5GHOUL: Unleashing Chaos on 5G Edge Devices via Stateful Multi-layer Fuzzing

Matheus E. Garbelini, Zewen Shang, Shijie Luo, Sudipta Chattopadhyay, Sumei Sun, Ernest Kurniawan

***Abstract*—In this paper, we present 5GHOUL, a framework to systematically discover and replicate security vulnerabilities on arbitrary 5G edge devices (UE). At the core of 5GHOUL is a stateful fuzzing strategy that provides full control to arbitrarily manipulate any packet down to the data link layer. Moreover, 5GHOUL automatically constructs the protocol state machines to guide the fuzzing process and employs novel strategies to reliably exploit vulnerabilities on commercial-off-the-shelf (COTS) UEs over-the-air. The design choices in 5GHOUL were carefully taken to allow packet manipulation in real-time, which, in turn allowed us to fuzz down to data link layer. As of today, we have evaluated 5GHOUL with seven COTS 5G UEs (smartphones and USB modems) and one open source framework (OpenAirInterface). 5GHOUL has uncovered 12 unknown security vulnerabilities (14 in total) out of which ten exist in COTS UEs (ten CVEs assigned) from major vendors (e.g., Qualcomm and MediaTek). Moreover, six of these COTS UE vulnerabilities have been confirmed to have high severity. We also won a bug bounty of over 20K USD from Qualcomm and MediaTek for discovering these vulnerabilities. We envision 5GHOUL to open the door for 5G security testing at scale.**

***Index Terms*—Wireless Fuzzing, 5G NR, Mobile Security.**

I. INTRODUCTION

Mobile communications are indispensable in our daily lives, and the advent of 5G networks has brought many new opportunities for low-latency communication in critical domains like internet-of-things (IoT), virtual reality, and medical and automation industries. However, the potential vulnerabilities in 5G networks [1] undermine trust in their security, necessitating the validation of 5G protocol stacks prior to deployment. As commercial 5G stacks are typically closed source, verification and static analysis are not feasible. Therefore, there is a need for technologies to automatically interact with any off-the-shelf 5G device and systematically identify implementation vulnerabilities.

To address these challenges, this paper introduces 5GHOUL, a framework designed to discover and replicate vulnerabilities on arbitrary 5G devices over-the-air. 5GHOUL enables full control of packets down to the data-link layer, allowing manipulation before release to the target UE. The framework employs efficient strategies for fuzzing at the 5G MAC Layer (i.e., OSI Layer 2 - Data link), leveraging an abstract 5G protocol state machine constructed from communication traces. An evolutionary algorithm maximizes transition coverage during the fuzzing campaign. 5GHOUL also provides interfaces for controlling and monitoring the health of 5G smartphones and USB modems, making it flexible and applicable to any off-the-shelf 5G UE. Furthermore,

M.E. Garbelini, Z. Shang, S. Luo and S. Chattopadhyay are with Singapore University of Technology and Design. S. Sun and E. Kuniawan are with I^{2}R, A*Star, Singapore.

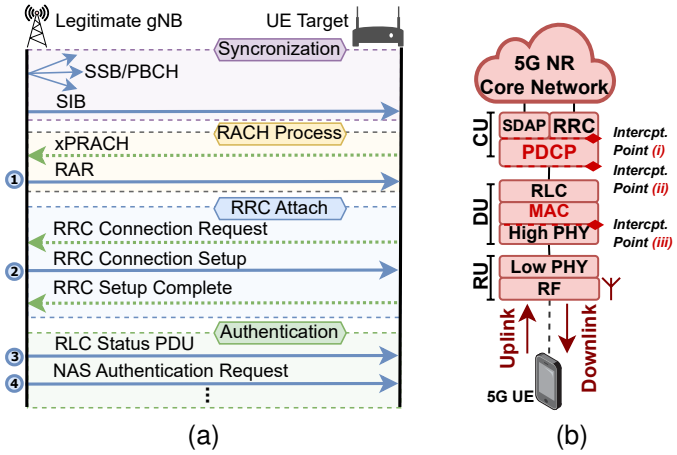


Fig. 1. **(a)** depicts the overall 5G NR communication procedures between gNB and UE. **(b)** showcases the layering of protocols at the gNB side with its interception points for downlink packet manipulation.

5GHOUL facilitates root cause identification of discovered vulnerabilities by proposing a methodology for efficient and reliable OTA exploitation. Such exploitation does not require any details of UE's SIM card and can target vulnerabilities down to the data link, making it practical to launch attacks in the wild. *Notably,* 5GHOUL *is the first stateful fuzzing approach to target all 5G NR protocols down to the data link layer.*

To control the real-time 5G communication for fuzzing, 5GHOUL gains access to contextual information, including security configurations, only available during live communication. The framework intercepts protocol packets at different components of the 5G base-station (*gNB*) and handles protocol-specific behaviors such as *integrity, encryption,* and *packet fragmentation* at different *interception points.* This allows full control over uplink and downlink communication. Although the evaluation primarily focuses on downlink fuzzing, 5GHOUL's implementation supports uplink monitoring and control. Figure 1b provides an overview of the interception points installed by 5GHOUL.

Little works have been performed to comprehensively fuzz 5G devices. Earlier works on 5G fuzzing or testing [1]–[3] did not target COTS UEs and only validated some fuzzing actions on software simulation [2], [3] or models [1]. Additionally, 5Greplay [2] was not designed for data link fuzzing. Although several works exist in cellular LTE fuzzing [4]–[9], they do not directly target 5G implementations. Moreover, none of these works are capable to control and manipulate data link packets. Finally, these approaches require significant manual effort to write test cases [6], may rely on

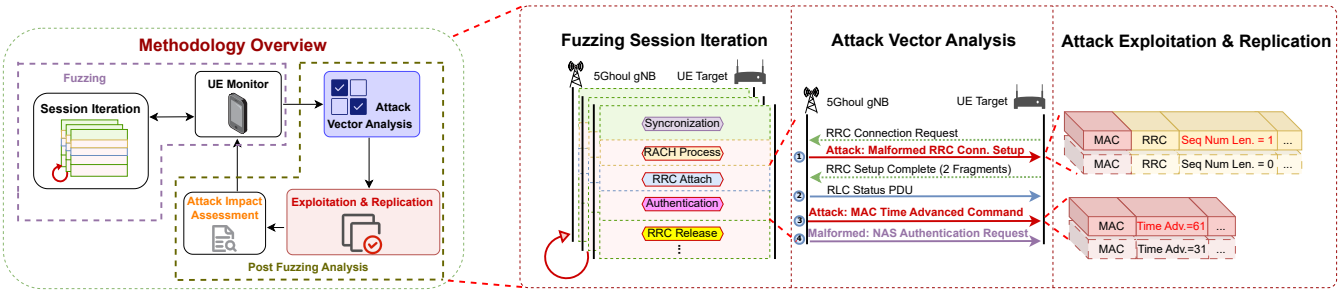


Fig. 2. The illustration of 5GHOUL approach. The left part shows the overall workflow of 5GHOUL. The right part illustrates an instance of the 5GHOUL approach for one of the discovered vulnerability ***V4*** (see Table III). For the `Malformed RRC Conn. Setup` and `MAC Time Advanced Command` packets, the exact values of the mutated fields (highlighted in red) and their original values are shown in the attack exploitation and replication.

commercial logs [5] or involve significant reverse engineering or emulation [7]–[9]. In particular, emulation requires additional effort to support new baseband architectures and introduce imprecisions during modem hardware emulation.

After providing the necessary background and outlining the threat model and overall workflow of 5GHOUL (Section II), we make the following contributions in the paper:

1) We present 5GHOUL– a framework based on stateful fuzzing to fully manipulate 5G protocols down to the data link layer for arbitrary COTS 5G UEs (Section III).
2) We show that 5GHOUL provides a generic and flexible approach for health monitoring of 5G UEs. This makes 5GHOUL usable out-of-the-box (Section III-B).
3) We present an efficient, yet reliable methodology to replicate any vulnerability and generate over-the-air exploits (Section III-C).
4) We discuss our implementation details (Section IV) and evaluate 5GHOUL with seven COTS UEs employing modems from MediaTek and Qualcomm as well as with open source OAI UE. We have found 12 new vulnerabilities (14 total), out of which 10 affect 5G modems from Qualcomm and MediaTek which are employed in over 710 smartphone models [1] currently in the market. Moreover, six of these ten vulnerabilities are confirmed to have high severity. Finally, *three COTS UE vulnerabilities found by* 5GHOUL *are in the 5G data link layer and thus, such vulnerabilities cannot be exposed by any existing approach (Section V).*
5) We compare 5GHOUL with the state-of-the-art 5G UE security test cases [10]. We show that none of the vulnerabilities found by 5GHOUL can be discovered using such test cases (Section V).
6) We show that 5GHOUL could be extended to automatically identify security issues beyond crashes and hangs. In particular, we show the extensibility of 5GHOUL to facilitate discovery and detection of downgrade attacks (Section V).
7) We launch concrete attacks exploiting the vulnerabilities discovered in our study on smartphones and Customer Premises Equipment (CPE). We discuss the impact of these attacks on users (Section VI-VII).

II. Background and Framework Overview

A. Background

5G cellular network architecture consists of three key components: The gNodeB (gNB), User Equipment (UE), and Core Network. The gNB is also known as the base station in traditional cellular network. It serves as the access point for wireless communication between the UE and the 5G core network. The UE refers to end-user devices, such as smartphones and tablets, that connect to the 5G network through the gNB. Lastly, the Core Network acts as the backbone of the 5G architecture by providing control and management functions, including authentication, security, mobility management, session establishment, and data routing between network entities.

Figure 1a illustrates a clean connection process between a 5G UE device (e.g., a smartphone or a 5G USB modem) and a legitimate gNB. Multiple protocols including Radio Resource Control (RRC), Non-Access Stratum (NAS), Medium Access Control (MAC), Packet Data Convergence Protocol (PDCP) and Radio Link Control (RLC) from both network layer (OSI layer 3) and data link layer (OSI layer 2) are involved to ensure that the connection is securely established.

The *synchronization* procedure encompasses both cell search and downlink synchronization, which are essential procedures for a UE to acquire time and frequency synchronization to correctly communicate with the gNB.

Following the *Synchronization* stage, *RACH Process* only requires MAC layer communication to establish uplink synchronization. To initiate the RACH procedure, the UE transmits a RACH preamble to the gNB. Upon receiving the RACH preamble, the gNB responds by sending a RACH Response (RAR) message to the UE. RAR includes crucial information e.g., the timing advance command and the Cell Radio Network Temporary Identifier (C-RNTI). *RRC Attach* process then establishes a connection between the UE and gNB by initiating a connection request from the UE (`RRC Connection Request`). gNB responds to the request via `RRC Connection Setup` message and upon completion of the setup, the UE responds back via `RRC Setup Complete` message. This process involves complex interaction between layer 3 protocol like RRC and data

[1]This is accomplished by scraping https://www.kimovil.com/en/.

link layer protocols e.g., MAC, PDCP and RLC. The PDCP protocol handles the compression and decompression of IP (Internet Protocol) packets, as well as provides encryption. Concurrently, the RLC protocol is responsible for the segmentation, reassembly, and error control of data packets transmitted over the radio link.

Finally, the *Authentication* stage establishes authentication between UE and the core network by exchanging NAS challenge-responses messages. Such negotiations have the objective to ensure secure communication for subsequent data exchanges between the gNB and the UE. The authentication stage involves interaction among all protocol layers.

B. Challenges of Fuzzing 5G and Technical Contribution

In this section, we first discuss the challenges with the design of fuzzing for COTS UEs that employ 5G software. Subsequently, we outline our strategies to address these challenges within 5GHOUL.

Challenges in Fuzzing COTS 5G UEs: In conventional greybox fuzzing approaches [24], feedback from the software under test (SUT) is leveraged to guide the fuzzing towards deeper parts of the SUT. This feedback is usually a coverage metric that indicates how much of the SUT implementation or SUT state space has been covered. While this type of feedback can be collected from open-source software or common OS programs, COTS UEs employ 5G software, which is strictly closed-source and does not expose any sort of code tracing (for coverage collection) to the user without making use of hardware-intrusive approaches (i.e., JTAG debugging). This makes conventional greybox fuzzing approaches inapplicable for fuzzing COTS UEs. Moreover, the protocol behavior of 5G (and 4G) is too complex to rely on traditional stateful fuzzers, which often rely on partially implementing their test generation [4], [5], [8]: For instance, generating tests by ignoring encryption, integrity or data-link protocols that require low-latency communication such as MAC and RLC. Handling such low-latency communication is crucial for end-to-end fuzzing of COTS UEs, as delays in the data-link (e.g., $>1ms$) can break the communication during fuzzing, making the test generation futile in practice.

5GHOUL Technical Contribution: To tackle the aforementioned issues, we adopt a stateful, yet *end-to-end* approach when fuzzing COTS UEs. This enables the following contributions over state-of-the-art, which includes 4G (See Section VIII for details): *(i)* As opposed to 4G/5G fuzzing tools that only support certain protocols, the end-to-end nature of 5GHOUL, enables discovery of bugs from the earliest to the latest interaction between COTS UEs and the network by targeting data-link and low-latency protocols such as MAC, RLC and PDCP as well as network protocols such as RRC and NAS. This enables a much wider spectrum of vulnerability discovery as compared to the state-of-the-art techniques. *(ii)* 5GHOUL state machine (i.e., automaton) is generic, easy to extend via a few rules and is only used to guide the fuzzing (i.e., lightweight) as opposed to directly using it for generating test inputs [15]. In this fashion, 5GHOUL employs stateful fuzzing of 5G UEs, yet avoids complex (and manual) implementation of 5G state machine to generate and communicate 5G packets. In contrast, prior techniques [1], [6], [14] rely on bespoke wireless state models that do not cover data-link states and are difficult to extend or adapt to other protocols. *(iii)* As opposed to software-only fuzzing techniques, over-the-air fuzzing embodied in 5GHOUL yields more realistic attack due to the practicality of triggering bugs once their attack vector is discovered. While software-only fuzzing is generally faster than over-the-air fuzzing, we argue that the design of 5GHOUL is well justified. This is because 5GHOUL applies to fuzzing arbitrary COTS UEs. Moreover, 5GHOUL improves reproducibility of bugs and triaging by employing a systematic process to discover the minimal set of modified packets (i.e., the attack vector) and generate an exploit code. To the best of our knowledge, we are not aware of any (stateful) 4G/5G fuzzing techniques that solve the dual problem of bug discovery and minimal exploit generation. *(iv)* Finally, since 5G communication often exhibits bursts of messages containing the same data-link protocol (MAC, RLC) alongside network messages with hundreds of fields, we employ an optimization and balancing strategy that helps to select which message and payload to mutate during the communication with COTS UEs, hence increasing the bug-finding effectiveness of 5GHOUL.

C. Threat Model

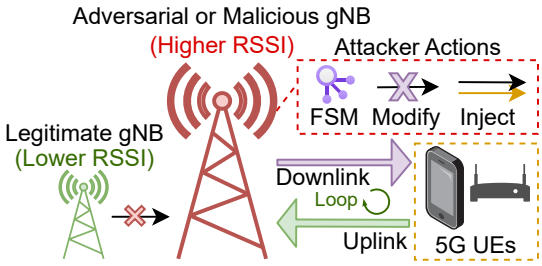


Fig. 3. Adversarial-Controlled gNB Attacker Model.

Attacker Model: As illustrated in Figure 3, 5GHOUL makes use of an attacker model which mimics a *limited* Dolev-Yao adversary. This is accomplished by exposing an *Adversary-Controlled Downlink channel* that can arbitrarily inject and/or modify 5G NR Downlink Packets generated from a finite-state-machine (FSM) of a real gNB implementation. Such an attacker model is used in previous works [1], [3], [8] and can freely manipulate downlink messages at the data-link layer (i.e., MAC, RLC, PDCP, RRC). This is because such messages are encoded in plain-text or partially encrypted (i.e., only part of the message is encrypted).

Practicality of the Attacker: In the aforementioned attacker model, no knowledge of the target UE is known to complete the *NAS network registration* (which requires full details of UE's SIM card). To overcome this limitation, the attacker can still easily exploit procedures before network registration by cloning the legitimate gNB and hijacking the UE connection via well known methods [11]. For instance, once the attacker is sufficiently close to the target UE

such that the *Received Signal Strength Indicator* (RSSI) of the adversarial gNB is higher than the legitimate gNB, the target UE will connect to the adversarial gNB. Then, the UE starts exchanging messages up to step 4 of Figure 1a. Procedures that appear later are subjected to failure since key information from UE's SIM card is unknown. However, throughout the message exchanges, the adversarial gNB can freely manipulate downlink messages to the target UE, opening a window to launch attacks at any step of the 5G NR procedures shown in Figure 1a.

It is worthwhile to mention that to deploy such adversarial model, an attacker only needs a mini-PC/Laptop (≈ 1000 USD) to run the rogue base station software, and an SDR such as the *USRP B210* (≈ 1600 *USD*) or XTRX (≈ 800 USD), the latter which can fit inside a laptop and hence appears visually stealthy to targets.

D. Challenges of Distributing 5G Patches Downstream

It is critical to make sure that the modem software development kit (SDK) is well tested and devoid of serious vulnerabilities before being released downstream. Otherwise, attackers may exploit a modem failure for a prolonged period and before the end user can actually pull the relevant security updates.

Figure 4 depicts the complexity of the software supply chain of a 5G modem. In summary, finding issues in the implementation of the 5G modem vendor heavily impacts product vendors downstream. This is because the software dependency of product vendors on the *Modem / Chipset Vendor* adds complexity and hence delays to the process of producing and distributing patches to the end-user. For example, each iteration that the upstream 5G modem software goes through, carrier recertification must be performed by the chipset vendor so that the updated modem firmware can be integrated into OS security patches by the smartphone OS vendor on a fixed release schedule (i.e., Google for Android and Apple for IOS). Next, such security patches ought to be manually built into the smartphone OS image by the *product vendor*. Therefore, it can often take six or more months (considering our own experience in this research) for 5G security patches to finally reach the end-user via an OTA update (final downstream node in Figure 4).

The chain of software dependencies are similarly applied to CPE routers or USB Modems. However, such type of products have a shorter time to distribute patches to the end-user since adhering to the release schedule of OS vendors is not required. Instead, the module vendor (i.e., OEM of Figure 4) directly builds modem patches into the module firmware and distribute it downstream via private channels. Therefore, the CPE product vendor can directly apply the patches from upstream to its platform software, which usually includes a customized Linux OS.

E. 5GHOUL *workflow*

Figure 2 outlines the 5GHOUL workflow. Broadly, 5GHOUL comprises of two steps: ① *Fuzzing* and ② *Post Fuzzing*

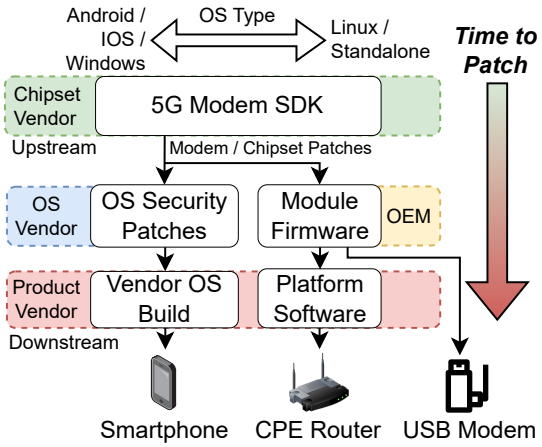


Fig. 4. 5G UE Software Supply Ecosystem

Analysis. 5GHOUL automatically constructs the 5G protocol state machine from a few benign communication traces. Then, this constructed state machine is leveraged during fuzzing sessions via an evolutionary search process, with the objective to maximize the transitions being covered (see Section III for details). To reliably detect crashes and monitor the health of the target device, 5GHOUL employs a generic approach that avoids reconfiguration depending on the target device. In particular, 5GHOUL fuzzer communicates via the Android Debug Bridge (ADB) for 5G enabled smartphones and via Qualcomm MSM Interface (*QMI*) for 5G USB modems, using generic commands to keep the design of the fuzzer easily applicable for the majority of 5G edge devices (see Section III-B for details). The outcome of 5GHOUL fuzzing is a set of communication traces with clearly identified security issues (e.g., crashes or hangs). These communication traces are subject to *post-fuzzing analysis* (see Figure 2).

Given a vulnerable communication trace, the objective of post-fuzzing analysis is to systematically identify the minimal set of modifications (e.g., minimal set of mutated or duplicated packets) by the fuzzer that results in the crash (see "*Attack Vector Analysis*" in Figure 2). After analyzing the attack vector, 5GHOUL creates a C++ script to exploit the vulnerability on an arbitrary 5G edge device (see "*Exploitation and Replication*" in Figure 2). We discuss the details of our post fuzzing analysis methodologies in Section III-C.

Running Example: Figure 2 illustrates an example of discovering and replicating vulnerability ***V4*** – `Invalid RRC Reconfiguration` (see Table III) using our 5GHOUL approach. After the UE target synchronizes with the malicious 5GHOUL gNB, all subsequent processes relevant to protocol layers MAC, RRC, NAS, RLC and PDCP can be fuzzed. Figure 2 highlights the MAC and RRC procedures that are subject to fuzzing. The 5GHOUL fuzzing session reveals a vulnerable communication trace containing several mutated packets as shown in Figure 2: *(i)* `Malformed RRC Conn. Setup`, *(ii)* `MAC Time Advanced Command` and *(iii)* `NAS Authentication Request`. After analyzing the attack vector, 5GHOUL determines that the same vulnerability can be reproduced even if the `NAS Authentication`

`Request` is not malformed. Indeed, during our evaluation, the attack vector of ***V4*** was computed to be the combination of just two messages, as highlighted by `Malformed RRC Conn. Setup` and `MAC Time Advanced Command`. Consequently, 5GHOUL facilitates development of an exploit script that intercept these messages and modifies the packet fields `Time Adv.` and `Seq. Num. Len.` in line with the vulnerable trace found during fuzzing, as shown in Figure 2. These modified packets are then released to the target UE to reliably reproduce the vulnerability ***V4***.

III. DESIGN OF 5GHOUL

Figure 5 illustrates the overall design of 5GHOUL.

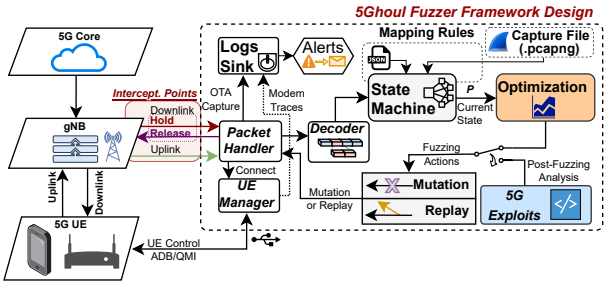


Fig. 5. 5GHOUL Fuzzing Framework for data link packet interception.

A. 5GHOUL *Fuzzer Design*

Latency Requirements: The *interception points* shown in Figure 5 *yield control over all packets down to the data link layer* that are produced at the gNB software stack. With such an approach, it is crucial to not block MAC frames for more than the time of the *Downlink* transmission slot. Otherwise, live packets cannot be delivered over-the-air. The *slot transmission* time (i.e., *numerology*) is configured at the gNB, and it can vary between $6.25us$ and $1ms$ [12]. Our interception strategy leverages shared memory to minimize the processing time between gNB and 5GHOUL.

Nevertheless, it is worth mentioning high data throughput is not relevant to the fuzzer. This is because our framework focuses in finding OTA bugs. Therefore, our 5GHOUL fuzzing architecture only considers gNB *transmission slot* configuration that are above $500us$ to ensure real-time communication. This is sufficient for fuzzing MAC or upper layer frames.

Interception Points: As previously shown in Figure 1, three *interception points* have been implemented in 5GHOUL: *(i)* After MAC, *(ii)* before RLC and *(iii)* at the PDCP layer. These are not only used to control MAC packets, but also packets that are yet-to-be fragmented by the RLC layer. More specifically, 5GHOUL intercepts before and after packets are encrypted. The rationale of these *interception points* is two-fold. Firstly, we ensure that any field manipulation will be received at the base station without being dropped due to encryption problems. Secondly, we aim to have complete control of fragmented packets.

1) **5GHOUL Fuzzer State Mapper:** The 5GHOUL state mapper provides real-time information about the protocol state of the gNB/UE, which is crucial for stateful fuzzing and monitoring protocol states during a fuzzing campaign. This information is used to dynamically update state transition coverage and guide the 5GHOUL fuzzer towards maximizing transition coverage.

Given the complexity of *3GPP* protocols, manual computation of 5G protocol state machines is impractical. Instead, 5GHOUL utilizes a lightweight method for learning the 5G protocol state machine. This process involves *(i:)* a set of Mapping Rules, *(ii:)* the capture traces (i.e., *pcap* file) as inputs. The Mapping Rules define how to identify a state for a specific protocol, while the capture traces serve as a reference for correct packet sequences during the learning phase. The resulting State Mapper outputs the reference sequence of states (i.e., the state machine M_{ref}) between the gNB and the UE. Once constructed offline in a one-time effort, M_{ref} is used to track the explored state during the fuzzing campaign and optimize the fuzzing process to increase protocol state coverage (using the “Optimization" component in Figure 5).

During packet interception at *interception point (iii)* (exemplified in Figure 6), each packet is parsed and assigned a unique state label (S). The state label is derived from packet information such as direction (***TX*** *for Downlink or* ***RX*** *for Uplink*), packet type, and the protocol layer name. The *Mapping Rules* M_u guide the State Mapper in obtaining the necessary information during packet decoding.

For example, consider the received RLC packet P, as illustrated in Figure 6. The mapping rule indicates that the packet type is contained within the packet field named `rlc.type`. Thus, once the packet P is received and decoded, the raw content of the `rlc.type` (i.e., `0x00`) is searched within a look up table and matches with a type string `Status PDU`. The state label for P can now be generated with the information about packet P's direction (`TX`), network layer (`RLC`) and the packet type (`Status PDU`). Thus, state label $S \equiv$ `TX` $\oplus$ `RLC` $\oplus$ `Status PDU`. In a similar fashion, the RRC packet P' was labeled. Figure 6 also illustrates the positioning of the generated state labels S and S' within the reference state machine M_{ref}.

The general steps to compute a state label S are described in procedure *state_mapping* of Algorithm 1. Specifically, after a packet P is intercepted and decoded, the process in Algorithm 1 searches through the mapping rules and first identifies the rule that matches the network layer L of packet P. Subsequently, the state mapping process locates the field f that contains the raw value of packet type. Finally, in Line 13, a lookup table converts the raw value of the identified packet field f, matched with rule `R.type_fields`, to a *type string*. Then state label S is generated by a string concatenation in Line 22.

2) **Mapping Rules** M_u**:** It consist of a set of rules for each relevant *3GPP* protocol layer (see Figure 7). As previously discussed, this set of rules are fed to the *State Mapper* to identify and label a state for an intercepted packet (see Figure 6). As illustrated in Figure 7, the property *"Filter"* filters

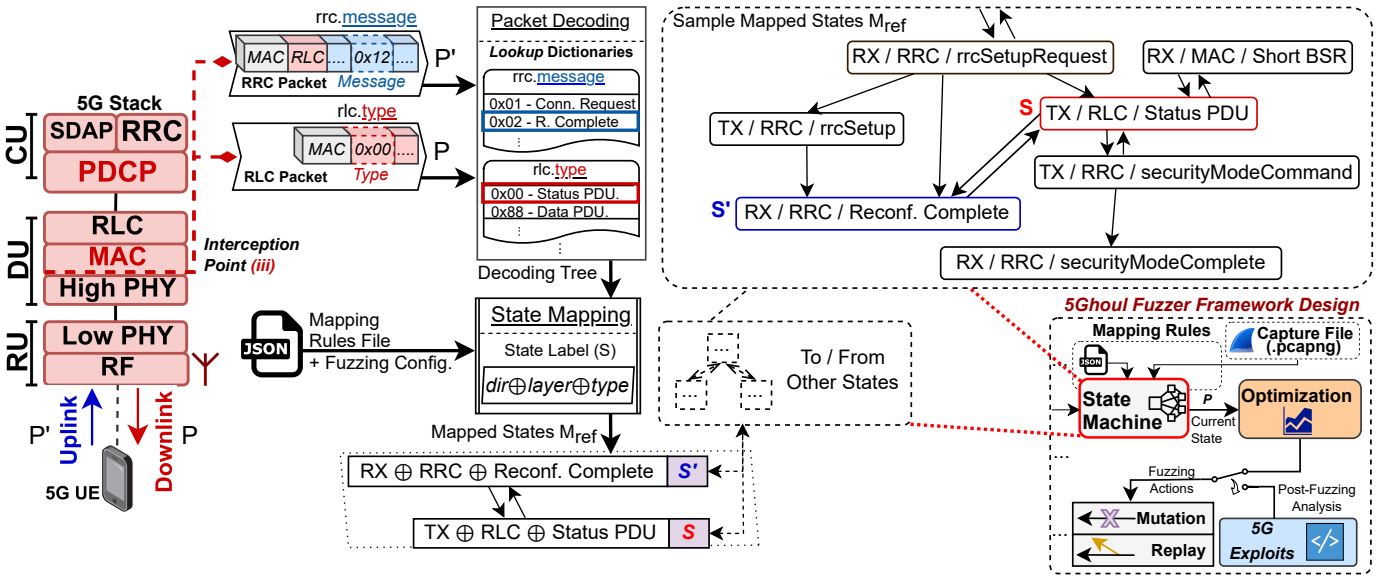


Fig. 6. An illustration of 5GHOUL state mapper. Packets P and P' are received via the interception point *(iii)*. Subsequently, P and P' are decoded and labeled with states S and S', respectively, using the mapping rules. Locations of S and S' within the reference state machine M_{ref} are also highlighted.

Algorithm 1 `state_mapping` Procedure

Input: Packet P, Mapping Rules M_u
Output: State label S generated for packet P

Decode packet P to get $P.layers$ and $P.fields$
for each $L \in P.layers$ **do**
  ▷ Check if the packet layer L match a rule in M_u
  for each $R \in M_u$ **do**
    if L satisfies $R.filter$ **then**
      ▷ Check if any field f is found within $R.type\_fields$
      for each $f \in P.fields$ **do**
        if $f \in R.type\_fields$ **then**
          v := value of field f in packet P
          $P.type$:= $lookup[v]$
          goto Line 20
        end if
      end for
    end if
  end for
  ▷ Label the state when matched with the rules
  if $(L \neq empty \wedge P.type \neq empty)$ **then**
    ▷ Create the state label S
    S := $P.dir \oplus L.name \oplus P.type$
    return S
  end if
end for

```
{
    "Filter": "nas-5gs",
    "StateNameField": [
        "nas_5gs.sm.message_type",
        "nas_5gs.mm.message_type"
    ]
},
{
    "Filter": "nr-rrc",
    "StateNameField": "nr-rrc.c1"
},
{
    "Filter": "rlc-nr",
    "StateNameField": [
        "rlc-nr.am.cpt",
        "rlc-nr.am.dc"
    ]
},
{
    "Filter": "mac-nr",
    "StateNameField": [
        "mac-nr.ulsch.lcid",
        "mac-nr.dlsch.lcid",
        "mac-nr.rnti-type"
    ]
}
```

Fig. 7. Mapping Rules for *MAC, RLC, RRC* and *NAS*. *PDCP* is not needed.

TABLE I
COMPLETENESS OF THE PROTOCOL STATE MACHINE OR SUPPORTED MOBILE PROCEDURES IN PRE-/POST-AUTHENTICATION COMMUNICATION WITH UE. **L2 Control* MEANS DATA LINK PROTOCOLS SUCH AS MAC, RLC AND PDCP.

Related Work	Pre-Authentication			Post-Authentication			
	L2 Control*	RRC Setup	Security Context	Paging	Calling	Handover	PDU Session
[4] Berseker	○	◗	●	○	○	○	◗
[6] DoLTEst	○	◗	●	○	○	○	○
[13] LTEFuzz	○	◗	◗	○	○	○	○
[2] 5Greplay*	○	○	◗	◗	◗	◗	◗
[1] 5GReasoner	○	◗	●	●	○	●	○
[7] FIRMWIRE	○	◗	●	○	●	○	●
[8] BASESPEC	○	○	○	○	○	○	○
[9] BaseSAFE	○	○	●	◗	●	◗	●
[14] Usenix' 23	○	●	●	●	○	◗	○
[10] Wisec' 23	○	●	●	○	○	◗	●
5Ghoul	●	●	●	◗	○	○	●

a protocol by its layer name (see `R.filter` of Algorithm 1) and the property `StateNameField` identifies each *type-field* of the protocol layer that can be used to map a packet to a protocol state (see `R.type_fields` of Algorithm 1). Such rules follow the *PCAP* filtering syntax, which is used in the *Wireshark* packet analyzer program.

We note that while PDCP is a separate data-link layer, 5GHOUL does not add an explicit rule for this layer. This is because there are no 5G NR downlink packets associated with PDCP-only state. By design, PDCP protocol is always followed by RRC, or is partially present inside fragmented RLC payloads. As a result, the rules shown in Figure 7 are sufficient to label the state for any packet intercepted by 5GHOUL. From Figure 7, we also note that *MAC, RLC* and *NAS* can be identified by multiple fields depending on the communication context. In our evaluation of 5GHOUL, by using just four rules (see Figure 7), 5GHOUL state mapper had generated a state machine with 38 states and 308 transitions for fuzzing 5G UE.

*3) **Completeness of State Machine**:* Table I highlights the 5G procedures covered during the state mapping process and contrasts 5GHOUL state machine with prior works

that only focus on testing layer 3 procedures (e.g., RRC and NAS protocol procedures). In general, the protocol procedures are classified as *Pre-Authentication* or *Post-Authentication*. In this context, the state mapping process within 5GHOUL mainly focuses on pre-authentication (denoted by full moon circle in the first three columns of Table I). This is because *such procedures can be exploited by an attacker without having any knowledge of UE SIM card authentication details*. Consequently, 5GHOUL exhibits an edge over prior work, none of which considers layer 2 communication and such a consideration is crucial for testing early communication between UE and gNB (see steps 1-4 of Figure 1a).

4) **Optimization:** In this section, we discuss optimization of 5GHOUL in two ways: *(i)* maximize the coverage of protocol state machine M_{ref} transitions via an evolutionary process, and *(ii)* dynamically balance the chance to mutate protocol layers for fairly mutating all protocol layers.

Maximizing M_{ref} coverage: 5GHOUL evolves the fuzzing process with the objective to increase the transition coverage of the M_{ref}. To this end, any standard evolutionary algorithm can be used with a cost function tailored to the number of covered state transitions. We apply particle swarm optimization (PSO) due to its superior performance in stochastic optimization scenarios. This fits well within the context of 5G UE fuzzing due to occasional packet drops, network interference and unpredictable network delays. Moreover, PSO has been successfully applied in prior wireless fuzzing work [15].

To employ PSO within 5GHOUL, we dynamically adjust the probability of mutating different layers and fields of the packet. Concretely, once a packet P is intercepted and mapped to a state $S \in M_{ref}$, the 5GHOUL fuzzer decides to mutate P in line with the following probabilities: *(a)* pr_g : the probability attributed to mutate P or release P without any mutation, *(b)* pr_l : the probability to mutate the layer l within packet P, and *(c)* pr_f : the probability to mutate any packet field f within P. To keep the fuzzing process simple and efficient, we do not distinguish between mutation probabilities of different packet fields in P. Therefore, during a fuzzing campaign, since a packet can be mapped to an arbitrary state of M_{ref}, the set of mutation probabilities guiding the fuzzing process is M where $|M| = \sum_{s \in M_{ref}} \left(\left|layers(s)\right| + 2\right)$ and $layers(s)$ is the set of protocol layers found in any packet mapped to state s. The 5GHOUL fuzzing campaign starts with a random population of M and after each fuzzing session, updates the cost function (i.e., number of covered state transitions) for the entire population. The best individual within the population is computed after each fuzzing iteration in line with the standard PSO process.

We evaluate the impact of our evolutionary process on 5GHOUL effectiveness in **RQ3**.

Balancing mutations across layers: While the evolutionary process within 5GHOUL aims to maximize the coverage of transitions in M_{ref}, we also employ simple heuristics to balance the mutations across different network layers. The balancing strategy is employed after the PSO recomputes the mutation probabilities as described in the preceding section. We illustrate the impact of such a balancing strategy via Figure 8. It captures the frequency distribution of fuzzed downlink packets with respect to different network layers. As observed, it exhibits a significantly high degree of fuzzing for *MAC* layer in contrast to *RLC* and *RRC* layers. Moreover, the number of packets mutating the *NAS* and *PDCP* layers is negligible. Thus, the mutation behavior exhibited in Figure 8 (*No Protocol Balancing*) limits the possibility of finding implementation vulnerabilities in network layers beyond *MAC*.

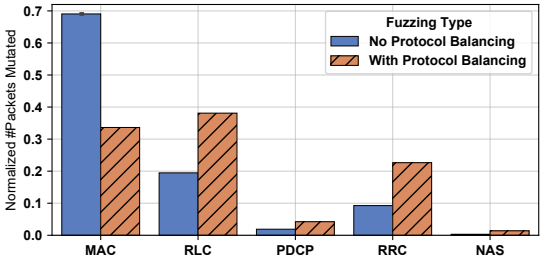


Fig. 8. Normalized Number of Packets Mutated w.r.t. 5G Protocol Layer.

To address the aforementioned situation, we reduce the mutation probability of a network layer l (i.e., pr_l) if it had been fuzzed frequently and compensate the mutation probabilities for network layers that were not fuzzed often. Concretely, in any fuzzing iteration, we compute the number of times a layer was fuzzed with respect to the total number of fuzzed packets. For example, if the fuzzing iteration had a total of N fuzzed packets and a layer l was fuzzed N_l times, then we refine pr_l as $pr_l \times \left(1 - \frac{N_l}{N}\right)$ which will be applied to the next fuzzing iteration. As a result, the mutation probabilities of a layer l is revised inversely proportional to the frequency of the layer was fuzzed in the current fuzzing iteration.

Figure 8 captures the result of refining the mutation probabilities *With Protocol Balancing*. As shown, the mutation rate of *MAC*, *RLC* and *RRC* layers are fairly balanced, whereas the mutation rate for *NAS* and *PDCP* improved.

B. UE Monitoring and Control

We have designed a *UE Manager* component (see Figure 9) that automatically configures, controls and monitors 5G UEs. This is accomplished by employing ***(i)*** *ADB* to manage 5G Android smartphones or ***(ii)*** *Freedesktop ModemManager (MM)* to similarly manage 5G USB modems during a fuzzing session. 5GHOUL leverages ***(i)*** or ***(ii)*** during fuzzing to collect logs and monitors the health of the targeted UE. Additionally, it recovers the UE from crashes or firmware hangs by reconfiguring or rebooting the UE.

Further, *UE Manager* detects firmware crashes in Android upon receiving log messages such as ***"ModemRestartStats"*** for Qualcomm-based smartphones or ***"ModemEvent: modem_failure"*** for MediaTek-based smartphones. We note that such crash detection can easily be modified.

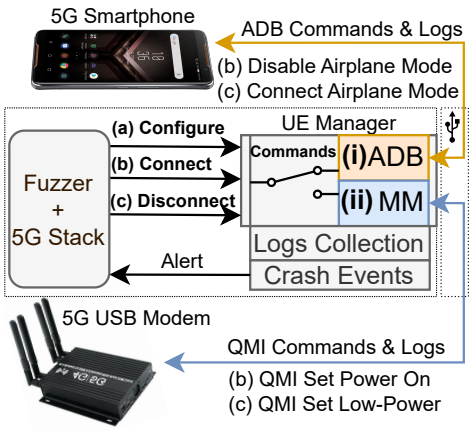


Fig. 9. An illustration of the UE Monitoring and Control workflow.

Android smartphones can be kept in a 5G connection cycle (fuzzing iteration) by simply disabling and enabling *Airplane* mode. However, doing the same with 5G USB modems is challenging. This is because 5G USB modems contain a combination of proprietary and common AT commands for its configuration, which respond differently depending on the vendor [16]. Furthermore, the AT interface can have random delays or unexpectedly close during the fuzzing process without a clear indication of a firmware issue. Therefore, to make our design ***modem agnostic***, 5GHOUL hands over control of the UE modem to *Freedesktop ModemManager* process (MM).

MM process communicates with the UE mostly via the standard QMI interface and falls back to AT when necessary. In particular, whenever the modem is configured and ready to be used, *MM* informs 5GHOUL (see "Alert" in Figure 9). Moreover, *MM* alerts modem crashes by informing QMI interface hangs or USB detachment caused by modem software reset.

C. Post Fuzzing Analysis

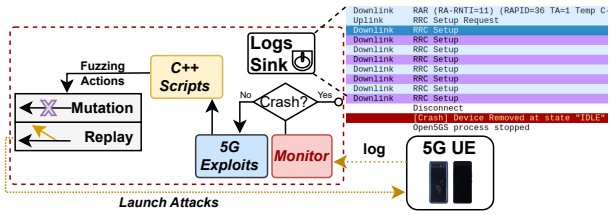


Fig. 10. The workflow of attack vector analysis for a crash or hang.

Figure 10 captures the workflow to systematically replicate and analyze crashes in our study. Upon completion of the fuzzing session, a *pcap* file is generated, containing all the communicated packets. In the event that downlink packets are mutated by 5GHOUL, they are color-coded in purple, while the original version of the malformed packet is captured (only for debugging) and color-coded in blue (see Figure 10). We note that if the packet was mutated by 5GHOUL, then only the mutated packet was sent to the target UE. Finally, upon crash detection, a customized crash packet in red is indicated in the *pcap* file (see Figure 10).

To facilitate the replication of vulnerabilities, we developed a systematic process to generate exploit code (C++ script). Such an exploit code leverages the network capture file (*pcap*) produced by the fuzzing session (see Figure 10) to launch an over-the-air attack exploiting the respective vulnerability. While generating the code for the exploit, 5GHOUL employs a simple heuristic to replicate the crash with minimal modifications to the communicated packets. This is desirable, as typically the crash log captured during the fuzzing session involves thousands of mutated or replayed packets. Let us assume $\langle \mu_1, \mu_2, \ldots, \mu_n \rangle$ captures the sequence of mutated packets (the same logic also applies for replayed packets) in a trace δ that results the crash. To replicate the crash systematically, we consider each mutated packet μ_i in isolation. To this end, 5GHOUL replays the trace δ where the only mutated packet is μ_i and each μ_j ($j \in [1,n] \wedge j \neq i$) is replaced with the original (non-mutated) counter part of μ_j. Intuitively, we aim to observe whether only the mutated packet μ_i is the root cause of the crash.

5GHOUL starts the replication process from the last mutated packet μ_n (i.e., the last purple packet in sequence as shown in Figure 10). This is because we posit that closer the mutated packet with respect to the crash location, higher the likelihood of the packet being the root cause of the crash. If the crash is not manifested only with μ_n, then 5GHOUL gradually traverses up the trace δ and repeats the process for the previously mutated packet in sequence i.e., μ_{n-1}. The replication process succeeds when the crash is reproduced in the process. Although it is theoretically possible for the crash to appear for an arbitrary combination of mutated packets, in our evaluation, all but one crash was reproduced by considering only one mutated packet in isolation. The exception was V4 from Table III, which necessitated a sequence of two packets to be mutated to replicate the crash. In general, based on the results of our experiments, we can confidently conclude that our heuristic on replication method works well in practice.

Figure 11 captures the exploit code generation process for V10 (see `Invalid RLC Data Sequence` in Table III). For the sake of brevity, we illustrate the "Original Trace" and "Fuzzed Trace" side-by-side in Figure 11. Moreover, we show the code generation for an arbitrary mutated packet μ_i. The process is simply repeated if multiple mutations are required for replicating the crash. The key idea is to ① first identify the state of the mutated packet in line with the process discussed in Section III-A2. To this end, we identify that the mutated packet matches the mapping rule corresponding to *Layer* `rlc-nr` and *StateNameField* `rlc-nr.am.dc`. Our replication process then generates the code to intercept packets that correspond to these layers and types. This is captured via the `wd_filter` function call. We note that the raw value of the `rlc-nr.am.dc` field was extracted from the mutated packet and the type value `CTRL` was obtained from Wireshark lookup dictionary (similar to the process illustrated in Figure 6). Finally, the `wd_filter` function call code is generated via a simple file

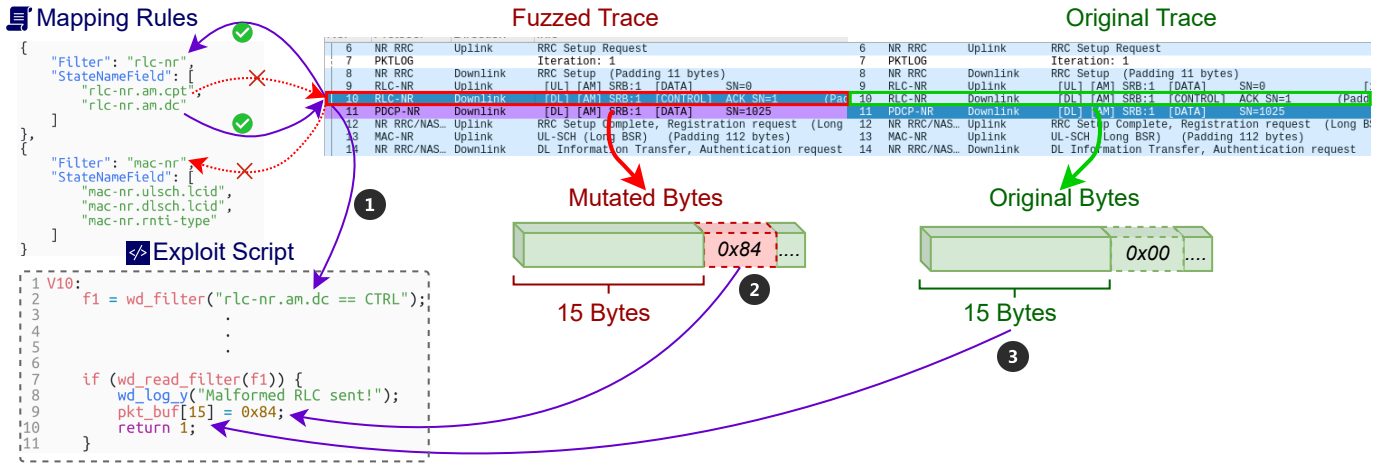


Fig. 11. The workflow of exploit C++ code generation. ① Mutated packet is matched with the state mapping rules to precisely generate a condition that intercepts packets at runtime. Such interception is for the purpose of replicating the respective vulnerability. Subsequently, the original and mutated communication traces are matched to compute ② the mutated value and ③ the location (i.e., byte offset) for the mutation.

template. Intuitively, this code intercepts the packets that need to be mutated at runtime for replicating the respective vulnerability.

Once the condition for packet interception is generated, we find ② the mutated field value and ③ the location of the mutation (i.e., the byte offset). This is accomplished by comparing the original and mutated version of the communication trace (see Figure 11). By locating the difference between the original and corresponding mutated packet, the code for the mutation is generated via another file template as illustrated in Figure 11 (lines 7-11). Intuitively, the exploit code automates the process of selecting only those packets that need to be mutated, applying the specific mutation to such packets and releases them to the target.

IV. Implementation and Evaluation Setup

Hardware Setup: Concurrently, Figure 13 illustrates the physical setup of our 5GHOUL. The hardware components include a *Beelink SER5 Mini PC* powered by an *AMD Ryzen 7 5800H* processor, *Software Defined Radio (SDR)* such as the *USRP B210*, and targets UE such as *Quectel RM500Q-GL* 5G USB modem and an off-the-shelf 5G smartphone (*OnePlus Nord CE 2*). Finally, the USB Per-Port Power Control Hub is used to provide automatic power-cycling of USB power in case the UE target hangs and requires a manual reboot. This setup allows for efficient analysis and evaluation of the 5G network e.g., identification of 5G UE vulnerabilities. All UE targets used in our evaluation and their corresponding firmware version are outlined in Table II.

Running 5GHOUL: To launch our experiment, we first connect the SDR to the PC via a USB. Next, the *gNB* operates in the N78 frequency band, which is widely-used in Europe and Asia. Furthermore, we enable a testing 5G network using *Mobile Country Code (MCC)* as 001 and the *Mobile Network Code (MNC)* as 01. This step is required for 5G Smartphones, which commonly reject *gNBs* with

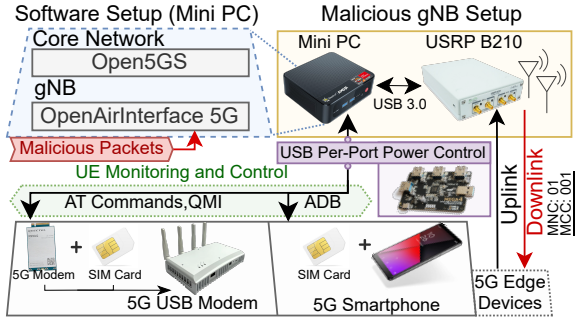


Fig. 12. Hardware Setup for 5GHOUL testing and evaluation.

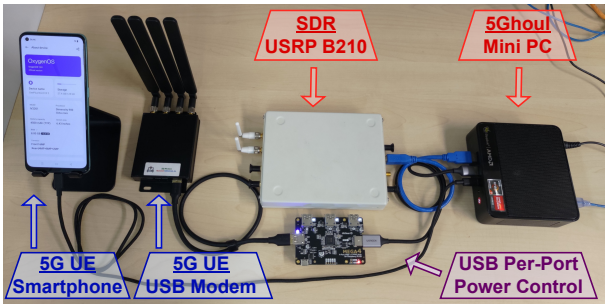


Fig. 13. Physical Setup for 5GHOUL. The 5G USB modem is connected to the *Mini PC* through the *USB Per-Port-Power Control Hub*. The 5G Smartphone is directly connected to the *Mini PC*.

arbitrary *MCC/MNC*. Finally, we run a 5GHOUL container, which starts a malicious gNB ready for fuzzing the UE.

Fuzzer Implementation: 5GHOUL is implemented in C++ (*7879 LoC*). This includes patches to Wireshark that improve decoding speed for 5G protocols, while exposing more fields information. Moreover, we develop patches to add the *interception points* to OpenAirInterface5G (see Section III-A). To generate reference traces of valid 5G communication for the *State Mapper* (Section III-A1), we connect each UE modem and allow normal communication and re-connection for approximately 12 hours for each modem. The communication logs from these normal communication are then

TABLE II
DEVICES USED FOR EVALUATION. THE SAMPLE CODE IS PROVIDED BY VENDOR TO TEST THE DEVELOPMENT BOARD. THIS IS NOT APPLICABLE (N.A) ON PRODUCTS RUNNING A FIXED APPLICATION.

Vendor / Product	5G Modem	Type	Monitor	Firmware/Software Version
OpenAirInterface UE	N.A	Software	ProcessMonitor	2023.w03
Quectel RM500Q-GL	Qualcomm X55	USB Modem	ModemManager	Aug 03 2021
Simcom SIM8202G	Qualcomm X55	USB Modem	ModemManager	SIM8202G-M2_V1.2
Fibocom FM150-AE	Qualcomm X55	USB Modem	ModemManager	89602.1000.00.04.07.20
Telit FT980m	Qualcomm X55	USB Modem	ModemManager	38.23.001-B001-P0H.000640
Oneplus Nord CE 2 5G	Dimensity 900 5G	Smartphone	ADB	M_V3_P10
Samsung S22 (5G)	Qualcomm X65	Smartphone	ADB	S901EXXU4CWCE
Asus ROG Phone 5s	Qualcomm X60	Smartphone	ADB	M3.13.24.73-Anakin2

TABLE III
SUMMARY OF 5G IMPLEMENTATION VULNERABILITIES AND AFFECTED UE DEVICES. VULNERABILITIES IN **bold** ARE ASSIGNED HIGH-SEVERITY BY THE VENDOR.

Implementation Vulnerability	Affected 5G Modems/Smartphones								Layer(s)	Impact	CVE Status
	OAI UE	Telit FT980m	Fibocom FM150-AE	SIMCOM SIM8202G	Quectel RM500Q-GL	Asus ROG Phone 5S	OnePlus Nord CE 2	Samsung Galaxy S22			
V1 - Invalid PUSCH Resource Allocation	X								RRC	Crash	Pending
V2 - Empty RRC dedicatedNAS-Message	X								RRC, NAS	Crash	Pending
V3 - Invalid RRC Setup			X	X					RRC	Crash	Patched
V4 - Invalid RRC Reconfiguration				X					MAC, RRC	Crash	Patched
V5 - Invalid MAC/RLC PDU		X	X			X			MAC, RLC	Crash	CVE-2023-33043
V6 - NAS Unknown PDU		X		X		X			NAS	Crash	CVE-2023-33044
V7 - Disabling 5G / Downgrade via RRC		X	X	X	X	X			RRC	Hang / Downgrade	CVE-2023-33042
V8 - Invalid RRC Setup spCellConfig							X		RRC	Crash	CVE-2023-32842
V9 - Invalid RRC pucch CSIReportConfig							X		RRC	Crash	CVE-2023-32844
V10 - Invalid RLC Data Sequence							X		RLC	Crash	CVE-2023-20702
V11 - Truncated RRC physicalCellGroupConfig							X		RRC	Crash	CVE-2023-32846
V12 - Invalid RRC searchSpacesToAddModList							X		RRC	Crash	CVE-2023-32841
V13 - Invalid RRC Uplink Config Element							X		RRC	Crash	CVE-2023-32843
V14 - Null RRC Uplink Config Element							X		RRC	Crash	CVE-2023-32845

leveraged to create the reference state machine. We note that the collection of reference traces is a one-time effort and is not required *during* fuzzing campaign.

5GHOUL supports running OAI stack with real hardware using USRP B210 for over-the-air communication.

V. EVALUATION RESULTS

To evaluate 5GHOUL and showcase its capability, we answer the following research questions:

A. *RQ1: How effective is* 5GHOUL *fuzzer in terms of generating error-prone inputs?*

Table III outlines the effectiveness of 5GHOUL in finding vulnerabilities on *COTS* edge devices. In the first column, each vulnerability name is identified with prefix ***V***. While the first two vulnerabilities (***V1*** and ***V2***) affect an open-source UE implementation from *OpenAirInterface* project, the rest of the vulnerabilities affect many popular 5G USB Modems (***V3-V7***) or representative Smartphones employing *Qualcomm* or *MediaTek* Modems (***V5-V14***).

In our findings, *Qualcomm* 5G USB Modems and 5G-enabled Smartphones such as *Asus ROG Phone 5S* were impacted by a range of previously unknown vulnerabilities in the handling of *MAC/RLC*, *RRC* and *NAS* messages (high severity ***V5-V7***). On the contrary, modems running an older 5G firmware e.g., *SIMCOM SIM8202G* and *Fibocom FM150-AE* (see Table II) were additionally affected from *MAC/RLC* and *RRC* vulnerabilities already patched (***V3*** and ***V4***). Such issues did not affect the other 5G USB Modems that had firmware with build date in 2023.

In summary, the *RRC Attach* and *Authentication* procedures (see Figure 1a) exhibited all vulnerabilities discovered by 5GHOUL. In particular, *RRC Attach* procedure, which contains the *RRC Connection Setup* message, was focus of most vulnerabilities. Such findings were aggravated during the fuzzing sessions for *OnePlus Nord CE 2*. With such *UE* employing *MediaTek Dimensity 900 5G* modem, many asserts and memory-related crashes (see Table III) were triggered during the exchange of *RRC Setup Connection*.

Lastly, we note that all vulnerabilities were found during the pre-authentication stage of the communication between UE and gNB. This means that attacks exploiting ***V1-V14*** do not require any secret information from the UE's SIM card to be successful. Such attacks can be launched by starting a malicious gNB with the same setup as shown in Section IV (see Section II-C). Overall, our results highlight 5GHOUL not only as the first OTA fuzzer to find 5G data link implementation vulnerabilities (***V4, V5***, ***V10***), but also demonstrates its effectiveness in finding vulnerabilities at layer 3 and above (e.g., *RRC, NAS*) for commercial 5G UEs.

B. *RQ2: How effective is* 5GHOUL *fuzzer w.r.t fuzzing time?*

Table IV outlines the time taken to complete one fuzzing session (10^4 fuzzing iterations) for each target UE. 5GHOUL

TABLE IV
TIMING AND MODEL COVERAGE OF 10^4 FUZZING ITERATIONS PER UE.

Vendor / Device	Total Time	1st Crash/Hang	Model Coverage (States)
OpenAirInterface UE	21 min.	1 min.	9.7% (30)
Quectel RM500Q-GL	12 h. 54 min.	>12h.	75.0% (231)
Simcom SIM8202G	6 h. 31 min	5h. 17 min.	83.4% (257)
Fibocom FM150-AE	12 h. 45 min.	1h. 37 min.	86.4% (266)
Telit FT980m	12 h. 26 min.	>12h.	50.6% (158)
Asus ROG Phone 5s	12 h. 47 min.	>12h.	74.4% (229)
OnePlus Nord CE 2 5G	13 h. 17 min.	19 min.	70.8% (218)
Samsung S22 5G	12 h. 25 min.	N.A	67.9% (209)

finishes each fuzzing iteration by triggering reconnections via ADB or QMI whenever the UE completes the 5G procedures shown in Figure 1a. However, fuzzing the data link can normally lead to UE unresponsiveness for several seconds (i.e., $2-4$ seconds) without necessarily indicating a firmware issue. This is because 5G modems implement their own waiting states after receiving decoding errors or handling expected failure states. Such *inherent delays* are evident with *OnePlus Nord CE 2*, which results in a total time of $13h$. to complete a fuzzing session (see Table IV). Nonetheless, across all COTS UEs, *OnePlus Nord CE 2* exhibited the fastest crash (19 min.) due to 5GHOUL fuzzing.

On the other hand, the aforementioned *inherent delays* are not present in *OpenAirInterface* UE, as the only waiting time is the UE process *restart/startup* delay before every fuzzing iteration starts. Consequently, such UE was the fastest to evaluate, thus completing the fuzzing session in 21 minutes. However, such UE had the least *Model Coverage* (last column of Table IV). This is because such UE does not complete the *Authentication* procedure of Figure 1a due to interoperability issues with *5G NR Core Network* (*Open5GS*).

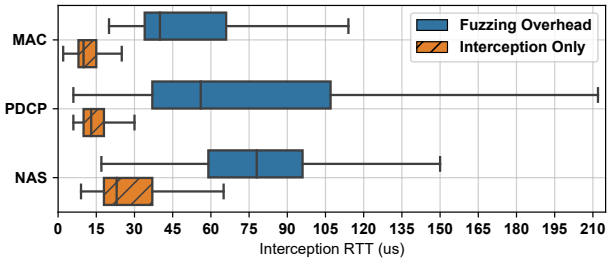


Fig. 14. Downlink Interception time. Worst-Case Outlier RTT=$971us$.

We also measure the performance of 5GHOUL from the time a downlink packet is intercepted and processed until the packet is released back to the gNB. The Round Trip Time (RTT) for intercepting *MAC, PDCP* and *NAS* protocols with and without the fuzzing overhead is shown as a box plot in Figure 14. In summary, the interception time of 5GHOUL makes it suitable for real-time data link fuzzing in 5G networks. The maximum RTT of $110us$ for the MAC protocol is well below the latency requirements specified by the *numerology* parameter at the gNB [12]. Moreover, the decoding speed for network layer payloads such as *RRC* (carried over *PDCP*) and *NAS* significantly outperforms recent work [4] that exhibits decoding timings $1-3ms$.

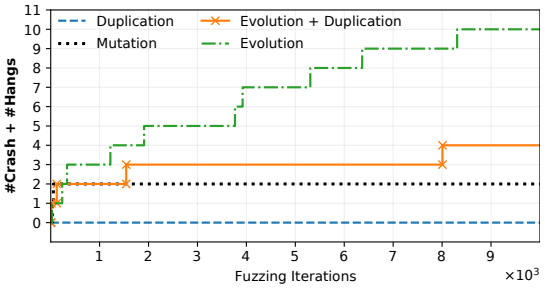


Fig. 15. Fuzzing Iterations vs *Unique Crashes + Hangs* for all target UEs.

C. *RQ3: How do the different design choices contribute to the effectiveness of* 5GHOUL *fuzzer?*

To evaluate the effectiveness 5GHOUL design, we generate four variants of 5GHOUL, each testing a specific fuzzing component discussed in Section III and evaluate each component across all target UEs.

To this end, we show in Figure 15 the number of firmware *Crashes* for 10^4 fuzzing iterations for each 5GHOUL variation: *Duplication* refers to 5GHOUL with only packet injection enabled, whereas *Mutation* has only random packet mutation enabled without any optimization to mutation probabilities pr. *Evolution* variant enables packet mutation and *Optimization*. Lastly, *Evolution + Duplication* enables both *Optimization* and packet injection.

Our evaluation of 5GHOUL across all devices revealed that *Evolution* variant indeed discovered many new vulnerabilities and at a higher rate than the other variants. As shown in Figure 15, such variant yielded *ten* unique *#Crash + #Hangs* at the end of the 10^4 fuzzing iterations. This exemplifies the need for our evolutionary optimization (Section III-A4) during fuzzing campaign. On the contrary, the variant *Duplication* adds more time-outs during the fuzzing session due to sporadic UE unresponsiveness. Such a behavior was observed to conflict with our optimization process. As a result, when *Duplication* is combined with *Evolution*, it reduced the effectiveness of 5GHOUL, as shown in the results of *Evolution + Duplication* in Figure 15.

Additionally, we evaluate the impact of the Mutation Balancing component during fuzzing. While Section III discusses the distribution of mutated packets across different protocols (see Figure 8), we aim to evaluate how the *correction* for such distribution affects 5GHOUL capability to find bugs. To this end, we evaluate two variants of 5GHOUL (with and without mutation balancing) against the UE target *Oneplus Nord CE 2 5G*, which exhibits the most vulnerabilities. The results are shown in Figure 16 and represent the total number of crashes (i.e., containing repeated vulnerabilities) across 7000 iterations. Notably, the 5GHOUL variant without balancing takes more time to trigger vulnerabilities since the higher occurrence of MAC packets skews the fuzzing towards often mutating the MAC layer. In contrast, the 5GHOUL variant with mutation balancing can consistently trigger more vulnerabilities within a shorter number of iterations due to the MAC layer being less prioritized as opposed to other less frequent protocol

TABLE V
POST FUZZING ANALYSIS FOR REPRODUCTION OF VULNERABILITIES ***V1-V14.***

# Messages		# Mutated Fields		#Replication Steps	
1	2	1	2	1	≥2
All Except V4	V4	V1-4 V6-7 V9-10 V12-14	V5 V8 V11	V1-3, V5-6 V8-11 V13,V14	V4: 6 V7: 3 V12: 2

layers such as RLC, PDCP, RRC and NAS. In a broader view, the result also highlights the suitability of employing such balancing component when fuzzing other wireless protocols that might repeatedly transmit data-link packets during communication.

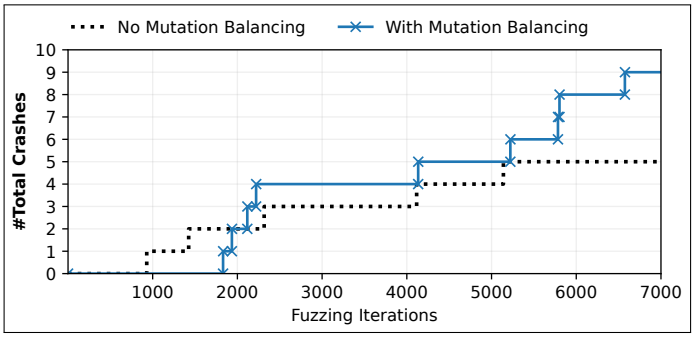


Fig. 16. Fuzzing Iterations vs *Total Crashes* **with** and **without** mutation balancing for target *Oneplus Nord CE 2 5G*.

D. ***RQ4 - How efficient is the Post Fuzzing Analysis in replicating crash/hang?***

The outcome of 5GHOUL post-fuzzing analysis is outlined in Table V. We show the number of messages transmitted for over-the-air exploitation together with the number of fields subjected to mutation and the number of iterations involved during replication (see Figure 10). With the exception of *V4*, which necessitates the transmission of two messages, and *V5, V8, V11*, which require the mutation of two fields, all vulnerabilities can be exploited by mutating just one message and a solitary field.

It is worthwhile to highlight that the replication steps hold paramount significance in the post-fuzzing analysis. This is because an exhaustive enumeration of all possible combinations of mutated packets, as found in the vulnerable communication trace, is infeasible to identify the root cause of the vulnerability. Thanks to the systematic process described in Section III-C, replication of all vulnerabilities are completed within a maximum of six iterations. In fact, except for *V4, V7, V12*, all vulnerabilities were replicated in just one iteration. This results show that our heuristic of replicating vulnerabilities is effective in practice.

E. ***RQ5 - How does 5GHOUL compare to existing 5G security testing frameworks?***

Although 5GHOUL is a 5G fuzzing framework, it can also be used to perform standard 5G protocol testing towards the UE. Therefore, we evaluate and compare 5GHOUL to state-of-the-art 5G security testing frameworks which allow users to design communication scenarios (test cases) according to user-provided configuration files. However, there exists several limitations of such frameworks with respect to 5GHOUL capabilities and we outline such limitations in Table VI. Firstly, existent frameworks have limited protocol support (column *Protocols*) and hence do not support manipulation of data link messages such as MAC, RLC and PDCP. Therefore, discovering any data link vulnerabilities such as *V4* and *V5* (see Table III) is infeasible in *5Greplay* [2]. Secondly, current frameworks only allow injection or replay of either *(i)* known RRC or NAS messages [10] or *(ii)* NAS messages from a PCAP file [2]. As a result, our evaluation with existing 5G UE testing framework [10] reveals that it fails to discover any vulnerabilities found by 5GHOUL. This is because the vulnerabilities found by 5GHOUL involves *injection of invalid or malformed RRC and NAS messages*. Similarly, 5Greplay [2] only replicates **V6** as it modifies communication between core network and gNB. Thus, only NAS packets can be injected towards the UE.

TABLE VI
CAPABILITIES OF 5GHOUL WHEN COMPARED TO OTHER SECURITY TESTING FRAMEWORKS.

Frameworks	Communication Testing Capabilities					
	Injection	Replay	Flooding	Validation	Protocols	UE Monitoring
[2] 5Greplay	◗	●	●	●	◗	○
[10] Wisec' 23	◗	◗	◗	◗	◗	◗
5Ghoul	●	●	●	●	●	●

Finally, while both 5GHOUL and 5Greplay [2] support a *Deep Packet Inspection* (DPI) library to validate responses (*Validation* column in Table VI) in real-time, only 5GHOUL supports automated monitoring and control of UE targets such as smartphones, modems and software processes. This enables 5GHOUL to be used as a single solution to automate replication and validation of 5G security-related test cases once a C++ script is provided by the user (or created as a byproduct of the post-fuzzing analysis as described in Section III-C). In summary, our evaluation highlights the feasibility and practicality of 5GHOUL to be used as a standard 5G security testing framework as opposed to just an over-the-air 5G fuzzing tool.

F. ***RQ6 - Can 5GHOUL be extended to find vulnerabilities beyond crash or hangs?***

As discussed in **RQ1**, 5GHOUL discovered security issues related to hang or crashes (see Table III). Nonetheless, 5GHOUL can be extended to discover security issues beyond crashes due to the comprehensive injection and mutation capabilities embodied within the 5GHOUL fuzzing. To illustrate this, we evaluate the capability of 5GHOUL to replicate and automatically discover downgrade attacks. To this end, we configured a legitimate 4G and a malicious 5G network, and the logs of the base stations are monitored in real-time. By default, UEs prefer connecting to 5G due to its latest-generation status. However, following a specific attack, when a UE shifts from the 5G network to the 4G network,

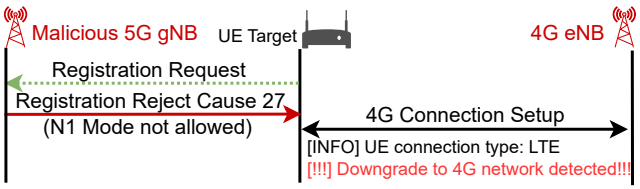


Fig. 17. 5G UE downgrade attack by sending downlink Registration Reject

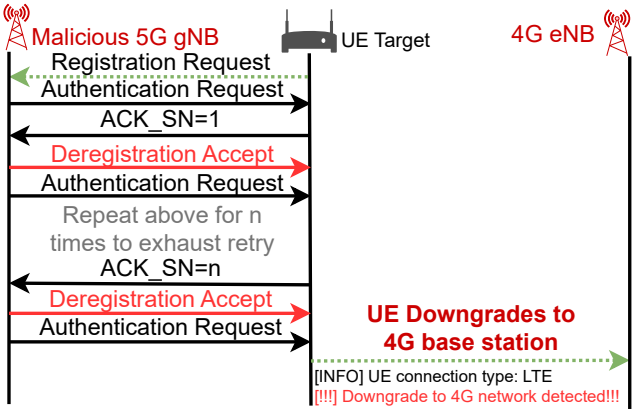


Fig. 18. 5G UE downgrade attack by flooding downlink Deregistration Accept message.

the log will indicate a connection to the eNB instead of the gNB, 5GHOUL automatically flags this behavior as a downgrade attack.

As Illustrated in Figure 17, 5GHOUL automatically detects a downgrade attack proposed by earlier works [10]. During such an attack, our rogue base station swiftly responds to the UE's *Registration Request* by a *Registration Reject* with a specific Reject Cause (Cause 27 - N1 Mode Not Allowed). This prompts the UE to disconnect from the 5G and reconnect to the 4G network. After the attack, user has to force the UE to re-connect to 5G network by toggling the airplane mode.

It is worthwhile to mention that 5GHOUL *injects messages to launch the attacks instead of directly altering the internal states of the core network stack.* Therefore, 5GHOUL *is able to find new downgrade attacks which exploit message flows that contains authentication retry procedures.* As shown in Figure 18, we discovered that flooding a `Deregistration Accept` message before the 5G base station sends the *Authentication Request* message results in the base station keep re-sending *Authentication Request* with a duplicated sequence number. Subsequently, after consecutive failures of delivering *Authentication Request*, the UE disconnects from the 5G network and shifts to the 4G network, indicative of a downgrade attack. Furthermore, our findings reveal that the rogue base station may also employ other type of messages (including *message_type* = `0x00`) instead of `Deregistration Accept` to force the UE to downgrade to 4G network. Notably, unlike the downgrade attack shown in Figure 17, the UE cannot re-establish the 5G connection by merely toggling the airplane mode; it necessitates

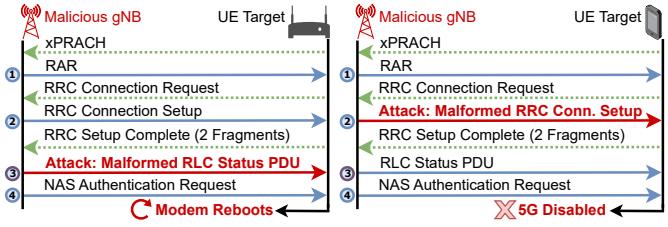


Fig. 19. Figure 19a illustrates *"Invalid MAC/RLC PDU"* (***V5***) and Figure 19b depicts *"Invalid RRC Setup pdcch-Config"*. While ***V5*** triggers a modem reboot on the target device, ***V7*** brings the target to a 5G disabled state.

initiating a new connection process with the base station. This attack is not possible with prior framework since the handling of injected messages is hard-coded and hence, retries are not considered. We successfully launched the new downgrade attack (Figure 18) on four different smartphones: *Xiaomi Redmi K40* (MediaTek Dimensity 1200), *Oneplus Nord CE 2* (MediaTek Dimensity 900), *Samsung Galaxy S22* (Qualcomm X65) and *Asus ROG Phone 5s* (Qualcomm X60).

In summary, 5GHOUL not only demonstrates its effectiveness by replicating the downgrade attack which initially proposed by an earlier work [10], but also shows its higher potential for targeting different attack scenarios including new downgrade attacks (Figure 18).

VI. 5G Implementation Vulnerabilities

In this section, we discuss the pre-conditions to launch 5G attacks through 5GHOUL and provide a general summary of the discovered vulnerabilities.

TABLE VII
PRE-CONDITIONS FOR MALICIOUS GNB TO TRIGGER VULNERABILITIES.

5G NR Procedure	Message Name	Vulnerabilities	Required Information
RRC Attach (Initial Setup)	RRC Connection Setup	**V1, V3, V7-V9, V11-V14**	MCC, MNC
	MAC Time Adv. Command	**V4**	MCC, MNC
Authentication	RLC Status PDU	**V5, V10**	MCC, MNC
	NAS Authentication Req.	**V2, V6**	MCC, MNC

The attacks that trigger the vulnerabilities described in Table III are tested against each UE target by leveraging the attack model discussed in Section 3. At minimum, knowledge about the legitimate gNB *MCC/MNC* is required to start the cloned gNB and launch the attacks. The exact *5G NR Procedure* and *Message Name* (see Figure 1a) that correspond to each vulnerability is detailed in Table VII. All vulnerabilities are triggered before NAS authentication finishes (after step 4 of Figure 1a). For example, to trigger vulnerabilities ***V2*** and ***V6***, which rely on mutating the *NAS Authentication Req.*, the attacker may simply modify the 5G Core Network (*Open5GS*) stack to accept arbitrary UE's *Subscription Permanent Identifier (SUPI)* so that *NAS Authentication Req.* message is forcibly generated containing arbitrary and incorrect authentication values and nonces.

In practicality, such vulnerabilities can be easily exploited over-the-air by starting a malicious gNB within radio range of the target 5G UE device (see Figure 13). This is a practical

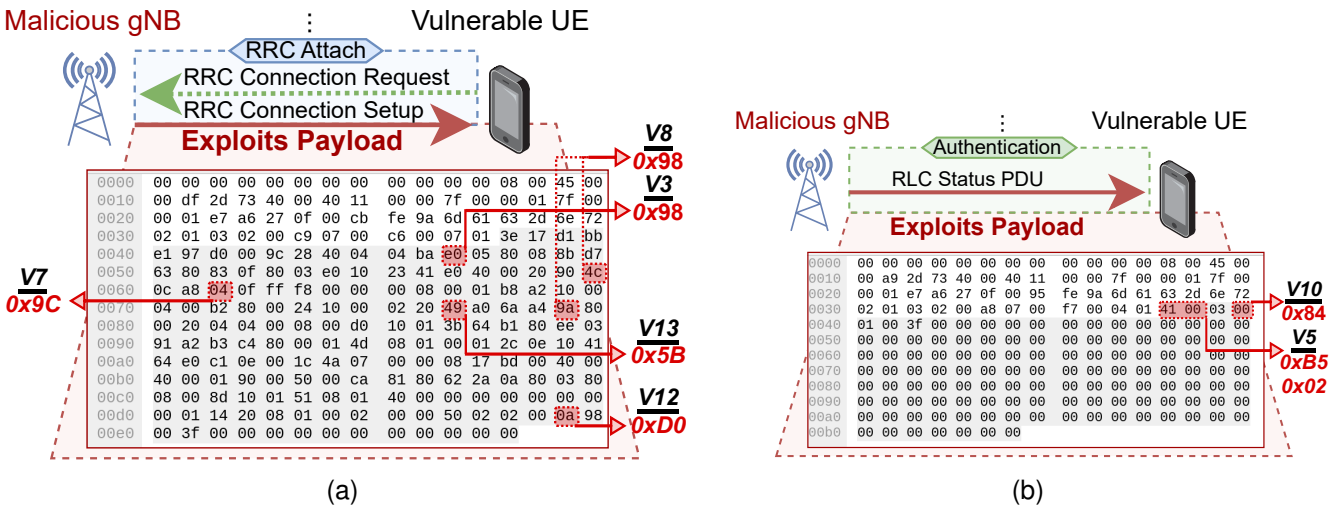


Fig. 20. Illustration of the adversarial-controlled gNB attack vectors via mutation of the RRC message payload **(a)** and RLC message payload **(b)**.

setup which relies on using Software Defined Radio (SDR) to behave as a cloned gNB. While USRP B210 used in our setup could be recognized from afar, thus making the attack visually noticeable, such type of equipment has already been miniaturized to the size of a raspberry-pi [17]. This, in turn, enables the use of SDR for visibly stealthy attacks.

A. Summary of Vulnerabilities

While 5GHOUL found 5G modem Vulnerabilities in almost all downlink messages up to *NAS Authentication Req.*, representative vulnerabilities ***V5*** and ***V7*** are illustrated in Figure 19a and Figure 19b, respectively. The key difference between these vulnerabilities, aside from being triggered from different messages, is their impact on the target UE:

V7 - Invalid RRC Setup pdcch-Config: Improper optional bits toggled in the RRC pdcch-Config element puts the modem in an unstable state which does not initiate connection to any 5G network. Instead, only connection to 4G or 3G works, hence such attack is classified as downgrade. This invalid state cannot be recovered by itself without a power cycle and hence it requires a manual reboot from the user. This takes about a minute in modern smartphones. Moreover, the attack vector to trigger this vulnerability is simply writing **0x9C** to the 38^{th} byte of the *RRC Connection Setup* Message (see Figure 20a).

V5 - Invalid MAC/RLC PDU: Writing invalid bytes to the 5G NR MAC Header causes a modem assert. In such a scenario, the modem automatically restarts via some software reset or watchdog mechanism, thus ensuring normal operation without intervention of the user. A sample Wireshark capture of the exploitation payload that immediately triggers ***V5*** over-the-air is shown in Figure 20b. In such attack vector, the malicious gNB mutates the first 2 bytes of the MAC header to bytes **0xB5** and **0x02** respectively.

Overall, the vulnerabilities summarized in Tables III and VII are triggered by either sending a malformed *RRC Connection Setup* (albeit writing to different RRC fields as depicted in Figure 20a) or sending a malformed *NAS Authentication Request*. In the particular case of ***V2***, the *NAS Authentication Request* is malformed such that the actual NAS message payload is empty. In contrast, ***V4*** is the only vulnerability which requires sending both an invalid *MAC Time Advance Command* and a malformed *RRC Connection Setup*.

Finally, while most vulnerabilities result a *Modem Reboot* behavior due to firmware reachable asserts, *MediaTek Dimensity 900* 5G Modem yields a memory access violation upon receiving an unexpected RLC data fragment due to a mutated *RLC Sequence Counter* field (***V10***). Further, the Modem triggers another memory access violation upon receiving a truncated *physicalCellGroupConfig* element in the *RRC Connection Setup* message (***V11***).

VII. Vulnerabilities Impact

A. Impact on Mobile Devices

To test the impact of 5G vulnerabilities on mobile devices and hence user experience, we exploit vulnerabilities *V5* to *V10* against *Asus ROG Phone 5S (ARP5s)* (Qualcomm Modem) and *OnePlus Nord CE 2 (OnePlus)* (MediaTek Modem). First, when vulnerability **V5** (see Figure 19a) or **V6** is triggered on *ARP5s*, its 5G modem immediately reboots and automatically recovers connection to gNB in few seconds (temporary DoS). Hence, an attacker exploiting ***V5*** and ***V6*** must continuously launch the attacks if the intention is to completely disrupt mobile network connectivity of the user. Additionally, 3G and 4G communication are also disrupted upon modem reboot since the modem handles all *3GPP* related communication.

More surprisingly, vulnerability ***V7*** (see Figure 19b) can prevent *ARP5s* to connect to any 5G network, while keeping 4G/3G connectivity intact (*downgrade denial-of-service attack*). Nevertheless, such behavior of ***V7*** highlights that the mobile device modem enters an erratic state such that the user needs to manually reboot the phone, thus power cycling the modem to fully restore 5G connectivity.

Attack *V7* is illustrated in Figure 21. Initially, the *ARP5s* is connected to a legitimate gNB (shown as ***SUTD 00101***

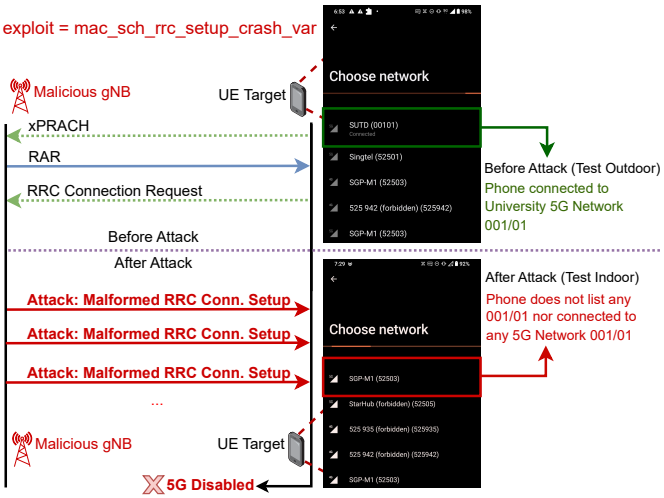


Fig. 21. The impact of exploiting V5 on *ARP5s*. The upper half of the figure illustrates the availability of *SUTD 00101* 5G network before the attack. In contrast, the lower half illustrates the unavailability of any 5G network connectivity after the attack is launched.

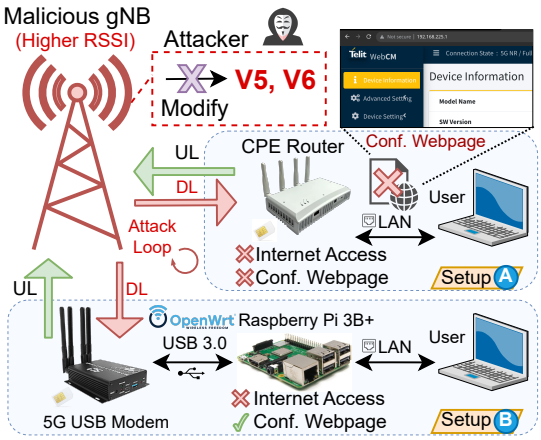


Fig. 22. Impact of ***V5, V6*** on *Telit FT980m* operating as CPE Router.

before the attack). After the malicious gNB starts and *V7* is exploited by the 5GHOUL framework, the mobile device depicts the following behavior:

1) Neither legitimate or malicious gNB are listed when re-scanning for mobile networks (see Figure 21).
2) No connection to any 5G network even when manually attempted by the user via the Android mobile network selection menu.

Lastly, vulnerabilities ***V8-V14*** trigger crashes on *OnePlus* which employs *MediaTek Dimensity 900 5G Modem*. More specifically, ***V8, V9*** and ***V12-V14*** trigger reachable asserts in the internal microcontroller and DSP of the 5G modem. Likewise, ***V10*** produces an invalid memory access exception. For all cases, the modem immediately reboots and takes a few minutes to recover 5G connectivity. Similar to vulnerabilities ***V5*** and ***V6***, an attacker would need to continuously launch the attacks to keep disrupting all 3G/4G/5G communications on *OnePlus*. However, ***V10*** and ***V11*** are more concerning due to the resulting memory access violations. Hence, they expose potential risk to enable arbitrary code execution or other memory related attacks directly in the 5G modem.

B. Impact on specialized 5G use cases

As highlighted in Table III, vulnerabilities ***V5*** to ***V10*** affect 5G devices that employ modems from Qualcomm and MediaTek. Therefore, these vulnerabilities affect not only smartphones and USB modems, but also appliances that rely on low-latency communication.

To assess the practical impact of 5GHOUL vulnerabilities, we conducted exploit tests on specific 5G UE Modems listed in Table II and analyzed their behavior. Two Qualcomm-based Modems were configured as Customer Premises Equipment (CPE) to evaluate the impact of **V5** to **V7**. In ***Setup A***, the *FT980m* modem within the *FT980WW* platform was tested, which provides 5G internet connectivity through its LAN port and hosts a *Configuration Webpage*. Continuous attacks ***V5*** and ***V6*** from a malicious gNB resulted in a complete loss of internet connectivity for devices connected to the FT980WW LAN port, and the Configuration Webpage became inaccessible due to modem reboots caused by the attacks.

In ***Setup B*** (shown in Figure 22), a Raspberry Pi 3B+ with *OpenWRT 22.03.4* and *RM500Q-GL* 5G USB Modem were used. This setup offered better isolation against attacks ***V5*** and ***V6*** since the *Configuration Webpage* was hosted by the *Raspberry Pi 3B+* processor instead of the modem itself. This allowed the user to remotely attempt to steer the CPE device by selecting a different mobile network to connect.

The effects of the launched attacks on CPE devices are summarized in Table VIII. The table describes the impacted behaviors within the CPE caused by attacks exploiting ***V5-V7*** and indicates whether manual intervention was required for recovery (the last column). During our tests, the *FT980-WW* CPE exhibited more proactive attempts to recover 5G connectivity after attacks exploiting ***V5*** and ***V6***, while the *Raspberry Pi 3B+ with OpenWRT* was less proactive, often requiring manual reconnection of the 5G Modem via *OpenWrt's Webpage*. Such a reconnection instability (shown in column 3 of Table VIII as ⚠) involves manually rebooting via OpenWrt's Webpage and restore proper 5G connectivity when ***V5-V7*** stopped. On the other hand, since *FT980-WW* runs the web application within its 5G Modem, the user was unable to access the *Webpage*. As a result, the user cannot steer the CPE from the malicious gNB while the attack was still under-going. Which means the user intervention does not apply when trying to recover from the attack.

For attack ***V7***, its effects were immediate, causing the targeted modem to disconnect from any 5G mobile network and fallback to 4G networks(indicated as ⚠in column 1), exposing potential vulnerabilities in the 4G domain [11].

VIII. Related Work

Attacks on Cellular Protocol: 3GPP specification related flaws have resulted in several vulnerabilities such as LTRACK [18], SigUnder [19] and others [20]–[23]. These

TABLE VIII
IMPACT OF 5G VULNERABILITIES ***V5-V7*** ON CPE ROUTER TARGETS.
✓- CPE PERFORMS ACTION; ✗- CPE ***DOES NOT*** PERFORM ACTION; ⚠- CPE ***POORLY*** PERFORMS ACTION.

CPE Target	**Vulnerabilities Impact (V5-V7)**			
	Internet Connection?	Access to Conf. Webpage?	Recovers Connection?	Requires User Intervention?
FT980-WW	V5,V6: ✗ | V7: ⚠	V5,V6: ✗ | V7: ✓	V5,V6: ✓ | V7: ✗	V5,V6: *N.A* | V7: **Reboot**
Raspbery Pi 3B+ & OpenWrt 22.03.4	V5,V6: ✗ | V7: ⚠	V5,V6: ✓| V7: ✓	V5,V6: ⚠ | V7: ✗	V5-V7: **Reboot**

TABLE IX
COMPARISON BETWEEN 5GHOUL AND OTHER RELATED APPROACHES.
●: FULLY CONSIDERED, ❍: NOT CONSIDERED, ◗: PARTIALLY CONSIDERED.
FOR THE LAST COLUMN, ❍ CAPTURES THE ABSENCE OF COTS TARGETS.

	5G-NR Support	Data Link Support	Stateful	Basestation / UE	OTA Exploitation	Target Support
Berserker [4]	◗	❍	❍	● / ●	◗	❍
DoLTEst [6]	◗	❍	●	❍ / ●	●	●
LTEFuzz [5]	◗	❍	◗	● / ●	●	●
5Greplay [2]	●	❍	●	● / ●	◗	❍
FIRMWIRE [7]	◗	❍	❍	❍ / ●	●	◗
BASESPEC [8]	◗	❍	❍	❍ / ●	❍	◗
BaseSAFE [9]	❍	❍	❍	❍ / ●	❍	◗
[3]	●	●	●	● / ❍	●	❍
5Ghoul	●	●	●	❍ / ●	●	●

attacks may allow unauthorized remote access or launch remote code execution. 5GHOUL approach is orthogonal to these efforts. Specifically, these prior works involve extensive manual effort (e.g., reverse engineering and code inspection) to uncover specific, yet critical attack vector. In contrast, 5GHOUL finds security issues directly in the 5G edge devices by learning the protocol states automatically. Additionally, the attack discovery within 5GHOUL is supported by our novel over-the-air fuzzing strategy and the subsequent generation of exploits.

Cellular Baseband Fuzzing: Despite advancements in cellular baseband fuzzing in the last few years [2], [4], [6]–[9], several challenges persist in terms of making such fuzzing general, comprehensive (in terms of supporting all network layers) and realistic to target arbitrary commercial hardware. Table IX compares 5GHOUL approach with respect to some recent approaches. Majority of the works target LTE protocol implementation, but mentions the potential applicability of these works for 5G implementation (exemplified by the ◗ in Table IX). Specifically, LTEFuzz [5] demands commercial logs for the creation of fuzzing test cases and requires additional effort for test case creation on 5G use cases. Commercial logs are unlikely to be available for data link layer frames. In contrast, 5GHOUL approach is much simpler and works completely at the user end without requiring access to commercial logs. Moreover, 5GHOUL is the first fuzzing approach to fully control all packets down to the data link layer for commercial UE targets. Even though our prior work [3] achieved the capability of controlling data link frames, it was only achieved for software emulation. Indeed, in our evaluation, we show that controlling data link frames are crucial for finding the vulnerabilities ***V4***, ***V5*** (high severity) and ***V10***. Thus, all other approaches except 5GHOUL, as shown in Table IX, will fail to discover these vulnerabilities.

Emulation-based Analysis: Approaches based on reverse engineering [8] and/or emulation [7], [9] are promising direction for efficient static and dynamic analysis of baseband implementations. However, such approaches involve manual effort to extend support for new or additional baseband architecture exemplified by the ◗ in Table IX. Not only such effort demands reverse engineering (if at all possible), but also requires effort in replicating the crash over-the-air [7]. In contrast, 5GHOUL does not require *any additional manual effort* to support arbitrary baseband architectures for 5G. Moreover, the 5GHOUL fuzzing discovers crashes directly on the device implementation. As a result, the crash reproduction is also automated and generic as discussed in Section III-C. Moreover, emulation may not exhibit the exact timing behavior found on the edge devices, as emulation typically involves approximation of modem hardware. In contrast, 5GHOUL does not suffer from such approximation as it runs all tests and exploits in situ.

Over-the-air Exploitation and Stateful Fuzzing: In contrast to 5GHOUL, several prior works [8], [9] do not support exploitation over-the-air, whereas such support is unclear for 5Greplay [2] and Berserker [4] from the description. Moreover, even though some prior works [6], [10] support commercial UE targets and over-the-air exploitation, they require extensive manual effort to create the test cases. Additionally, DoLTEst [6] does not explicitly target 5G. Finally, only a few prior works consider the LTE/5G protocol states systematically to generate valid packet sequences [2], [6]. In contrast, 5GHOUL automatically constructs the protocol state machine to guide the fuzzing process and manipulate the 5G packet sequences over-the-air.

Model-based Approaches: Alternative approaches such as 5GReasoner [1] aims to discover the vulnerabilities in 5G protocol by modeling the protocol behavior and using verification tools. Such an approach does not target vulnerabilities in real implementation of 5G stack (e.g., 5G UE). Moreover, this work requires the protocol state modeling manually. In contrast, 5GHOUL targets vulnerabilities on commercial edge devices. This is accomplished via a stateful 5GHOUL fuzzing, where even the protocol states are automatically generated from live communication traces.

Conventional Greybox Fuzzing: Most greybox fuzzers instrument code to optimize code coverage. Such is not possible for commercial and closed wireless stacks. Moreover, classic greybox fuzzers aim to generate a single input leading to crashes. For wireless protocols, often a sequence of packets with strict timing constraints triggers crashes. Even though stateful greybox fuzzing has been developed in recent years [24], they are not applicable to fuzz closed

wireless stacks due to the required instrumentation. Moreover, such work demands access to the protocol source code variables to construct the state machine. Such is impractical for cellular fuzzing, especially due to unavailability of proprietary wireless protocol implementation. In contrast to the aforementioned works, 5GHOUL is fully capable to test arbitrary 5G stacks without requiring any access to the source code or emulation effort.

Notably, 5GHOUL*is the first fuzzer that targets all 5G NR protocols down to the MAC. Moreover, it supports fuzzing arbitrary COTS over-the-air and without inspection to their firmware*. The approach taken by 5GHOUL is shown to be practical via the exposure of new implementation vulnerabilities across major vendors.

IX. Conclusion

In this paper, we propose 5GHOUL, a framework to automatically discover and replicate security vulnerabilities on arbitrary 5G UE devices. Compared to prior works, 5GHOUL brings some concrete advantages: *(i)* 5GHOUL is the first approach to have full control on all downlink packets down to the data link layer and *(ii)* 5GHOUL approach can be employed out-of-the-box for any COTS 5G UEs. This opens up significant opportunities to automatically and comprehensively test all 5G layer 2 and layer 3 protocol implementations at scale. We have demonstrated that our approach is practical: specifically, 5GHOUL discovered ten previously unknown vulnerabilities on COTS UEs that use cellular modems from major vendors. Moreover, 5GHOUL shows extensibility beyond the discovery of crashes and hangs: it can faciliate discovery of existing as well as new downgrade attacks. For reproduction and advance research activities in 5G cellular fuzzing, 5GHOUL source code and exploits are available at the following URL:

https://github.com/asset-group/5ghoul-5g-nr-attacks

Acknowledgment

This research is partially supported by MOE Tier 2 grant (Award number MOE-T2EP20122-0015), National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research & Development Program (Award number FCP-SUTD-RG-2022-017) and National Research Foundation, Singapore, under its National Satellite of Excellence Programme "Design Science and Technology for Secure Critical Infrastructure: Phase II" (Award No: NRF-NCR25-NSOE05-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the respective funding agencies.

References

[1] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 669–684.

[2] Z. Salazar, H. N. Nguyen, W. Mallouli, A. R. Cavalli, and E. M. de Oca, "5Greplay: A 5G Network Traffic Fuzzer – Application to Attack Injection," in *The 16th International Conference on Availability, Reliability and Security*, Aug. 2021, pp. 1–8.

[3] M. E. Garbelini, Z. Shang, S. Chattopadhyay, S. Sun, and E. Kurniawan, "Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*. Rio de Janeiro, Brazil: IEEE, Dec. 2022, pp. 86–92.

[4] S. Potnuru and P. K. Nakarmi, "Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G," in *2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2021, pp. 295–300.

[5] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1153–1168.

[6] C. Park, S. Bae, B. Oh, J. Lee, E. Lee, I. Yun, and Y. Kim, "DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices," in *USENIX Security Symposium*, 2022.

[7] G. Hernandez, M. Muench, D. Maier, A. Milburn, S. Park, T. Scharnowski, T. Tucker, P. Traynor, and K. Butler, "FirmWire: Transparent Dynamic Analysis for Cellular Baseband Firmware," in *Proceedings 2022 Network and Distributed System Security Symposium*. San Diego, CA, USA: Internet Society, 2022.

[8] E. Kim, D. Kim, C. Park, I. Yun, and Y. Kim, "BaseSpec: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols," in *Proceedings 2021 Network and Distributed System Security Symposium*. Virtual: Internet Society, 2021.

[9] D. Maier, L. Seidel, and S. Park, "BaseSAFE: Baseband SAnitized Fuzzing through Emulation," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, Jul. 2020, pp. 122–132.

[10] E. Bitsikas, S. Khandker, A. Salous, A. Ranganathan, R. Piqueras Jover, and C. Pöpper, "Ue security reloaded: Developing a 5g standalone user-side security testing framework," in *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 121–132. [Online]. Available: https://doi.org/10.1145/3558482.3590194

[11] M. Kotuliak, S. Erni, P. Leu, M. Röschlin, and S. Čapkun, "{LTrack}: Stealthy Tracking of Mobile Phones in {LTE}," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1291–1306.

[12] S.-Y. Lien, S.-L. Shieh, Y. Huang, B. Su, Y.-L. Hsu, and H.-Y. Wei, "5G New Radio: Waveform, Frame Structure, Multiple Access, and Initial Access," *IEEE communications magazine*, vol. 55, no. 6, pp. 64–71, 2017.

[13] S. Potnuru, "Fuzzing Radio Resource Control Messages in 5G and LTE Systems: To Test Telecommunication Systems with ASN. 1 Grammar Rules Based Adaptive Fuzzer," 2021.

[14] D. Klischies, M. Schloegel, T. Scharnowski, M. Bogodukhov, D. Rupprecht, and V. Moonsamy, "Instructions unclear: Undefined behaviour in cellular network specifications," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 3475–3492. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/klischies

[15] M. E. Garbelini, C. Wang, and S. Chattopadhyay, "Greyhound: Directed Greybox Wi-Fi Fuzzing," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 817–834, Mar. 2022.

[16] I. Karim, F. Cicala, S. R. Hussain, O. Chowdhury, and E. Bertino, "Opening Pandora's Box through ATFuzzer: Dynamic Analysis of AT Interface for Android Smartphones," in *Proceedings of the 35th Annual Computer Security Applications Conference*, ser. ACSAC '19. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 529–543.

[17] "Vodafone Unveils Prototype 5G Network Built on a Raspberry Pi Computer."

[18] M. Kotuliak, S. Erni, P. Leu, M. Röschlin, and S. Čapkun, "{\vphantom}LTrack\vphantom{}: Stealthy Tracking of Mobile Phones in {\vphantom}LTE\vphantom{}," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1291–1306.

[19] N. Ludant and G. Noubir, "SigUnder: A stealthy 5G low power attack and defenses," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Abu Dhabi United Arab Emirates: ACM, Jun. 2021, pp. 250–260.

[20] D. Fraunholz, R. Schörghofer-Vrinssen, H. König, and R. Zahoransky, "Show Me Your Attach Request and I'll Tell You Who You Are: Practical Fingerprinting Attacks in 4G and 5G Mobile Networks," in *2022 IEEE*

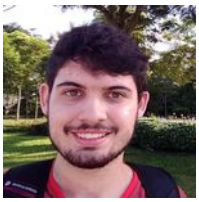
Conference on Dependable and Secure Computing (DSC), Jun. 2022, pp. 1–8.

[21] E. Bitsikas and C. Pöpper, "Don't hand it Over: Vulnerabilities in the Handover Procedure of Cellular Telecommunications," in *Annual Computer Security Applications Conference*, ser. ACSAC '21. New York, NY, USA: Association for Computing Machinery, Dec. 2021, pp. 900–915.

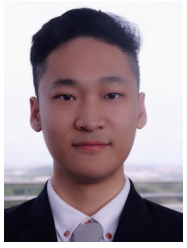
[22] S. Bae, M. Son, D. Kim, C. Park, J. Lee, S. Son, and Y. Kim, "Watching the Watchers: Practical Video Identification Attack in {LTE} Networks," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1307–1324.

[23] B. Karakoc, N. Fürste, D. Rupprecht, and K. Kohls, "Never Let Me Down Again: Bidding-Down Attacks and Mitigations in 5G and 4G," 2023.

[24] J. Ba, M. Böhme, Z. Mirzamomen, and A. Roychoudhury, "Stateful Greybox Fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3255–3272.



Matheus E. Garbelini graduated with a PhD from Singapore University of Technology and Design (SUTD). His research interests include Wireless Security, cyber-physical Systems and IoTs and embedded systems. Matheus is known for the discovery of a collection of wireless vulnerabilities such as Sweyntooth, Braktooth and 5Ghoul. For his discovery of SweynTooth, he was recognized as an outstanding research contributor by Medtronic Inc.



Zewen Shang graduated with a Bachelor's degree in Computer Science and Cybersecurity (Merit with Distinction) from the Singapore University of Technology and Design (SUTD). Currently, he is pursuing a PhD at SUTD's ASSET Research Group, under the supervision of Professor Sudipta Chattopadhyay. His research focuses on 5G security, IoT security, and vulnerability analysis. He was awarded over 5,000 USD in funding for his work on high-severity 5G vulnerabilities in MediaTek's systems.

Shijie Luo graduated with a Bachelor's degree in Computer Science from the Singapore University of Technology and Design (SUTD). Currently, he is pursuing a PhD at SUTD's ASSET Research Group. His research focuses on 5G security, IoT security, and vulnerability analysis.

Sudipta Chattopadhyay is an Associate Professor in the Information Systems Technology and Design pillar, Singapore University of Technology and Design, Singapore. His general research interests are in the broad area of cybersecurity, including but not limited to safety & security for AI, wireless technologies and the Internet of Things (IoTs). Together with his student, he discovered SweynTooth, BrakTooth and 5Ghoul, families of Bluetooth and 5G NR vulnerabilities affecting billions of devices worldwide. His research has been featured in Channel News Asia, WIRED, PC Magazine and Hacker News, among others. His discovery has also led to cybersecurity alerts from government regulatory agencies including the Cyber Security Agency (Singapore), the Department of Homeland Security (DHS) and the Food and Drug Administration (FDA). For his discovery of SweynTooth, he was recognized as an outstanding research contributor by Medtronic Inc. He is an Associate Editor of ACM Computing Surveys and IEEE Transactions on Software Engineering.

Sumei Sun is the Executive Director of the Institute for Infocomm Research (I2R), A*STAR, Singapore. She is also holding a joint appointment with the Singapore Institute of Technology, and an adjunct appointment with the National University of Singapore, both as a full professor. Her current research interests are in next-generation wireless communications, cognitive communications and networks, industrial internet of things, communications-computing-control integrative design, joint radar-communication systems, and signal intelligence. Sumei has a strong passion in industry-relevant research and technology creation. She has authored and co-authored 300 technical papers and received three best paper awards. She is the inventor/co-inventor of over thirty patented technologies, with most of them licensed to industries.

Ernest Kurniawan is (Senior Member, IEEE) was a Post-Doctoral Fellow at the Department of Electrical Engineering, Stanford University, Stanford, CA, USA, from 2011 to 2013. He is currently a Principal Scientist with I2R, A*STAR, Singapore. His research interests include signal processing, information theory, integrated sensing and communications, and artificial intelligence for wireless communication.