, F_{rx} \rangle \in seedPool$ as a 'complex interaction' if $(latency_{f_{tx}} + latency_{f_{rx}}) \geq (Avg_T + Avg_R)$; otherwise, it is considered a 'simple interaction'.

Algorithm 1 illustrates the processing-latency guided fuzzing process. In the initial phase, as shown in lines 1-4, a bug detector is initialized to monitor the runtime status of the Wi-Fi NIC in real time, and a timer is set up to measure the processing latency of each test frame. The Wi-Fi NIC under test begins by processing the initial test frame pairs from the seed pool $seedPool$, during which the timer records the latency for each frame and calculates the average processing times for transmission and reception frames, respectively. In each fuzzing iteration, DualFuzz dequeues a frame pair $\langle F_{tx}, F_{rx} \rangle$ from the $seedPool$ and mutates it into a new pair $\langle F'_{tx}, F'_{rx} \rangle$ based on the mutation strategy described in Section IV-B. The Wi-Fi NIC then processes this

mutated transmission-reception pair, and the timer dynamically measures the processing latency of both F'_{tx} and F'_{rx}. If the processing latency of both F'_{tx} and F'_{rx} is greater than or equal to the average latency, then the test frame pair $\langle F'_{tx}, F'_{rx} \rangle$ is regarded as an interesting input and is stored back into the seed pool to guide subsequent fuzzing, as shown in lines 10–11. Meanwhile, the bug detector monitors the runtime status of the Wi-Fi NIC and checks for abnormal behaviors, as outlined in lines 13–17. If any new anomalies are detected, DualFuzz logs the corresponding test case and adds $\langle F'_{tx}, F'_{rx} \rangle$ into the seed pool for further exploration. Through this process, DualFuzz continuously generates high-quality, collaborative transmission-reception test inputs and effectively explores deep interaction logic in Wi-Fi NICs.

Algorithm 1: Latency Priority Fuzzing Process.

```
   Input  : NIC: Wi-Fi NIC under Test
   Output: B: Bugs
1  B={}, seedPool = initialSeed() ;
2  Detector = setupDetector();
3  timer = setupTimer();
4  Avg_T, Avg_R = NIC.process(timer, seedPool);
5  while true do
6      ⟨F_tx, F_rx⟩ = seqPool.dequeue();
7      ⟨F'_tx, F'_rx⟩ = mutate(⟨F_tx, F_rx⟩);
8      latency_t, latency_r =
         NIC.process(timer, ⟨F'_tx, F'_rx⟩);
9      if (latency_{f_tx} + latency_{f_rx}) ≥ (Avg_T + Avg_R)
        then
10         seedPool.enqueue(⟨F'_tx, F'_rx⟩) ;
11         Avg_T, Avg_R.update(latency_{f_tx}, latency_{f_rx});
12     end
13     async:
14         newBug = Detector.checkBug());
15         B.add(newBug);
16         seedPool.enqueue(⟨F'_tx, F'_rx⟩) ;
17     end async;
18 end
```

D. Anomaly Monitor

To ensure generality, DualFuzz is designed as a black-box tool. Therefore, the Anomaly Monitor is implemented to track the runtime behavior of the Wi-Fi NIC under test without requiring intrusive access to internal data such as source code or proprietary interfaces. Figure 7 shows the process. It includes two main types of bug detectors: The first is a liveness detector, which checks whether the Wi-Fi NIC has gone offline or is no longer functioning correctly due to issues such as crashes or connection failures. The second is an equivalence detector, which identifies inconsistencies between the injected and emitted Wi-Fi frames, revealing potential frame processing errors within the NIC.

Liveness Detector: The liveness detector continuously monitors the operational status of the Wi-Fi NIC to ensure it remains responsive throughout testing. It periodically sends heartbeat packets and expects timely responses. If a valid response is not received for a long time, the detector retries up to three times, following common practice adopted by prior tools. If all retries fail, the NIC is considered offline. At this point, the liveness detector reports a liveness anomaly and logs the relevant transmitted and received packets to help diagnose potential vulnerabilities associated with the abnormal state.

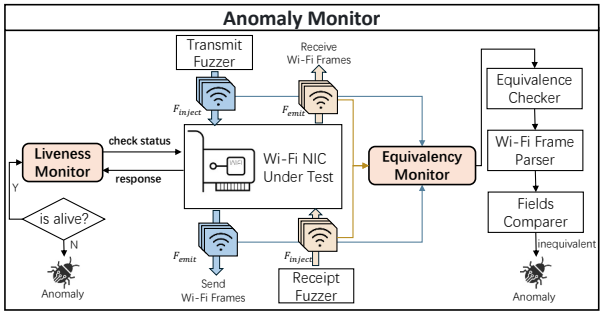

Fig. 7. Anomaly Monitor in DualFuzz for monitoring the runtime status and identifying anomalies in real time.

Equivalence Detector: The equivalence detector functions by intercepting and comparing Wi-Fi frame contents before and after they are processed by the Wi-Fi NIC's transmission and reception logic. Specifically, for frame transmission, the detector first records the frame content before it is injected into the Wi-Fi NIC, denoted as F_{inject}. After transmission, it intercepts and records the frame again by monitoring the NIC in its monitor mode, now denoted as F_{emit}, and compares these two frames to determine whether their contents are consistent. Similarly, for frame reception, the detector records the frame content before injection (F_{inject}) and again after it has been processed by the NIC and emitted as a response (F_{emit}), checking for inconsistencies.

Please note that even if the Wi-Fi NIC functions correctly without any bugs, the contents of Wi-Fi frames before and after NIC processing, namely F_{inject} and F_{emit}, are typically not entirely identical. Certain fields may change during transmission and reception, especially due to lower-layer protocol handling. For example, the Frame Check Sequence (FCS) [4], which ensures data integrity, is typically computed and appended by hardware during transmission, so it is absent from the original frame content F_{inject}. Similarly, timestamps in some management frames (such as Beacon frames) are dynamically updated based on the actual transmission time, resulting in discrepancies between the injected and emitted versions. To address these inconsistencies, reduce false positives, and improve the accuracy of equivalence detection, we propose a **key-content equivalence comparator**. This comparator focuses solely on critical fields such as payload, source and destination MAC addresses, frame type, control flags, and sequence numbers when comparing frames. Specifically, the comparator first parses both F_{inject} and F_{emit} to extract all fields, then performs a field-by-field comparison on this subset, ignoring dynamic or hardware-modified fields. If all key fields match, the frames are considered equivalent; otherwise, an equivalence anomaly is reported.

Bug Reproducer: DualFuzz collects all transmitted and received frames throughout the entire testing process, sorts them by timestamp, and stores them for further analysis. It identifies the first state in which the bug detector reports a anomaly while processing test frames, referred to as the triggering state. The state when the target Wi-Fi NIC is initialized is recorded as the starting state. When an anomaly is detected, DualFuzz replays the sequence of frames between the starting state and the triggering state multiple times to confirm consistent triggering and help analyze its root cause.

Bug De-duplicates: DualFuzz performs bug deduplication inspired by AFL's testcase minimization process [32]. It iteratively removes or merges $\langle F_{tx}, F_{rx} \rangle$ pairs while checking if the bug persists. The minimization prioritizes frame pairs that operate on shared data in TRModel, ensuring the reproducing sequence is minimal and effective. Bugs are then deduplicated by comparing minimized sequences and anomaly types. Two bugs are considered duplicates if their minimized sequences share the same structural pattern (same frame types with identical mutated fields) and trigger the same anomaly type.

V. Implementation

Figure 8 illustrates the implementation of DualFuzz, which consists of two fuzzers: **TransmitFuzzer**, responsible for generating transmission test frames, and **ReceiptFuzzer**, which mutates received frames to test the reception logic of the Wi-Fi NIC. A TRModel is implemented for coordinating these two fuzzers to explore transmission-reception interactions. We ran DualFuzz on a host machine, with the Wi-Fi NIC under test connected via USB, mPCIe, SDIO, or other interfaces. All devices were connected to a locally isolated test network, which was set up through a dedicated Wi-Fi AP. Additionally, we used a separate Wi-Fi NIC as a repeater to capture and hook received Wi-Fi frames, which were then forwarded to the receipt fuzzer for mutation.

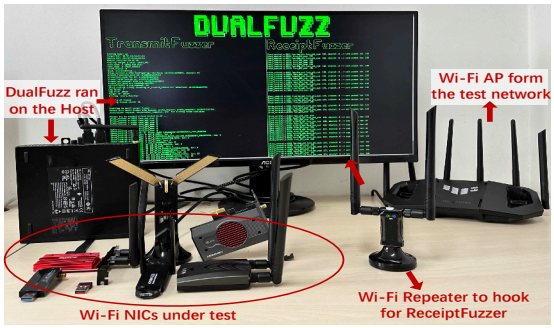

Fig. 8. Implementation of DualFuzz for fuzzing various Wi-Fi NICs.

Figure 9 presents the core components of DualFuzz, which are divided into three key parts. The first is the Coordination Controller, responsible for controlling, coordinating, and scheduling the transmit and receipt fuzzers to effectively explore the transmission-reception interaction logic of the Wi-Fi NIC. The second is the Wi-Fi Frame Generator, which parses, mutates, and generates concrete Wi-Fi frames as test inputs. The third is the Bug Detector, designed to monitor the

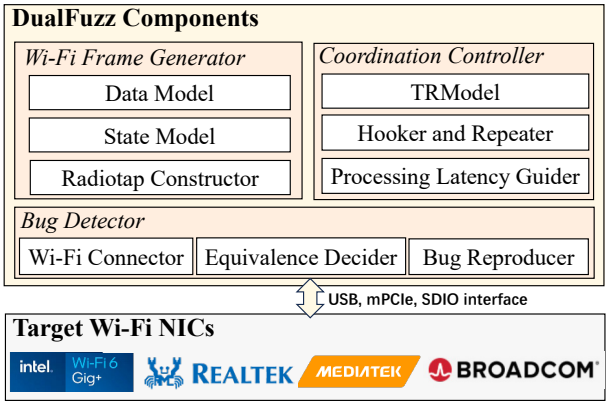

Fig. 9. Core components of DualFuzz implementation, contain three key parts: Coordination Controller, Wi-Fi Frame generator, and Interaction Adaptor.

behavior of the NIC in real time and detect abnormal states. All three components are independent of the target Wi-Fi NIC, making them reusable across different NICs. The rest of the section describes notable implementation details.

Wi-Fi Frame Generator: In network fuzzing, a data model defines the structure and format of individual messages, while a state model captures the sequence and dependencies of message exchanges within valid protocol flows. DualFuzz leverages the data and state models provided by Scapy [6], a widely used packet-generation tool in network testing and security research. Specifically, DualFuzz uses Scapy's data model to parse and format Wi-Fi frames, and its state model to generate frame sequences and drive the Wi-Fi processing workflow, including probing, association, authentication, handshake, and data exchange. For all frame pairs in the sequence involving transmission-reception interactions modeled in the TRModel, DualFuzz performs latency-guided dual-directional fuzzing. To enable fine-grained control and monitoring of Wi-Fi transmission and reception, DualFuzz leverages the Radiotap interface [5]. It handles combined packets, Radiotap headers plus 802.11 frames, using raw sockets.

Hooker and Repeater: The repeater device is configured in monitor mode to capture raw 802.11 frames with radiotap headers. Using a packet sniffer built on Scapy, the repeater continuously hooks incoming frames and filters those that match predefined criteria based on frame type, MAC addresses, and protocol fields. Once a frame is identified as relevant, it is passed to the ReceiptFuzzer, which performs mutation operations and sends the modified frame back into the Wi-Fi NIC under test.

Adaptation to New WiFi NICs: Thanks to the black-box design of DualFuzz, which offers high generality, the framework can be easily and quickly adapted to test a new Wi-Fi NIC by following three steps: (1) Connect the target NIC to the host machine using a standard interface such as USB, mPCIe, or SDIO. (2) Join the isolated test network by connecting the NIC to the same Wi-Fi access point used during testing. (3) Configure the Wi-Fi repeater to forward received frames to the target NIC using the 'sendp' function in Scapy.

VI. Evaluation

In this section, we evaluate DualFuzz to answer the following four research questions:

- **RQ1**: Is DualFuzz effective in detecting vulnerabilities of real-world Wi-Fi NICs?
- **RQ2**: Do the TRModel and latency guided fuzzing effectively improve testing performance?
- **RQ3**: What is the accuracy of DualFuzz's Detector?
- **RQ4**: Can DualFuzz cover more code of Wi-Fi NICs compared with state-of-the-art methods?

A. Experiment setup

Subject: We evaluated DualFuzz on eight real-world Wi-Fi NICs from various manufacturers, each equipped with different mainstream chipsets. Table I summarizes the detailed information of these evaluation targets. The selection of NICs was based on two criteria: (1) They should be widely deployed in real-world environments to demonstrate DualFuzz's ability to identify vulnerabilities that could impact a broad range of users; and (2) their chipsets are sourced from different mainstream vendors (e.g., Intel, Realtek, MediaTek, Broadcom), demonstrating DualFuzz's generality across diverse hardware and firmware implementations.

TABLE I
Detailed information of target Wi-Fi NICs.

NIC Model	Chipset Model	Driver Version	Firmware Version
Intel AX210NGW	Intel AX210	iwlwifi v5.10	ty-a0-gf-a0-84
TP-Link AX1800	BCM6755	v5.10.91	v222.0.144.0
ALFA AWUS036ACH	RTL8812AU	Aircrack-ng v5.6.4.2	v5.6.4.2
ASUS AX56AX	RTL8832AU	Lwfinger v5.2.2.4	v5.2.2.4
EDUP 8774M	Intel BE200	iwlwifi v6.5	gl-c0-fm-c0
Netcore NW336	RTL8188EU	Ivanovborislav v5.13.3	v5.13.3
GRiS LW04-3E03	RTL8812AE	Morrownr v5.13.6	v5.13.6
NETGD AX1800M	MT7921U	V5.12	V5.12

Compared Tools: We compared DualFuzz with three state-of-the-art Wi-Fi fuzzing tools, OwFuzz [9], wpaspy [50], and Greyhound [17], which generate Wi-Fi frames as test inputs and are capable of evaluating the Wi-Fi NIC as a whole, including its driver, firmware, and hardware layers. In addition, to assess DualFuzz's effectiveness across different layers of the NIC, we also compared it with VIRTFUZZ [26], a state-of-the-art driver fuzzer that could identify vulnerabilities in the driver layer of Wi-Fi NICs. Similarly, we included a comparison with μAFL [31], an efficient firmware fuzzer that could perform non-intrusive feedback collection to detect vulnerabilities in the firmware of Wi-Fi NICs.

Metrics and Settings: We employed two metrics for our evaluation: the number of unique bugs detected and the average speed-up to find these bugs. These metrics are commonly used to measure the effectiveness of fuzzers. We do not use coverage metrics because most Wi-Fi NICs are closed-source, making it difficult to instrument them for coverage tracking. We ran each testing tool under the same environment setup on a computer equipped with an Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz, 32 GiB of DDR4 memory, and running x86_64 Ubuntu Linux 20.04. All target Wi-Fi NICs were tested using their default configuration parameters. The test wireless network was isolated and configured locally. All experiments were repeated 10 times under identical conditions, and the average results are used in this paper.

B. Bug Detection in Real-World Wi-Fi NICs

We applied DualFuzz, OwFuzz, Greyhound, wpaspy, VIRTFUZZ, and μAFL on all 8 target Wi-Fi NICs for vulnerability detection in 24 hours. Since OwFuzz supports both Wi-Fi AP and STA fuzzing modes, we evaluated it separately in each mode for 24 hours, denoted as OwFuzz-AP and OwFuzz-STA. For VIRTFUZZ, which only supports fuzzing the driver layer, we loaded the drivers of all eight NICs into its virtual machine and executed fuzzing for 24 hours per driver. Similarly, we extracted the MCU firmware from each NIC and tested it with μAFL for 24 hours. In total, DualFuzz discovered 21 previously unknown vulnerabilities across the eight Wi-Fi NICs, significantly outperforming other state-of-the-art fuzzers, none of which detected more than 11 bugs. Detailed information on these vulnerabilities is summarized in Table II.

When DualFuzz detects an anomaly (either liveness or equivalence violation), it automatically records all Wi-Fi frames transmitted and received by the target NIC, organizing them by timestamp into a reproduction log. This log is then provided to developers, and we replay the Wi-Fi frames to help reproduce the anomalies and facilitate root cause analysis. Once an anomaly is successfully reproduced and its root cause is identified, it is confirmed as a vulnerability. As shown in Table II, the majority of the discovered vulnerabilities (9 out of 21) are buffer overflows, which can lead to severe consequences such as memory corruption, unexpected crashes, and even remote code execution. These vulnerabilities could be exploited by attackers to inject malicious payloads remotely, posing significant threats to wireless network security. In addition, DualFuzz uncovered 5 Use-After-Free and 4 NULL Pointer Dereference vulnerabilities. These issues can cause system instability, leading to crashes or undefined behavior, and may allow attackers to exploit dangling pointers or uninitialized memory access for further attacks. Two stack overflow vulnerabilities were also identified, which can corrupt stack memory and result in denial-of-service conditions due to stack exhaustion. Finally, an aggregation vulnerability in the MAC layer was discovered, where improperly validated A-MPDU frames allowed the injection of malicious subframes, leading to buffer corruption. It has already been fixed by adding strict boundary checks on subframe lengths and sequence ordering during A-MPDU reassembly.

Comparison with existing Fuzzers: In our 24-hour experiments, Greyhound, wpaspy, and OwFuzz identified 6, 8, and 11 vulnerabilities, respectively, which could be triggered via single-direction fuzzing. However, the remaining 10 vulnerabilities were hidden in the complex interaction logic between transmission and reception, requiring coordinated and interdependent TX-RX test frames. These bugs were only exposed by DualFuzz's dual-directional fuzzing strategy, which existing fuzzers lack. For the driver fuzzer, VIRTFUZZ detected only 6 vulnerabilities and missed 8 others occurs in the driver

TABLE II

21 NEW VULNERABILITIES DETECTED BY THE TOOLS WITHIN 24 HOURS. DUALFUZZ FOUND ALL 21 BUGS, INCLUDING 2 IN INTEL, 6 IN TP-LINK, 4 IN ALFA, AND 3 IN ASUS. IN COMPARISON, OWFUZZ FOUND 11 BUGS, WPASPY FOUND 8 BUGS, GREYHOUND FOUND 6 BUGS, VIRTFUZZ FOUND 6 BUGS, AND μAFL FOUND 3 BUGS, RESPECTIVELY. WE ANONYMIZE THE BUG IDENTIFIERS FOR THE DOUBLE-BLIND PROCESS.

#	Wi-Fi NIC Model	Root Cause Analysis	Location	Identifier
1	Intel AX210	A Use-After-Free occurs in function 'rc80211_wext_compat' when frequently requesting large data frame with ack timeout.	Driver layer	Bug-13733
2	Intel AX210	A Buffer Overflow occurs in function 'rc80211_minstrel_ht' when updating rate adaptation with rapid transmissions.	Firmware layer	Bug-13731
3	TP-Link AX1800	A Buffer Overflow occurs in function 'halbb_8852a_2' when transferring large-size frames and switching channel.	Driver layer	CVE-2024-56852
4	TP-Link AX1800	A NULL Pointer Dereference occurs in function 'pfx_disconnect_hdl' caused by unexpected events with state uninitialized.	Driver layer	Bug-4120
5	TP-Link AX1800	A Buffer Overflow occurs in function 'pfx_hal_rtw_memcpy' caused by oversized frame payloads without boundary checks.	Firmware layer	Bug-4121
6	ALFA AWUS036ACH	A Use-After-Free occurs in function 'idempotent_init_module' when reinitializing the module during frame processing.	Driver layer	CVE-2024-56853
7	ALFA AWUS036ACH	A Buffer Overflow occurs in function 'rtw_wlan_util' caused by improperly validating length fields of frames management.	Driver layer	CVE-2024-56854
8	ALFA AWUS036ACH	A Buffer Overflow occurs in function 'hal_com_phycfg' when rapid channel-switch during frame aggregation and reception.	Firmware layer	Bug-1198
9	ASUS AX56AX	A Stack Overflow occurs in function 'phl_connect' when processing nested connection retries with repeated dis/association.	Driver layer	CVE-2024-56855
10	ASUS AX56AX	A Buffer Overflow occurs in function 'phl_watchdog' when handling excessive frame status reports without size validation.	Driver layer	CVE-2024-53456
11	ASUS AX56AX	A Buffer Overflow occurs in function 'phl_btc_fsm' without boundary checks when handling unexpected control events.	Driver layer	Bug-2460
12	ASUS AX56AX	A NULL Pointer Dereference occurs in function 'hal_usb' caused by handling USB transfer callbacks during frame reception.	Firmware layer	Bug-2461
13	ASUS AX56AX	A Use-After-Free occurs in function 'hal_efuse' caused by accessing freed memory triggered by dynamic config updates.	Firmware layer	Bug-2464
14	EDUP 8774M	A NULL Pointer Dereference occurs in function 'mlmeext_disconnect' when disconnect triggered with incomplete association.	Driver layer	CVE-2024-53448
15	EDUP 8774M	A Use-After-Free occurs in function 'config_channel_plan' caused by concurrent scan and channel-switch Wi-Fi frames.	Driver layer	Bug-140
16	Netcore NW336	An aggregation vulnerability due to improperly validated A-MPDU frames, leading to buffer corruption.	MAC layer	Bug-1197
17	Netcore NW336	A Use-After-Free occurs in function 'ioctl_cfg80211' when processing async frame while a previous op is being deleted.	Driver layer	Bug-1198
18	GRiS LW04-3E03	A Buffer Overflow occurs in function 'rtw_wlan_util' caused by constructing management frames with incorrect IE lengths.	Driver layer	CVE-2024-53449
19	GRiS LW04-3E03	A Stack Overflow occurs in function 'phydm_phystatus' caused by recursive processing of malformed PHY status reports.	Firmware layer	Bug-125
20	NETGD AC1900	A NULL Pointer Dereference occurs in function 'wireless_sme' when state transitions triggered by unexpected disassociation.	Driver layer	Bug-1017
21	NETGD AC1900	A Buffer Overflow occurs in function 'driver_common' caused by copying unvalidated frame payloads with reception events.	Driver layer	Bug-1018

layer. Similarly, the firmware fuzzer μAFL uncovered just 3 vulnerabilities, leaving 3 additional firmware-level bugs undetected. This is because they do not consider the runtime interaction logic between frame transmission and reception, which is essential for thoroughly testing Wi-Fi NIC behavior. In contrast, DualFuzz performs dual-directional fuzzing by leveraging the TRModel and a latency-guided algorithm, enabling the generation of high-quality test inputs that explore transmission-reception interaction logic. As a result, DualFuzz successfully uncovered all 21 vulnerabilities, demonstrating its effectiveness and addressing **RQ1**. Compared with other state-of-the-art Wi-Fi Fuzzers, DualFuzz found all the vulnerabilities that other fuzzing tools found.

TABLE III

BUGS FOUND BY EACH TESTING TOOLS. OTHERS DETECT NO MORE THAN 11 BUGS, WHILE DUALFUZZ DETECTS ALL 21 BUGS.

Tool	Bug Number	Bugs ID#
DualFuzz	21	#1 - #21
OwFuzz-AP	4	#5,11,18,21
OwFuzz-STA	7	#4,7,12,15,16,19,20
wpaspy	8	#4,7,11,12,15,16,19,20
Greyhound	6	#5,7,11,12,18,20
VIRTFUZZ	6	#4,7,11,15,18,20
μAFL	3	#5,12,19

1) Case Study: We present a case study to demonstrate how vulnerabilities uncovered by DualFuzz in Wi-Fi NICs can threaten wireless network security, and how DualFuzz effectively detects these vulnerabilities. This case corresponds to bug #1 listed in Table II. This Use-After-Free vulnerability in the Intel AX210 driver layer causes Wi-Fi NIC crashes or potential data leakage due to incorrect buffer release after an ACK timeout. It results in access to invalid or reused memory, allowing attackers to crash the NIC or leak other users' data. Figure 10 illustrates the key triggering steps.

To trigger this bug, four key transmission-reception interactions are required: (1) A data frame is sent; the NIC stores it in the buffer pool under identifier fid and waits for an ACK. (2) The ACK times out, and the NIC mistakenly frees the buffer, leaving fid dangling. (3) A large data frame is sent while the buffer pool happens to be full at this time, triggering reallocation. (4) The delayed ACK arrives, and the NIC accesses fid, now pointing to invalid or reused memory, triggering a Use-After-Free. If fid points to an invalid address, the NIC crashes; if it points to reused memory, data leakage may occur. An attacker could exploit this by sending large data frames to force reallocation and delaying ACKs to trigger the bug. Developers have fixed this by nullifying fid after freeing and adding null checks before future accesses.

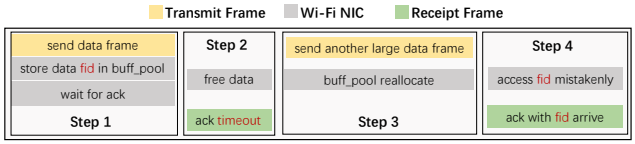

Fig. 10. An Use-After-Free vulnerability that causes NIC crash or privacy leakage in buff_pool management within the driver layer of Intel AX210.

In our experiments, this Use-After-Free vulnerability was only detected by DualFuzz. It is challenging to detect due to the complex sequence of interactions required between transmission and reception: 'transmit data frame→receipt ACK timeout→transmit other large data (triggering reallocation mechanism)→receipt ACK arrives→bug triggered.' Triggering such a deep vulnerability requires precise execution dependencies. Specifically, the buffer pool happens to be full after the receipt ACK timeout, causing reallocation, and at this moment, the delayed receipt ACK happens to arrive. Existing Wi-Fi fuzzers miss this bug due to their inability to explore intricate transmission-reception interaction logic. In contrast, DualFuzz leverages a TRModel capturing key interaction behaviors (e.g., ACK timeout, flow control, buffer management) and applies latency-guided fuzzing to mutate both transmission and reception frames that operate on shared data like ACK states and buffer pools. This enables DualFuzz to generate large data frames to trigger buffer reallocation while delaying ACKs, increasing the likelihood of exposing the vulnerability.

C. Evaluating TRModel and Latency Guidance

To evaluate the effectiveness of the TRModel and the latency guided fuzzing algorithm, we conducted an experiment comparing three variants: (1) DualFuzz, the full version with both the TRModel and latency guided fuzzing enabled; (2) DualFuzz$_{m-}$, a baseline version that disables the TRModel and randomly generates all transmit and receive Wi-Fi frames without considering interaction logic; and (3) DualFuzz$_{f-}$, a variant that retains the TRModel but disables the latency guided fuzzing strategy, instead randomly mutating the Wi-Fi frames identified by the TRModel as relevant to transmission-reception interactions. We measured the number of vulnerabilities detected by each variant over a 24-hour testing period on the eight target Wi-Fi NICs. In addition, we recorded the detection time for each identified bug.

TABLE IV
COMPARISON BETWEEN DUALFUZZ$_{m-}$, DUALFUZZ$_{f-}$, AND DUALFUZZ.

Metric	DualFuzz$_{m-}$	DualFuzz$_{f-}$	DualFuzz
Bugs Found	12	15	21 (↑75%, ↑40%)
Avg Bug Detection Time (Min)	436	359	271 (↑62%, ↑32%)

As shown in Table IV, with the support of the TRModel and latency guided fuzzing, DualFuzz successfully detects all 21 vulnerabilities within 24 hours. In contrast, DualFuzz$_{m-}$ detects only 12 vulnerabilities, while DualFuzz$_{f-}$ detects 15. The remaining 6 vulnerabilities (e.g., Bug #1 and #3, detailed in the case study) require specific execution dependencies between transmission and reception, which are difficult to uncover through random testing methods. In addition, as shown in Figure 11, DualFuzz demonstrates faster bug detection. On average, its vulnerability discovery speed is 32% faster than DualFuzz$_{f-}$ and 62% faster than DualFuzz$_{m-}$. These results demonstrate the effectiveness of both components: compared to DualFuzz$_{m-}$, DualFuzz shows that the TRModel enhances both vulnerability detection count and detection speed. Similarly, compared to DualFuzz$_{f-}$, the latency guided fuzzing strategy effectively improves these two metrics. These statistics provide a comprehensive answer to **RQ2**.

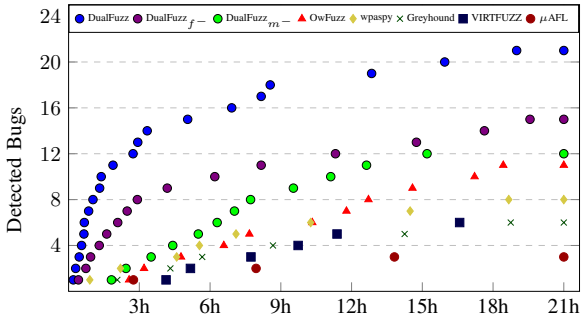

Fig. 11. Number of bugs detected by DualFuzz and other SOTA tools over time. DualFuzz consistently outperforms others with faster detection speed.

Analysis correlation between latency and bug detection: To evaluate whether higher latency correlates with bugs in complex interaction logic, we recorded the processing latency of each TX-RX frame pair and whether it triggered a bug. The results show that frame pairs triggering bugs had significantly higher latency (around 150–200ms) compared to non-bug-triggering pairs (around 10–20ms), suggesting that latency spikes are more likely to indicate activation of deep interaction paths where bugs may reside.

D. Accuracy of Anomaly Detector in DualFuzz

To evaluate the accuracy of DualFuzz and assess potential **false negatives** and **false positives** that may be introduced by the bug deduplication process, we first collected all anomalies reported by the liveness and equivalence detectors, along with their corresponding reproduction sequences during the testing phase. We then manually analyzed them to identify true positives and false positives.

TABLE V
ACCURACY OF DUALFUZZ'S ANOMALY DETECTOR.

	Before Deduplicate		After Deduplicate	
	Liveness	Equivalence	Liveness	Equivalence
Reported Number	24	31	9	12
True Positive	9	12	9	12
False Positive	2	0	0	0

Before bug deduplication, DualFuzz reported a total of 55 anomalies, including 25 from the Liveness Detector and 31 from the Equivalence Detector. Among them, two were false positives, caused by test frames with extended timeouts that inadvertently triggered the Wi-Fi NIC's power-saving mode, leading to temporary unresponsiveness and a misidentification as offline. Thanks to DualFuzz's bug reproduction and deduplication process, these false positives were eliminated. After deduplication, the Liveness and Equivalence Detectors reported 9 and 12 unique bugs, respectively, all of which were true positives, with no new false negatives introduced. The false positives is reduced to zero. Thus, the accuracy of DualFuzz is satisfying, which adequately answers **RQ3**.

E. Effectiveness in Code Coverage

TABLE VI
BRANCH COVERAGE ON THE NIC DRIVERS IN 24 HOURS. DUALFUZZ CONSISTENTLY COVERS MORE BRANCHES COMPARED TO SOTA TOOLS.

Tool	Tp-Link AX1800	ALFA AWUS036ACH	ASUS AX56AX	EDUP 8774M
OwFuzz-AP	11958	10243	10466	11061
OwFuzz-STA	12634	10782	10948	11821
wpaspy	11516	10121	10423	11271
Greyhound	11865	10487	10299	11062
VIRTFUZZ	12853	11093	11490	12074
DualFuzz	**15722 ↑ 29%**	**12991 ↑ 23%**	**13814 ↑ 25%**	**14292↑ 25%**

Given that most Wi-Fi NICs are closed-source and only a subset of NIC drivers are open-source, to evaluate the effectiveness of DualFuzz in terms of code coverage on Wi-Fi NICs, we instrumented the open-source drivers using gcov [18] and collected branch coverage data over a 24-hour fuzzing period. The statistics are shown in Table VI. In conclusion, DualFuzz consistently outperforms all other state-of-the-art tools across the four open-source NIC drivers. Compared to OwFuzz-AP, OwFuzz-STA, wpaspy, Greyhound, and VIRTFUZZ, DualFuzz achieves an average branch coverage improvement of 23% - 29%. These results provide a clear and comprehensive answer to **RQ4**.

VII. Discussion

Extend DualFuzz to other network middleware. Currently, DualFuzz proposes a dual-directional fuzzing framework for Wi-Fi NICs to test transmission-reception interactions, successfully uncovering 21 new vulnerabilities. In fact, this dual-directional fuzzing framework can also be extended to test other network middlewares, such as the broker in MQTT [39], the resource handler in CoAP [46], or the message broker in AMQP [3], which serve as intermediaries that manage and route communication between various devices, involve rich bidirectional interactions.

For example, in the case of the MQTT protocol, DualFuzz can follow a similar process: (1) It first performs static analysis on the MQTT protocol's RFC 6455/ISO standard [16], a specification that defines the broker-client communication behavior, to automatically extract shared data (e.g., session flags, message identifiers, QoS levels) and build a corresponding TRModel; (2) Based on the TRModel, it coordinates the mutation of transmission and reception packets (e.g., PUBLISH, SUBSCRIBE, ACK) to continuously test the broker's internal interaction logic and uncover latent vulnerabilities. However, unlike Wi-Fi NICs, many middlewares are open-source. We plan to extend DualFuzz's guidance and detector to support gray-box or white-box modes in the future.

Enhance TRModel with implicit dependencies and extend it to proprietary NICs. Currently, TRModel captures many shared data dependencies between transmission and reception frames, but may miss implicit or non-obvious dependencies that are difficult to extract automatically from IEEE 802.11 standard. Besides, some Wi-Fi NICs introduce proprietary logic, such as custom frame types, vendor-specific retransmission, etc., which fall outside the standard and are not modeled in TRModel, potentially causing certain implementation-specific bugs to be missed. We can enhance the TRModel by incorporating dynamic analysis techniques to infer implicit dependencies or vendor-specific interaction logic by monitoring whether certain variables or memory regions are accessed during both transmission and reception. However, this requires fine-grained instrumentation and analysis, which introduces additional overhead. How to find a balance point needs to be explored in the future.

Longer Frame Sequence Exploration. At present, DualFuzz generates only one TX-RX frame pair per test input. Incorporating higher-level function semantics and modeling longer frame sequences could enhance state awareness and enable deeper exploration of the state space, as demonstrated by prior protocol fuzzing studies (e.g., Bleem [34], MPFuzz [33], SPFuzz [60]). However, modeling long sequences leads to exponential growth in the state space. Designing an efficient approach that balances modeling granularity and fuzzing efficiency is a promising direction for future work.

VIII. Related Work

Wi-Fi Fuzzing: Wi-Fi fuzzing has been widely applied in identifying vulnerabilities across various wireless devices. Wdev-Fuzzer [36] targets low-level driver code by injecting malformed packets, exposing critical flaws in proprietary Wi-Fi drivers. Greyhound [17] is a directed greybox fuzzer for Wi-Fi STA devices that identifies vulnerabilities by exploring critical protocol paths. Wpaspy [50] fuzzes various Wi-Fi protocols to detect bugs in firmware and drivers. OWFuzz [9] uses over-the-air fuzzing to mutate and inject 802.11 management/control frames, uncovering flaws in malformed frame handling. However, existing Wi-Fi fuzzers separately test the transmission and reception, neglecting their interaction dependencies, thus missing certain vulnerabilities.

Driver Fuzzing: Several works focus on Linux driver fuzzing and can detect bugs in the Wi-Fi NIC driver layer. Syzbot [21] uses Syzkaller with low-level syscall inputs to uncover memory and concurrency issues. Tools like USBFuzz [44], Nyx [51], SATURN [59], and FuzzUSB [29] emulate device behavior to reveal deep system bugs, while DEVFuzz [58] uses device models to guide syscall generation. VIRTFUZZ [26] targets the 802.11 stack via VirtIO. In contrast, DualFuzz operates at the protocol level, crafting high-level Wi-Fi frames to expose vulnerabilities with broader attack surfaces. Moreover, these driver fuzzers overlook the interaction logic between transmission and reception, as well as vulnerabilities hidden in other layers of Wi-Fi NICs.

Firmware Fuzzing: Some works target firmware fuzzing [61] and may apply to the firmware layer of Wi-Fi NICs. Firmadyne [11] emulates embedded Linux firmware for dynamic analysis. P2IM [15] infers peripheral models to uncover bugs in embedded systems. Fuzzware [48] enhances firmware fuzzing by using precise Memory-Mapped I/O modeling, allowing effective and accurate exploration of embedded firmware behavior. μAFL [31] uses ARM's ETM for feedback-based MCU fuzzing. However, these methods focus solely on firmware and overlook vulnerabilities in complex cross-layer transmission-reception interaction, which DualFuzz is designed to expose.

IX. Conclusion

In this paper, we propose DualFuzz, a dual-directional fuzzing framework designed to simultaneously test both transmission and reception logic within Wi-Fi NICs. First, we develop a TRModel to describe transmission-reception interactions. Second, DualFuzz employs a latency-guided fuzzing strategy to efficiently coordinate and explore the generation of transmission and reception test frames. Finally, we introduce two real-time anomaly detectors to accurately identify abnormal behaviors and uncover vulnerabilities. We implemented DualFuzz and evaluated it on eight widely used Wi-Fi NICs, successfully discovering 21 new vulnerabilities. Our future work will enhance DualFuzz by adding dynamic modeling to the TRModel and adapting DualFuzz to other network middleware testing scenarios.

X. Acknowledgements

This research is sponsored in part by the National Key Research and Development Project (No. 2022YFB3104000), and NSFC Program (No. U2441238, 62021002).

References

[1] Ramia Babiker Mohammed Abdelrahman, Amin Babiker A Mustafa, and Ashraf A Osman. A comparison between ieee 802.11 a, b, g, n and ac standards. *IOSR Journal of Computer Engineering (IOSR-JEC)*, 17(5):26–29, 2015.

[2] Muhammad Raisul Alam, Mamun Bin Ibne Reaz, and Mohd Alauddin Mohd Ali. A review of smart homes—past, present, and future. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 42(6):1190–1203, 2012.

[3] AMQP. Advanced message queuing protocol. https://www.amqp.org/, 2025. Accessed at March 29, 2025.

[4] PM Anderson and AA Fouad. Institute of electrical and electronics engineers. *Power system control and stability*, 2003.

[5] Johannes Berg. Radiotap: De facto standard for 802.11 frame injection and reception, 2022.

[6] Philippe Biondi. Scapy documentation (!). *vol*, 469:155–203, 2010.

[7] Nader Bouacida, Ahmad Showail, and Basem Shihada. Buffer management in wireless full-duplex systems. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 557–564. IEEE, 2015.

[8] Broadcom. Bcm6755: Dual 2x2 802.11ax wi-fi 6 quad-core arm soc. https://www.broadcom.com/products/wireless/wireless-lan-infrastructure/bcm6755, 2025. Accessed at March 29, 2025.

[9] Hongjian Cao, Lin Huang, Shuwei Hu, Shangcheng Shi, and Yujia Liu. 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*, pages 263–273, 2023.

[10] M Čermák, Š Svorenčík, R Lipovskỳ, and O Kubovič. Kr00k-serious vulnerability deep inside your wi-fi encryption. *ESET White paper*, 2020.

[11] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS*, volume 1, pages 1–1, 2016.

[12] National Vulnerability Database. Cve-2019-15126 detail. https://nvd.nist.gov/vuln/detail/CVE-2019-15126, 2023. Accessed on November 11, 2024.

[13] NATIONAL VULNERABILITY DATABASE. Cve-2022-21745 detail. https://nvd.nist.gov/vuln/detail/CVE-2022-21745, 2025. Accessed at March 29, 2025.

[14] Garlisi Domenico, Giovanni Garbo, and Ilenia Tinnirello. Design, implementation and experimental evaluation of a wireless mac processor over commercial wifi cards. *Relatório técnico*, 2014.

[15] Bo Feng, Alejandro Mera, and Long Lu. {P2IM}: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1237–1254, 2020.

[16] Ian Fette and Alexey Melnikov. Rfc 6455: The websocket protocol, 2011.

[17] Matheus E Garbelini, Chundong Wang, and Sudipta Chattopadhyay. Greyhound: Directed greybox wi-fi fuzzing. *IEEE Transactions on Dependable and Secure Computing*, 19(2):817–834, 2020.

[18] GCOV. 11 gcov—a test coverage program. https://gcc.gnu.org/onlinedocs/gcc/Gcov.html, 2025. Accessed at March 29, 2025.

[19] Yao Ge, Ahmad Taha, Syed Aziz Shah, Kia Dashtipour, Shuyuan Zhu, Jonathan Cooper, Qammer H Abbasi, and Muhammad Ali Imran. Contactless wifi sensing and monitoring for future healthcare-emerging trends, challenges, and opportunities. *IEEE Reviews in Biomedical Engineering*, 16:171–191, 2022.

[20] Carles Gomez, Stefano Chessa, Anthony Fleury, George Roussos, and Davy Preuveneers. Internet of things for enabling smart environments: A technology-centric perspective. *Journal of Ambient Intelligence and Smart Environments*, 11(1):23–43, 2019.

[21] Google. syzbot. https://syzkaller.appspot.com/upstream, 2025. Accessed at March 29, 2025.

[22] Yohei Hasegawa and Kazuya Suzuki. A multi-user ack-aggregation method for large-scale reliable lorawan service. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[23] Johan Henning. Pentesting on a wifi adapter: Afirmware and driver security analysis of a wifi adapter, with a subset of wifi pentesting, 2023.

[24] Guido R Hiertz, Dee Denteneer, Lothar Stibor, Yunpeng Zang, Xavier Pérez Costa, and Bernhard Walke. The ieee 802.11 universe. *IEEE Communications Magazine*, 48(1):62–70, 2010.

[25] Toke Høiland-Jørgensen, Per Hurtig, and Anna Brunstrom. The good, the bad and the wifi: Modern aqms in a residential setting. *Computer Networks*, 89:90–106, 2015.

[26] Sönke Huster, Matthias Hollick, and Jiska Classen. To boldly go where no fuzzer has gone before: Finding bugs in linux'wireless stacks through virtio devices. In *2024 IEEE Symposium on Security and Privacy (SP). Los Alamitos, CA, USA: IEEE Computer Society*, pages 24–24, 2024.

[27] Oliver C Ibe. *Fundamentals of data communication networks*. John Wiley & Sons, 2017.

[28] Intel. Intel wi-fi 6e ax210. https://www.intel.com/content/www/us/en/products/sku/204836/intel-wifi-6e-ax210-gig/specifications.html, 2025. Accessed at March 29, 2025.

[29] Kyungtae Kim, Taegyu Kim, Ertza Warraich, Byoungyoung Lee, Kevin RB Butler, Antonio Bianchi, and Dave Jing Tian. Fuzzusb: Hybrid stateful fuzzing of usb gadget stacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2212–2229. IEEE, 2022.

[30] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Revisiting acknowledgment mechanism for transport control: Modeling, analysis, and implementation. *IEEE/ACM Transactions on Networking*, 29(6):2678–2692, 2021.

[31] Wenqiang Li, Jiameng Shi, Fengjun Li, Jingqiang Lin, Wei Wang, and Le Guan. μafl: non-intrusive feedback-driven fuzzing for microcontroller firmware. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1–12, 2022.

[32] American Fuzzy Lop. Performance tips - afl (american fuzzy lop) - keep your test cases small. https://afl-1.readthedocs.io/en/latest/tips.html, 2025. Accessed at March 29, 2025.

[33] Zhengxiong Luo, Junze Yu, Qingpeng Du, Yanyang Zhao, Feifan Wu, Heyuan Shi, Wanli Chang, and Yu Jiang. Parallel fuzzing of iot messaging protocols through collaborative packet generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3431–3442, 2024.

[34] Zhengxiong Luo, Junze Yu, Feilong Zuo, Jianzhong Liu, Yu Jiang, Ting Chen, Abhik Roychoudhury, and Jiaguang Sun. Bleem: Packet sequence oriented fuzzing for protocol implementations. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4481–4498, 2023.

[35] MediaTek. Mt7921au product description. https://www.edaltech.com/products/mediatek/mt7921au.html, 2025. Accessed at March 29, 2025.

[36] Manuel Mendonça and Nuno Neves. Fuzzing wi-fi drivers to locate security vulnerabilities. In *2008 Seventh European Dependable Computing Conference*, pages 110–119. IEEE, 2008.

[37] Andreas F Molisch. *Wireless communications*, volume 34. John Wiley & Sons, 2012.

[38] Erfan Mozaffariahrar, Fabrice Theoleyre, and Michael Menth. A survey of wi-fi 6: Technologies, advances, and challenges. *Future Internet*, 14(10):293, 2022.

[39] MQTT.ORG. Mqtt: The standard for iot messaging. https://mqtt.org/, 2025. Accessed at March 29, 2025.

[40] Rohan Murty, Jitendra Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill. Designing high performance enterprise wi-fi networks. In *NSDI*, volume 8, pages 73–88, 2008.

[41] Haitham Ameen Noman and Osama MF Abu-Sharkh. Code injection attacks in wireless-based internet of things (iot): A comprehensive review and practical implementations. *Sensors*, 23(13):6067, 2023.

[42] Luiz Oliveira, Joel JPC Rodrigues, Sergei A Kozlov, Ricardo AL Rabêlo, and Victor Hugo C de Albuquerque. Mac layer protocols for internet of things: A survey. *Future Internet*, 11(1):16, 2019.

[43] Hassan Aboubakr Omar, Khadige Abboud, Nan Cheng, Kamal Rahimi Malekshan, Amila Tharaperiya Gamage, and Weihua Zhuang. A survey on high efficiency wireless local area networks: Next generation wifi. *IEEE Communications Surveys & Tutorials*, 18(4):2315–2344, 2016.

[44] Hui Peng and Mathias Payer. {USBFuzz}: A framework for fuzzing {USB} drivers by device emulation. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2559–2575, 2020.

[45] Norbert Pohlmann, Helmut Reimer, Wolfgang Schneider, Sami Petäjäsoja, Ari Takanen, Mikko Varpiola, and Heikki Kortti. Case studies from fuzzing bluetooth, wifi and wimax. In *ISSE/SECURE 2007 Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe/SECURE 2007 Conference*, pages 188–195. Springer, 2007.

[46] The Constrained Application Protocol. Coap, rfc 7252 constrained application protocol. https://coap.space/, 2025. Accessed at March 29, 2025.

[47] Realtek. Rtl8812au 802.11ac/abgn usb wlan network controller. https://www.realtek.com/Product/Index?id=579&cate_id=194, 2025. Accessed at March 29, 2025.

[48] Tobias Scharnowski, Nils Bars, Moritz Schloegel, Eric Gustafson, Marius Muench, Giovanni Vigna, Christopher Kruegel, Thorsten Holz, and Ali Abbasi. Fuzzware: Using precise {MMIO} modeling for effective firmware fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1239–1256, 2022.

[49] G Scheible, J Schutz, and C Apneseth. Novel wireless power supply system for wireless communication devices in industrial automation systems. In *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02*, volume 2, pages 1358–1363. IEEE, 2002.

[50] Domien Schepers, Mathy Vanhoef, and Aanjhan Ranganathan. 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*, pages 368–370, 2021.

[51] Sergej Schumilo, Cornelius Aschermann, Ali Abbasi, Simon Wörner, and Thorsten Holz. Nyx: Greybox hypervisor fuzzing using fast snapshots and affine types. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2597–2614, 2021.

[52] Sunil Kr Singh, Ajay Kumar, Siddharth Gupta, and Ratnakar Madan. Architectural performance of wimax over wifi with reliable qos over wireless communication. *International Journal of Advanced Networking and Applications*, 3(1):1017, 2011.

[53] Dionysios Skordoulis, Qiang Ni, Hsiao-Hwa Chen, Adrian P Stephens, Changwen Liu, and Abbas Jamalipour. Ieee 802.11 n mac frame aggregation mechanisms for next-generation high-throughput wlans. *IEEE Wireless Communications*, 15(1):40–47, 2008.

[54] Saber Talari, Miadreza Shafie-Khah, Pierluigi Siano, Vincenzo Loia, Aurelio Tommasetti, and João PS Catalão. A review of smart cities based on the internet of things concept. *Energies*, 10(4):421, 2017.

[55] Manesh Thankappan, Helena Rifà-Pous, and Carles Garrigues. Multi-channel man-in-the-middle attacks against protected wi-fi networks: A state of the art review. *Expert Systems with Applications*, 210:118401, 2022.

[56] Yuli Vasiliev. *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020.

[57] Prasaja Wikanta. *Study and design of a new PHY/MAC Cross-Layer architecture for Wireless Sensor Networks Dedicated to Healthcare*. PhD thesis, Université Polytechnique Hauts-de-France, 2021.

[58] Yilun Wu, Tong Zhang, Changhee Jung, and Dongyoon Lee. Devfuzz: automatic device model-guided device driver fuzzing. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 3246–3261. IEEE, 2023.

[59] Yiru Xu, Hao Sun, Jianzhong Liu, Yuheng Shen, and Yu Jiang. Saturn: Host-gadget synergistic usb driver fuzzing. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4646–4660. IEEE, 2024.

[60] Junze Yu, Zhengxiong Luo, Fangshangyuan Xia, Yanyang Zhao, Heyuan Shi, and Yu Jiang. Spfuzz: Stateful path based parallel fuzzing for protocols in autonomous vehicles. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[61] Chi Zhang, Yu Wang, and Linzhang Wang. Firmware fuzzing: The state of the art. In *Proceedings of the 12th Asia-Pacific Symposium on Internetware*, pages 110–115, 2020.

[62] Meng Zheng, Yongheng Zhao, Huaguang Shi, and Wei Liang. A flexible retransmission scheme for reliable and real-time transmissions in industrial wireless networks for factory automation. *IEEE Transactions on Vehicular Technology*, 72(8):10867–10878, 2023.