

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Understanding Implicit Trust Errors in Core Carrier Networks through Multi-Agent Flaw Discovery and Analysis

Ziyu Lin, Ziting Wang, Xinfeng Li, Wei Dong, and XiaoFeng Wang, *Nanyang Technological University*

https://www.usenix.org/conference/usenixsecurity26/presentation/lin-ziyu

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Understanding Implicit Trust Errors in Core Carrier Networks through Multi-Agent Flaw Discovery and Analysis

Ziyu Lin, Ziting Wang, Xinfeng Li, Wei Dong, XiaoFeng Wang
Nanyang Technological University

Abstract

Cellular core networks (CNs) are critical infrastructure, yet their internal security model has historically relied on physical isolation: interfaces between core components often operate within an assumed trust zone. As CNs transition to cloud-native deployments, this assumption weakens, expanding the attack surface and enabling external adversaries to reach previously internal interfaces. From a root-cause analysis of security flaws reported in GitHub issues for open-source CN implementations, we found a recurring pattern of *blind trust* among CN components. Components may omit syntactic validation, fail to enforce semantic invariants, or allocate resources without checking availability. Once internal interfaces become reachable, these weaknesses can lead to severe impacts such as denial of service and session hijacking. We call these vulnerabilities *implicit trust errors* (*iTrue*).

To detect iTrues and understand their security impacts, we designed *iFinder*, an LLM-driven multi-agent system that summarizes known flaws, distills them into detection patterns, and applies them to discover new iTrues in CN implementations. To suppress hallucinations produced by large language models (LLMs), we built an innovative strategy that cross-checks both 3GPP specifications and CN code to capture existing protection missed by the agents. Further, we developed a technique that uses LLMs to generate proof-of-concept (PoC) exploits for potential iTrues and iteratively refine the PoCs by automatically executing them against CN implementations and analyzing results. Running iFinder on seven prominent open-source CN implementations, we discovered 84 previously unknown vulnerabilities. Among them, 83 have already been confirmed and 81 have been assigned CVEs. Importantly, a session-hijacking flaw has been confirmed on real-world commercial 5G core networks. Our findings highlight the pervasiveness of iTrue risks across the cellular core networks and the urgent need for elevated protection within the original trust domains. We plan to make iFinder publicly available to help enhance the security of cellular core networks.

1 Introduction

Cellular core networks are critical infrastructure underpinning everyday communication for billions of users. Vulnerabilities in these systems can cause large-scale service disruptions and compromise user privacy, making their security of paramount importance [1]. Historically, core network (CN) security has relied on physical isolation: internal interfaces between network functions operate within a trusted zone, often deployed without strong encryption or mutual authentication [2]. This trust model, however, is increasingly fragile. As operators migrate CNs to cloud-native deployments, the security perimeter shifts from physically isolated private infrastructure to shared cloud environments with complex network topologies [3]. Prior work has demonstrated that attackers can reach CN interfaces through cloud misconfigurations or by abusing GTP-U tunnels to inject arbitrary traffic [4]. These architectural changes expose internal interfaces to external adversaries, invalidating the implicit-trust assumption that has long underpinned core network security.

Vulnerabilities in CN. Given these emerging attack surfaces, there is an urgent need for systematic approaches to uncover vulnerabilities in core network implementations. Yet, prior work has largely focused on analyzing interactions either between user equipment (UE) and CN components through the Non-Access Stratum (NAS) [5–9], or between base stations and the CN via protocols such as the Next-Generation Application Protocol (NGAP) [7, 9] and the GPRS Tunnelling Protocol User Plane (GTP-U) [9]. To our knowledge, no systematic study has ever been done to look into security risks within the CN itself, particularly those arising from inter-component communication that may become externally exposed in cloud deployments, as discussed earlier.

On the other hand, industry has continued to report security-critical flaws affecting the internal CN [10, 11]. For example, Black Duck's security report [10] reveals that Open5GS's PFCP implementation lacks proper bounds checking on IE length fields, incurring denial of service (DoS) risks. Additional vulnerabilities have surfaced in GitHub Issues, often

involving open-source 5G implementations and simulators such as Open5GS, free5GC, and OpenAirInterface (OAI). Many of these findings, which are frequently uncovered using fuzzing tools such as Defensics [12], are only isolated samples from a much larger CN security problem space, but they offer a clear glimpse into the kinds of threats these networks face. What remains missing, however, is a systematic study that connects these dots, revealing their underlying relations and, more broadly, clarifying what can go wrong in the CN.

In our research, we take a first step toward understanding CN security by analyzing vulnerabilities reported through GitHub Issues in open-source CN implementations. Specifically, we collected all security issues from the GitHub repositories of Open5GS and free5GC, and conducted a systematic root-cause analysis using Claude Code. We designed a Chain-of-Thought (CoT) procedure to guide automated root-cause discovery (e.g., identifying assertion-triggered crashes due to session pool exhaustion) and further clustered these flaws according to their underlying causes. Our analysis revealed a striking similarity across the reported issues. All can be grouped into three broad categories: missing validation of message syntax, missing validation of message semantics, and failure to check resource availability. Collectively, these vulnerabilities reflect a fundamental pattern of *blind trust*: CN components accept internal messages as inherently trustworthy, an assumption rooted in the traditional belief that the CN is walled off from untrusted environments, but one that no longer holds in cloud-native deployments. We call them *implicit trust errors*, or *iTrue*.

Finding iTrues. To systematically discover iTrues, we developed a multi-agent system, *iFinder*, that summarizes the root causes within each cluster, generalizes these summaries into reusable detection patterns, and applies the patterns to identify additional iTrue flaws in CN implementations. Achieving this required addressing a key challenge: distilling cluster-level root causes into detection patterns that are both precise and general. The patterns must capture the essential vulnerability logic while remaining broad enough to cover unseen instances across different protocols and programming languages. To this end, we engineered prompts to emphasize language-agnostic and protocol-agnostic patterns, explicitly identifying key elements and the expected operations over them. We then use these patterns to guide the agent in scanning CN implementations and extracting iTrue candidates (Section 4.2). For example, *iFinder* abstracts reports of missing Information Element (IE) checks into a protocol-agnostic pattern: locating downstream uses of an IE, then verifying that its presence and validity are validated earlier in message handling. We pass this pattern to the agent to drive LLM-based backward analysis and flag potential iTrues.

Moreover, naively applying these patterns to CN implementations can yield many false positives, largely due to hallucinations and incomplete reasoning when the underlying LLM lacks sufficient context. In particular, we observed that when only a local code fragment is provided, the agent may flag it as an iTrue even though the required validation is performed elsewhere (e.g., an earlier stage of the processing pipeline) outside the snippet. At a deeper level, the problem is that protocols typically comprise multiple procedures, each spanning several states. Within a given procedure, security-critical checks may be performed in earlier states rather than in the flagged snippet, so validating an iTrue often requires analyzing the preceding states instead of a single snippet. However, identifying the code for those preceding states is itself challenging. Feeding the entire codebase to an LLM to "search" for relevant checks is often both inefficient and unreliable in large, stateful CN implementations. To mitigate this issue, we developed a novel *code-specification cross-checking* technique. The key idea is to map an iTrue candidate to the protocol procedure it implements, as defined in the relevant specifications, and then map the entire procedure back to the codebase to recover the full execution context. This expanded, procedure-level view enables the agent to determine whether the necessary validation and resource checks are actually enforced, substantially reducing false positives when assessing whether a finding is indeed an iTrue.

The final step of *iFinder* is to generate proof-of-concept (PoC) exploits to confirm the presence of an iTrue flaw. To this end, the agent follows an iterative test-and-refine loop: it deploys a simulator-based testbed to execute PoCs generated by the agent, analyzes execution logs to diagnose failures and extract actionable feedback, and then uses this feedback to revise the PoC. This process repeats until the vulnerability is successfully exploited or a testing budget is exhausted.

Running iFinder on seven prominent open-source CN implementations, we uncovered 84 previously unknown vulnerabilities in the Packet Forwarding Control Protocol (PFCP) and GPRS Tunnelling Protocol Control Plane (GTP-C) protocols. Among them, 83 have been confirmed by developers and 81 have been assigned CVEs. These vulnerabilities have significant security implications, including DoS and session hijacking. Moreover, we evaluated these vulnerabilities on two real-world commercial 5G core networks using PoCs derived from open-source platforms, identifying two DoS vulnerabilities and one session-hijacking vulnerability. Both DoS vulnerabilities affect the same commercial 5G core network, and we have received a CVE assignment for one of them (CVE-2026-8232). The session-hijacking flaw was found in both commercial 5G core networks: one vendor has since fixed the issue, with a CVE assigned (CVE-2026-8233), while the other is still in the remediation process. At a high level, this flaw allows an attacker to inject a PFCP Session Modification Request, causing the User Plane Function (UPF) to forward the victim UE's uplink traffic to the attacker. We include this vulnerability as a case study in Section 6.2.

Contributions. We list our following contributions below:

• *New findings*. We present the first systematic root-cause analysis of vulnerabilities reported in open-source CN imple-

mentations, showing that they largely arise from *implicit trust* among internal CN components.

• *New intelligent techniques*. We developed *iFinder*, the first multi-agent system for automated vulnerability discovery in carrier core networks. Our techniques address key technical barriers, and an ablation study demonstrates their effectiveness in reducing hallucinations.

• *Implementation and evaluation*. We implemented *iFinder* and evaluated it on seven widely used open-source CN implementations, uncovering **84** previously unknown vulnerabilities and **81** assigned CVEs, including a session-hijacking flaw confirmed on real-world commercial 5G core networks.

2 Background

2.1 Cellular Network Overview

A cellular network consists of user equipment (UE), a radio access network (RAN), and a core network (CN) that implements control-plane and user-plane functions. The RAN is composed of eNodeBs (eNBs) in 4G networks [13] and gNodeBs (gNBs) in 5G networks [14]. Cellular control-plane procedures are typically initiated by the UE and transmitted through the RAN before entering the core network as protocol messages. In 4G systems, this signaling traverses the S1 interface, while in 5G systems it traverses the N2 interface [14, 15]. These access-facing interfaces form the primary boundary between the external RAN and the internal CN. Although this boundary is often treated as trusted in operational deployments, the signaling originates from UE interactions and directly influences CN state. As a result, the RAN-CN boundary has been a recurring focus in security analyses of cellular systems [6, 9, 16].

The architecture of the core network has evolved significantly from 4G to 5G, with important implications for trust and attack surface. In the 4G Evolved Packet Core, control-plane logic is largely centralized in the Mobility Management Entity (MME) [17], while the Serving Gateway (SGW) and Packet Data Network Gateway (PGW) anchor user-plane traffic and connectivity to the Internet. Although control and user planes are conceptually separated, their protocol logic and session state are closely coupled across a small set of components. The 5G Core further decomposes these roles into the Access and Mobility Management Function (AMF), the Session Management Function (SMF), and the User Plane Function (UPF), with AMF handling access and mobility control, SMF managing session state, and UPF performing packet forwarding [14]. These functions interact through a larger set of internal interfaces, including N2, N4, and N9. While this decomposition improves flexibility and scalability, it also relies more heavily on implicit trust between internal network functions [18]. In practice, internal interfaces are often deployed without strong cryptographic protection to prioritize performance [2, 19], which increases the impact of protocol misuse and configuration errors inside the core network as the architectural gap between the cellular core and the Internet continues to close [4, 18].

2.2 Core Network Protocol

Core network behavior is governed by a small set of protocols that manage session establishment, state propagation, and packet forwarding. Among these, Packet Forwarding Control Protocol (PFCP) and GPRS Tunnelling Protocol (GTP) are central to how control-plane decisions are translated into user-plane behavior [17, 20]. PFCP governs control-plane interaction between the SMF and the UPF, while GTP supports both control signaling through GTP-C and user-plane tunnelling through GTP-U across 4G and 5G cores [17, 21]. These protocols do not operate as isolated message exchanges; instead, they establish long-lived session contexts shared across multiple network functions. System behavior therefore depends on protocol execution paths that span multiple messages and handlers rather than on individual packets [4, 18].

PFCP operates on the N4 interface and is responsible for installing forwarding rules in the user plane. A PFCP association is first established to exchange node identifiers, after which session-specific state is created through PFCP session establishment procedures [20]. Each session is identified by a Session Endpoint Identifier (SEID) that is referenced in subsequent operations such as modification and deletion. Because PFCP directly links control-plane signaling to concrete forwarding resources, message-level validity alone does not guarantee safe system behavior. Errors in session state or control logic can immediately lead to traffic hijacking or denial-of-service [4, 22]. GTP underpins user-plane communication and separates control signaling from data forwarding through GTP-C and GTP-U [21, 23]. GTP-U remains the primary mechanism for tunnelling user traffic across the core network [4, 21]. GTP relies on Tunnel Endpoint Identifiers (TEIDs) to associate packets with tunnels, with each tunnel being unidirectional and identified by a receiver-assigned identifier. Bidirectional communication therefore requires coordinated tunnel setup through a sequence of control-plane messages [17, 21]. GTP messages also carry many Information Elements (IEs), such as access point settings and quality-of-service (QoS) parameters, which must be parsed and validated consistently across different handlers.

2.3 Agents and LLMs

The complexity of cellular core networks has motivated the use of automated analysis techniques such as fuzzing and static analysis. More recently, large language models have been explored as an additional tool for software analysis [24]. However, these approaches face fundamental limitations when applied to core network software due to a mis-

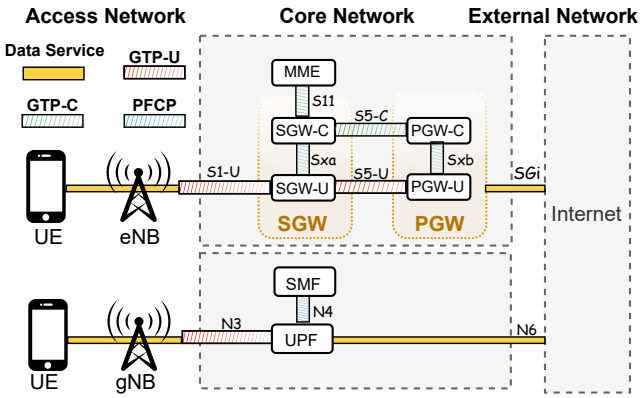

Figure 1: Cellular network architectures from 4G to 5G.

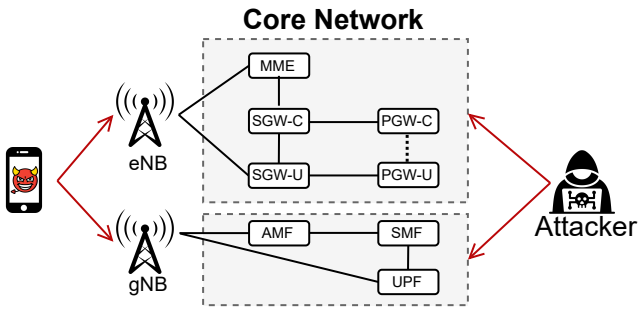

Figure 2: Threat model. Attackers are assumed to have access to internal core network interfaces and can send protocol messages that violate implicit trust assumptions, without compromising cryptographic protections.

match between their analysis models and the stateful, procedural nature of cellular protocols. Coverage-guided fuzzers are effective at exploring input-handling logic but struggle to reach protocol states that require multiple rounds of valid message exchanges and coordinated state updates across components [25–27]. Static analysis tools face similar challenges, as control logic is often distributed across asynchronous handlers and shared session contexts, which makes precise tracking of execution paths difficult and leads to false positives or missed flows [16, 28].

These limitations motivate the use of agent-based workflows as an orchestration mechanism rather than as a replacement for existing tools. In this setting, an agent coordinates protocol-aware reasoning, staged analysis, and runtime feedback to align analysis steps with protocol procedures and session evolution. By incorporating feedback from concrete executions and analyzing logs to classify triggered impacts [29, 30], agent-based workflows can bridge abstraction gaps that limit traditional techniques. This approach, leveraging Chain-of-Thought reasoning [31] and specification-grounded cross-checking [14, 20], is particularly relevant for cellular core networks, where security-relevant behavior emerges from the interaction between complex protocol logic and operational trust assumptions [29, 32].

2.4 Threat Model

We assume an adversary can obtain the IP address of core-network components. Such information may be derived from public documentation, passive enumeration, or active scanning [4]. As shown in Figure 2, we consider two classes of attackers, distinguished by their network vantage point and capabilities. The two attackers aim to disrupt cellular network services, which can result in DoS and session hijacking.

• *Internet attacker.* The attacker is located outside the cellular core network (e.g., on the public Internet) and does not control any core-network component. However, the attacker can craft and send arbitrary PFCP or GTP-C messages. In cloud-native deployments, misconfigurations may inadvertently expose internal interfaces to external networks. In this case, an attacker can send malformed PFCP or GTP-C messages directly to core components. The presence of such risks in real-world cloud environments has also been reported by prior research [4].

• *Malicious UE.* The attacker acts as a malicious UE using a commercial smartphone with a valid Universal Subscriber Identity Module (USIM). After completing the standard registration procedure, the UE obtains a legitimate IP address from the network. The attacker then injects carefully crafted payloads into the uplink data stream. The gNB encapsulates this traffic into GTP-U and forwards it through the UPF. These payloads may carry PFCP or GTP-C messages. By exploiting protocol tunnelling and network boundary bridging, the attacker smuggles crafted PFCP or GTP-C messages inside GTP-U messages so that, absent strict boundary enforcement, they cross the boundary and are delivered to and parsed by core-network components [33, 34]. We further confirmed that both protocol tunnelling and network boundary bridging are feasible against five of the seven evaluated open-source CN implementations.

3 Understanding Weaknesses in CNs

Under our threat model, attackers can reach core network components from outside the trusted boundary, as shown in prior work [4], invalidating the traditional isolation assumption that internal components only receive well-formed, benign messages. As a result, we discovered that implicit-trust-driven minimal validation becomes an exploitable attack surface. We made this observation through an empirical analysis of security issues reported in CN implementations' GitHub repositories. Specifically, we extracted the root cause of each issue, clustered issues with similar causes, and distilled recurring vulnerability patterns. We then utilized these patterns to guide an LLM in identifying previously unseen

iTrue vulnerabilities across the codebases.

GitHub security issues in CN. To study the relations among CN-related known vulnerabilities, we curated a dataset of previously reported GitHub security issues in CN. Specifically, we gathered 22 security issues from two widely used open-source CN implementations, Open5GS and free5GC. These reports follow a largely consistent template, typically including a title and bug description, the affected version and components, steps to reproduce, and supporting artifacts such as logs, expected and observed behavior. These fields provide context to localize the faulty checks, enabling systematic root-cause analysis across issues.

Root-cause analysis and clustering. Over the reported security issues, we performed a root-cause analysis using Claude Code to extract three elements from each issue: the message(s) that introduce security risk (the *trigger*), the program location in CN implementations where the risk manifests (the *failure point*), and the underlying vulnerability (the *flaw*). We then summarized these elements and their relations into a *root-cause statement*. For example, in `ogs_pfcp_handle_create_urr()`, an out-of-bounds read (failure point) can be triggered by a Session Establishment Request that omits mandatory fields (e.g., Downlink Volume) in the Volume Threshold IE (trigger), exposing a missing mandatory-field check (flaw) and causing a crash. Although these issues span different messages and code paths, many root-cause statements share the same structure: the implementation consumes untrusted message fields without validating mandatory IEs, lengths, or bounds. For instance, in `ogs_pfcp_extract_node_id()`, the code copies Node ID data into a fixed-size buffer using a length taken directly from an unverified IE length field in an Association Setup Request, due to missing IE-length validation. While the specific IE types and messages differ, both this case and the above example stem from the same underlying cause, that is, failure to verify the format and completeness of incoming messages. Guided by these similarities, we leveraged an LLM to infer relations among issues and obtained three distinct clusters.

Analysis and findings. Looking into these clusters, we found that CN components often forgo due diligence in validating message format, message semantics, and resource availability, and instead tend to *blindly* act on messages received from internal peers:

• *Missing syntactic validation*: 12 issues involve malformed messages being accepted without checking the presence of mandatory fields or enforcing basic format and length constraints. For example, on receiving a Session Establishment Request containing a Volume Threshold IE, the UPF *blindly* proceeds to copy volume fields based on presence flags, without verifying that all required fields are included. As a result, one can craft a truncated Volume Threshold IE to trigger an out-of-bounds read and crash the UPF.

• *Missing semantic validation*: 8 issues involve messages being accepted without validating semantic constraints or enforcing state-dependent invariants. For example, upon receiving a Session Establishment Request containing an F-TEID value of `0`, the UPF *blindly* uses the F-TEID to look up the corresponding PDR without checking whether the value falls within a valid range. As a result, an attacker can craft a PFCP message with an out-of-range F-TEID to trigger an assertion failure and crash the UPF.

• *Missing checking on resource availability*: 2 issues involve risky resource-handling operations where the recipient allocates requested resources without checking availability. For example, upon receiving a PFCP Session Establishment Request, the UPF *blindly* allocates buffers from its internal pool without verifying that sufficient resources remain. As a result, an attacker can repeatedly send PFCP messages to drain the buffer pool, ultimately triggering a segmentation fault and crashing the UPF.

Across these clusters, the common denominator is the *implicit trust* that a CN component places in messages from its internal peers, rooted in the assumption that the CN is well protected and walled off from untrusted environments. This assumption becomes increasingly fragile as CNs move to the cloud: as discussed earlier, new attack surfaces emerge through exposed CN components and loosely protected tunnel interfaces (Section 2.4). Consequently, we refer to the risks arising from this legacy "blind faith" as *implicit trust errors* (*iTrue*). This observation inspired us to summarize the defining patterns of iTrues to guide the discovery of previously unknown flaws in CN implementations. Finally, although these known issues are largely memory-safety bugs, in part because they were most likely detected by fuzzing, *we show later that the same patterns enable the discovery of more subtle and more severe vulnerabilities, including those that enable session hijacking* (Section 6.2).

4 Finding iTrues with iFinder Agents

Given the repeated appearance of iTrue flaws, we hypothesize that they are not isolated incidents but recurring weaknesses across core-network (CN) implementations. We therefore aim to distill the distinctive characteristics of known iTrue instances into a feature set that can guide an LLM to detect iTrues in diverse CN codebases and programming languages. However, we found that naively applying LLM inference is unreliable: it frequently produces hallucinated findings, leading to low precision and inconsistent results. Specifically, in our preliminary evaluation, direct pattern-guided LLM inference yielded only 8 true positives but 56 false positives and 14 false negatives (Precision 28.205%, Recall 36.364%, F1 31.769%), indicating severe context-induced hallucinations. Most false positives arise when required validations are implemented earlier in the same protocol procedure, but are invisible in the local snippet. In contrast, many false

negatives are due to the large search space. Consequently, when scanning an entire codebase, the LLM fails to focus on the right protocol-handling paths and misses the vulnerable snippet.

To address these challenges, we developed a multi-agent system, *iFinder*, which integrates LLM-based backward analysis, code-specification cross-checking, and proof-of-concept (PoC) exploit generation and refinement to improve detection quality. These techniques and the design of the whole system are elaborated in the rest of the section.

4.1 Overview

Idea. To reduce false negatives, we focus on likely failure points, such as operations that depend on message content (e.g., reading or copying based on a length field carried in a message), and then trace back to the *trigger*, such as the message-handling logic, to determine whether the expected validation is present. We also observe that a major source of hallucinations is missing protocol context, which leads the LLM to fixate on irrelevant code segments and report vulnerabilities that are infeasible along real execution paths. This issue is especially common in carrier protocols, where a single procedure spans multiple states and security-critical checks are often performed in earlier states rather than in the flagged snippet (see Figure 5). To mitigate this, we propose *code-specification cross-checking*: grounding each vulnerability candidate in the corresponding procedure defined by 3GPP specifications, recovering the prerequisite messages, and then mapping these messages to their handlers in the codebase by searching, so the agent can reason about whether the candidate is still feasible. This procedure-level context substantially reduces context-induced false positives without requiring the LLM to reason over the entire codebase.

To further reduce false positives, we use the LLM to generate proof-of-concept (PoC) exploits for identified iTrue candidates. A key challenge is that LLM-generated PoCs may fail to trigger a flaw due to implicit runtime constraints that are not evident from static code alone. Our idea is *feedback-aware refinement*, which frames exploitation as an iterative process: the agent executes the PoC against a testbed and analyzes runtime logs. If the vulnerability is triggered, the agent confirms the finding; otherwise, it diagnoses the failure and extracts actionable feedback to refine the PoC accordingly. This test-and-refine loop grounds PoC generation in observed execution behavior, reducing exploitation-time hallucinations and improving the success rate of vulnerability confirmation.

Design. Built upon these techniques, *iFinder* adopts a multi-agent design to perform four key tasks: preprocessing, discovery, vetting, and exploitation. Every agent in *iFinder* is an LLM session (realized with the Claude Agent SDK) driven by a task prompt and tool calls (`Grep`, `Glob`, `Read`, and build/execute), through which all reasoning is carried out by the LLM. The preprocessing stage supports the other tasks through two one-time operations (Section 4.2). First, it extracts protocol procedures and message schemas from 3GPP specifications [20]. Second, it summarizes detection patterns from cluster-level root causes; these patterns are designed to be both general, applicable across protocols and programming languages, and accurate. Guided by these patterns, a Discovery Agent detects iTrue candidates via LLM-based backward analysis. However, this approach can yield a high false-positive rate when the required checks are implemented earlier in the same procedure, as discussed above. To address this challenge, *iFinder* invokes a Vetting Agent to perform code-specification cross-checking (Section 4.3). The resulting candidates are further validated by an Exploitation Agent, which automatically generates a PoC from the inferred attack vector and compliant prerequisite messages, executes it against a testbed, and iteratively refines it using real-time runtime logs (Section 4.5).

These agents communicate via explicit intermediate artifacts, including detection patterns, iTrue candidate reports, feasible-candidate reports, and prerequisite messages, which ensures reliable information flow across agents and supports end-to-end iTrue discovery and confirmation. Figure 3 illustrates the system workflow. In what follows, we present the designs of these agents and the preprocessing component.

4.2 Preprocessing

As mentioned earlier, the preprocessing stage aims at (1) extracting procedures and message schemas from 3GPP documents, and (2) summarizing detection patterns distilled from cluster-level root causes. To ensure the reliability of the outcomes, particularly the quality of the patterns, a domain expert reviewed the generated patterns and, when necessary, re-invoked the generation process with additional feedback to improve them.

Summarizing detection patterns. CN implementations are written in heterogeneous languages (e.g., C and Go) and span multiple protocols (e.g., PFCP and GTP-C); thus, a pattern that is both precise and general is necessary to enable more effective and reliable exploration. To achieve generality, we carefully design prompts so that the generated patterns are expressed in a protocol-agnostic and language-agnostic manner. Meanwhile, to ensure precision and reduce ambiguity for iTrue discovery, we formulate each pattern as a triple: <element, dangerous operation, missing validation>, where element denotes the involved information element (IE), dangerous operation denotes the security-critical operation performed on the IE, and missing validation denotes the necessary check that is absent before the operation is performed. Figure 4 presents an example that can be instantiated as <mandatory IE, direct dereference, missing presence>. Finally, we summarize the six iTrue patterns.

Extracting procedures and message schemas from specifications. iFinder utilizes an instruction prompt to extract

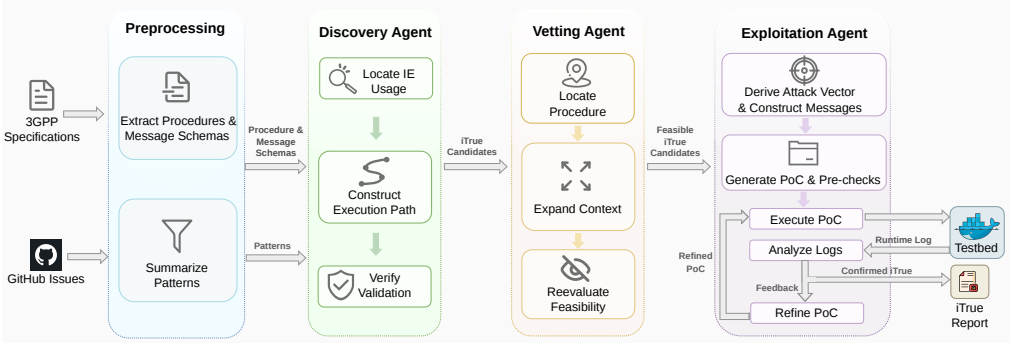

Figure 3: Overview of the iFinder framework. iFinder consists of three agents: a Discovery Agent that identifies candidate implicit trust violations via LLM-based backward analysis, a Vetting Agent that cross-checks code and protocol semantics to filter false positives, and an Exploitation Agent that validates security impact in a controlled testbed.

Absent Field Pattern

The code directly dereferences the mandatory information element without checking that the mandatory element is actually present. These unchecked accesses can lead to null pointer dereferences, leading to DoS.

Figure 4: Example of an iTrue detection pattern (*Absent Field*): a mandatory IE is dereferenced without a presence check, enabling a DoS.

procedures and message schemas from 3GPP specifications. The prompt includes an illustrative example (e.g., the Modify Bearer procedure) to explain what constitutes a procedure and reduce ambiguity. Such extracted knowledge helps downstream agents better understand protocol behavior, enabling them to reason about protocol execution when validating flagged iTrues and generating PoCs.

4.3 Discovery Agent

With the inputs from the preprocessing component, particularly the summarized patterns, the Discovery Agent (DA) searches for iTrue instances across various CN implementations. As mentioned earlier, directly prompting an LLM to scan the entire codebase and flag iTrue candidates matching a pattern is impractical: the codebase spans over one million lines of code, far exceeding typical context-window limits and making long-context reasoning both inefficient and prone to hallucinations. To address this challenge, the DA performs *LLM-based backward analysis* for iTrue discovery. The DA is an LLM session driven by a task prompt and carries out this analysis entirely through iterative `Grep`/`Glob`/`Read` tool calls. Specifically, the procedure has three steps: it first locates a dangerous operation that matches the pattern, then walks backward along caller chains to construct the execution path, and finally checks whether the validations required by the pattern are missing.

Locating risky IE usages. The DA first identifies code locations where protocol-controlled Information Elements (IEs) are used in a manner that matches a given pattern. As described earlier, each detection pattern is specified to be language-agnostic (e.g., "memory operations") to support cross-language applicability. The DA begins by reading the pattern description and grounding these abstract operations into concrete, language-specific constructions before detecting them in the target CN implementation. For each detected usage site, the DA then determines which IE field is used as an argument to the dangerous operation. For example, given a "Malformed Field" pattern, the DA first locates `memcpy`-like memory operations; then it finds that `flow_description_len` from the `SDFFilter` IE is passed as the size argument to a memory operation at `types.c:340`.

Constructing execution path. After locating a pattern-matched IE usage, the DA constructs the execution path by iteratively searching for and inspecting the callers along the backward chain, so that it can later check whether the required validations are present. Specifically, the DA iteratively expands the call chain discovered from a CN implementation by using `Grep` to identify candidate callers, `Glob` to locate relevant source files, and `Read` to inspect each caller's code, repeating these operations until it reaches the protocol message handler[1]. The resulting call chain spans from the protocol message handler, through intermediate parsing functions, to the operations matching the pattern.

Checking the existence of validation. With the execution path constructed, and guided by the pattern specification, the DA checks whether the necessary validations for the target IE appear before the risky use. Depending on the pattern, the DA may perform three kinds of validation: (1) protocol-syntax validation (e.g., the presence of mandatory

[1]We found that the agent's standard approach of constructing call graphs via Abstract Syntax Trees (ASTs) is less effective than a simple `Grep`-based method, possibly because the LLM is better optimized for such document-search commands.

field and basic format/length constraints before accessing IE fields), (2) protocol-semantics validation (e.g., range checks or state-dependent invariants on IE values), and (3) resource-availability validation (e.g., capacity checks or access control before allocating requested resources). The DA reads the functions in the execution path to look for these checks; if it does not find the required validation before the dangerous operation, it flags the site as an iTrue candidate.

4.4 Vetting Agent

As mentioned earlier, the vulnerability candidates produced by the DA include many false positives, because security checks may occur in earlier states rather than in the flagged snippet. Addressing this challenge is the Vetting Agent (VA), which filters candidates via code-specification cross-checking: a three-step approach that leverages 3GPP specifications to expand the analysis context and reason about each candidate's feasibility. The approach includes locating a iTrue candidate's message in the procedure of 3GPP specifications, then mapping the prerequisite messages to code by searching, and reevaluating iTrue candidates under the expanded code context.

Locating an iTrue candidate in specifications. The first step is to locate the procedure that carries an iTrue candidate in the specifications. For this purpose, the VA first identifies, among the procedures extracted during the preprocessing stage, the one that contains the message involved in the candidate (as identified by the DA). It then enumerates all other messages in the same procedure that are exchanged prior to the triggering message; these messages serve as "anchors" to help determine, later, the code segment that implements the procedure. For example, consider an iTrue candidate: the assertion `ogs_assert_if_reached()` at `s11-handler.c:608`, which is triggered along the `ModifyBearerRequest` handling path when `session_type` takes an unexpected value. Using the triggering message `ModifyBearerRequest`, the VA maps the candidate to the Modify Bearer procedure and then retrieves the prior messages in the procedure, including `CreateSessionRequest` and `CreateSessionResponse`.

Mapping specification messages to code. After identifying the messages for a candidate, the VA maps them to their code-level handlers by searching the codebase for the message names. For example, given a candidate in the `ModifyBearerRequest` handler, the VA identifies two prior messages in the same procedure, `CreateSessionRequest` and `CreateSessionResponse`, and then locates their implementations by searching for these names in the codebase; as a result, the S11 `CreateSessionRequest` handling path and the S5c `CreateSessionResponse` handling path are discovered, expanding the analysis context from the flagged handler to the full procedure.

Reevaluating iTrue candidates in expanded context. With the context expanded, the VA reevaluates whether the candidate vulnerability is still exploitable by examining the located handler code for these messages, looking for the relevant validation operations. For example, consider a candidate that accesses the type field in the PAA IE, where the semantic check on this IE is absent in the handler of `ModifyBearerRequest` that uses the IE; after expanding the context, however, the VA discovers that this IE is actually a reference to the IE parsed in a prior message handled by `sgwc_s5c_handle_create_session_response()`, which performs both syntactic and semantic checks on the message. This example is illustrated in Figure 5.

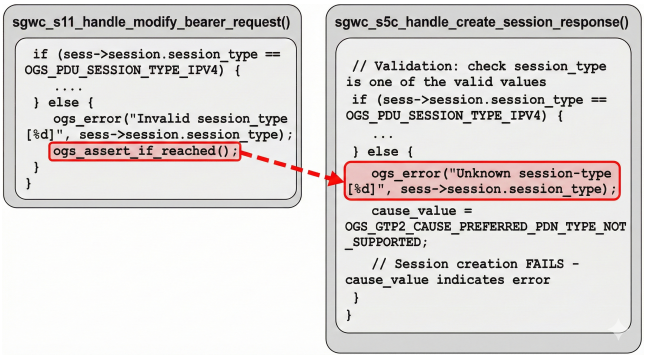

Figure 5: Example of locating an implicit trust violation from a real-world GitHub security issue. The issue description and discussion are used to extract missing-validation assumptions, which are then mapped to the prerequisite messages in the corresponding 3GPP procedure and to the code-level handlers that the VA then reevaluates.

4.5 Exploitation Agent

Even after reevaluation, some false positives may still remain, because static reasoning alone cannot guarantee that a candidate is (i) triggerable under a realistic protocol state, (ii) reproducible in an actual deployment, or (iii) genuinely security-impacting. Therefore, *iFinder* invokes the Exploitation Agent (EA) to automatically generate a PoC for each iTrue candidate. More specifically, the EA follows a three-step workflow. First, it derives an explicit attack vector and instantiates protocol-compliant messages using the message schemas extracted during preprocessing, and then generates the PoC strictly from these intermediate artifacts. Second, it performs pre-execution checks (e.g., compilability and consistency with the attack vector) to catch malformed or drifting PoCs. Finally, when a well-formed PoC still fails due to implicit runtime constraints, the EA invokes the *feedback-aware refinement* loop: it analyzes runtime logs, updates the PoC to satisfy missing constraints, and re-executes it until the flaw is triggered or the LLM testing budget is exhausted.

Deriving attack vector and constructing protocol mes-

sages. Given an iTrue candidate from the VA (which specifies the vulnerable IE usage site and triggering message type), the EA first derives a message-level attack vector by determining the specific IE field(s) to manipulate and the required content to trigger the target vulnerability. It then constructs the corresponding protocol message(s) according to the schemas, producing well-formed message structures with the attack payloads for PoC generation. For example, the VA identifies a heap buffer overflow risk introduced by the vulnerable `memcpy` at `s5c-handler.c:150`, which copies the `PAA` IE payload into the buffer `sess->paa` using the attacker-controlled length field `paa->len` without bounds checking; from these artifacts, the EA derives the attack vector: sending a `CreateSessionResponse` containing a malformed `PAA` IE, where the length field claims 200 bytes while the buffer `sess->paa` can only hold ∼21 bytes, causing an out-of-bound write; based on the message schema for `CreateSessionResponse`, the EA populates the required mandatory header fields (e.g., the TEID and sequence number) and embeds the malformed `PAA` IE at the correct location, constructing a message instance to drive PoC generation.

Generating PoC and performing pre-execution checks. After deriving the attack vector and protocol-compliant messages, the EA generates an initial PoC that instantiates these messages accordingly. Before execution, the EA performs lightweight, tool-assisted pre-execution checks, including (i) syntax checks by compiling the PoC and (ii) consistency checks to ensure the PoC's message types, field values, and message sequence align with the attack vector.

Executing in a testbed and refining with feedback. The EA then executes the generated PoC by automatically invoking a containerized testbed and analyzing runtime logs to determine whether the vulnerability is triggered. If the attack succeeds (e.g., a crash or an observable state change occurs), the EA confirms the presence of the iTrue vulnerability; otherwise, it diagnoses the failure based on the logs, refines the PoC accordingly, and re-executes it. This execute–analyze–refine loop continues until the vulnerability is triggered or the testing budget is exhausted. For example, an initial PoC may send a malformed `CreateSessionResponse` with a hardcoded TEID and sequence number. The first execution does not trigger a crash because SGW-C rejects the response: the message cannot be linked to any pending transaction ("unknown TEID"). Based on this feedback, the EA updates the PoC to extract the TEID from `CreateSessionRequest` and reuse the corresponding sequence number when crafting the response. With this refinement, SGW-C links the response with its pending transaction and processes the `CreateSessionResponse`, which leads to a crash, confirming the existence of the iTrue.

5 Evaluation

We evaluated iFinder on the ground-truth dataset with the known CN iTrues collected from GitHub security issues, and focused on the following research questions:

RQ1 *How effective is iFinder at discovering (known) iTrue vulnerabilities?*

RQ2 *How much does the code-specification cross-checking reduce false positives?*

RQ3 *To what extent does feedback-aware refinement improve PoC generation success?*

RQ4 *What are the time and token costs of running iFinder?*

RQ1 provides an overview of iFinder's effectiveness in detecting iTrue vulnerabilities. RQ2 and RQ3 evaluate the effectiveness of our two main proposed techniques, code-specification cross-checking and feedback-aware refinement, in mitigating LLM hallucinations. RQ4 further examines the efficiency of iFinder.

5.1 Experimental Setup

Ground truth dataset. As discussed in Section 4.2, these known iTrue vulnerabilities are classified into six patterns with the following distribution: *Malformed Field* (11), *Absent Field* (1), *Invalid Value* (6), *Invalid State* (1), *Invalid Reference* (1), and *Resource Exhaustion* (2). The distribution is heavy-tailed, with a small number of patterns accounting for most instances.

Tested CN implementations. We evaluated iFinder on two open-source CN implementations, Open5GS and free5GC. Since each ground-truth vulnerability exists only in a specific range of versions, we selected multiple version snapshots to cover the affected releases referenced by the corresponding issues. Specifically, we used Open5GS v2.7.5, v2.7.2, and v2.4.14, as well as free5GC v3.3.0 and v2.0.2. This setup ensures that each vulnerability is evaluated on a version in which it is known to exist.

Evaluation metrics. For vulnerability detection, we report true positives (TP), false positives (FP), and false negatives (FN), along with precision, recall, and F1. A report is counted as a true positive (TP) if it correctly identifies a ground-truth vulnerability; similarly, we define false positives (FPs) and false negatives (FNs). For PoC generation, we consider a PoC successful if it triggers the expected behavior.

System Implementation. We implement iFinder's multi-agent pipeline using the Claude Agent SDK. All methods, including the prompt-only baseline, use Claude Opus 4.5 as the underlying LLM. We ran all methods with the default configurations (e.g., decoding parameters and context limits) and capped the number of refinement iterations per vulnerability to five. We measured runtime as wall-clock time and reported token usage as the total input and output tokens consumed by all agents (including refinement), which we further converted

Table 1: Detection performance on the 22 ground-truth vulnerabilities.

Configuration	TP	FP	FN	New	Precision	Recall	F1
Prompt-only	8	56	14	14	28.205%	36.364%	31.769%
iFinder	15	12	7	21	75.000%	68.182%	71.429%

Table 2: Ablation study.

Configuration	TP	FP	FN	New	Precision	Recall	F1
DA	15	62	7	21	36.735%	68.182%	47.745%
DA+VA	15	19	7	21	65.455%	68.182%	66.790%
DA+VA+EA	15	12	7	21	75.000%	68.182%	71.429%

to cost using the pricing of the provider. All experiments are repeated five times, and we were reported the average results.

5.2 RQ1: Detection Effectiveness

To answer RQ1, we evaluated iFinder's ability to identify the 22 known vulnerabilities and compared its performance against a prompt-only baseline. Notably, this baseline is not a pure LLM setup; it is fed with the patterns we summarized from GitHub issues. We adopted this enriched baseline because directly asking the LLM to discover vulnerabilities yielded severe hallucinations and no usable results. We report TP and FN to compute recall, which measures how well the approach identifies the known vulnerabilities. In addition, we report the number of newly discovered vulnerabilities and FP among all findings, from which we compute precision. The results are summarized in Table 1. Overall, iFinder achieved an F1 score of 71.429%, which is more than twice that of the baseline. More specifically, iFinder identified 68.182% of the ground-truth vulnerabilities, which is roughly twice as high as the baseline. To understand the remaining gap, we manually reviewed all seven missed cases and attributed them to two root causes. First, *pattern coverage gaps*: four cases involve corner-case validation logic not captured by the generalized detection patterns distilled from cluster-level root causes. Second, *incomplete context construction*: three cases lie on long cross-module execution paths whose caller chains the DA cannot reliably reconstruct, so the missing check cannot be confirmed and therefore the candidate was not flagged. Notably, while the baseline performed substantially worse, it did not completely lose utility, achieving 28.205% precision. This is because the baseline incorporates the patterns we summarized, which also highlights the utility of these patterns in guiding LLM-based vulnerability discovery. In contrast, more than $2/3$ of the reported vulnerabilities were confirmed to be real. In a later section, we further discuss how our proposed code-specification cross-checking improves precision in detail.

5.3 RQ2 + RQ3: Ablation Study

To answer RQ2 and RQ3, we conducted an ablation study to evaluate the effectiveness of our two proposed primitives: code-specification cross-checking, performed by the VA, and feedback-aware refinement, performed by the EA. Comparing DA, DA+VA, and DA+VA+EA (Table 2), we found that code-specification cross-checking reduced the FP from 62 to 19, improving the precision from 36.735% to 65.455%. To assess feedback-aware refinement, we provided the 22 ground-truth vulnerabilities to the EA and to the prompt-only baseline (i.e., directly prompting the LLM) and measured how well each generated valid PoCs. The EA produced valid PoCs for 19 of the 22 ground-truth vulnerabilities, whereas the prompt-only baseline succeeded on only 8, demonstrating the effectiveness of feedback-aware refinement in enabling reliable PoC generation. The EA further reduced FP from 19 to 12 by discarding candidates whose PoCs fail to trigger the intended flaw, yielding the full-pipeline precision of 75%. Still 12 FPs remained because a generated PoC can trigger a real yet unintended vulnerability located prior to the target flaw on the execution path. The EA only checks whether a PoC produces the expected runtime effect, which in the case of an FP, is always a crash. It does not verify whether the crash site corresponds to the intended vulnerability candidate. We identify these cases by manual inspection: for each PoC the EA reports as successful, we compare the *observed crash site* from the runtime logs against the *expected crash site*, i.e., the dangerous operation and its handler path localized by the DA and the VA. If the observed crash site does not match the expected crash site, the PoC is believed to have triggered an unintended vulnerability, so we count the report as an FP.

5.4 RQ4: Overhead Analysis

To answer RQ4, we measured the runtime and token usage of each pipeline stage; the results are summarized in Table 3. The Discovery agent accounted for the majority of the runtime (39.436%), as it scans the codebase to locate message handlers and performs LLM-based backward analysis before producing vulnerability candidates. Vetting was comparatively lightweight (32.394%) because it operates only on DA vulnerability candidates and retrieves minimal protocol-aligned context. Exploitation time varies with the number of refinement iterations (up to 5).

Table 3: Time and token breakdown per target implementation.

Stage	**Avg. Time (s)** ↓	**Tokens** ↓
Discovery	280	2.1M
Vetting	230	3.3M
Exploitation	200	4.4M
Total	**710**	**9.8M**

Table 4: Vulnerability discovery results by implementation.

Target	Language	#Discovered	#Confirmed	#CVE
Open5GS 5G	C	10	9	9
free5GC	Go	14	14	12
OAI 5G	C++	11	11	11
SD-Core	Go	7	7	7
eUPF	Go	5	5	5
Open5GS LTE	C	30	30	30
OAI LTE	C++	7	7	7
Total		**84**	**83**	**81**

Table 5: Cross-protocol distribution of vulnerability patterns.

Pattern	GTP-C	PFCP	Total
Malformed Field	11	3	14
Absent Field	6	25	31
Invalid Value	4	8	12
Invalid State	7	1	8
Invalid Reference	2	3	5
Resource Exhaustion	7	7	14
Total	**37**	**47**	**84**

6 Measurement on New CN iTrues

We further ran *iFinder* on seven widely used open-source CN implementations, including Open5GS 5G, free5GC, OAI 5G, SD-Core, eUPF, Open5GS LTE, and OAI LTE, covering both 4G and 5G systems and in three different languages (C, C++, and Go). Our goal is to assess whether the discovered vulnerabilities are isolated implementation bugs or instead indicate systemic weaknesses in modern core networks.

6.1 Discoveries

Landscape. Across the seven evaluated implementations, *iFinder* reported 84 previously unknown vulnerabilities spanning two protocols, GTP-C and PFCP (Table 4). *At the time of writing, 83 findings have been confirmed by developers, and 81 have been assigned CVE identifiers*, spanning all seven implementations. While we do not claim to have exhaustively captured all iTrue vulnerabilities in these codebases, the pervasiveness of the flaws, and their recurrence across multiple systems (implemented in different programming languages) and across different protocols, strongly suggests that iTrues are not isolated implementation bugs but rather systemic weaknesses in carrier core networks, rooted in implicit trust in internal peers. As discussed earlier, this assumption is increasingly invalidated as CNs migrate to less protected cloud environments.

Of particular interest, we observe that certain iTrue flaws in 5G systems appear to be inherited from their 4G counterparts. This suggests that the new risk can arise from a failure to adapt legacy designs and assumptions to a deployment environment that is no longer equally trusted. Note that many production CNs today support both 4G and 5G simultaneously, so risks originating in 4G components remain a serious security concern.

What is most surprising is the diversity of vulnerabilities that nonetheless fall under the same high-level category. For example, in free5GC, repeatedly sending PFCP `Session Establishment Request` messages can drive the UPF into a crash by exhausting internal resources. In Open5GS, repeatedly sending GTPv2-C `Create Session Request` messages can similarly cause the SGW-C to terminate by depleting multiple internal resource pools, including UE contexts, transaction timers, and bearer-related structures. Despite differences in the targeted resources, protocols, and implementation languages, both cases can be attributed to the same root cause: handlers implicitly trust that internal requestors will behave prudently and will not intentionally deplete finite resources. This optimism often leads to fragile exception handling that is incomplete or inconsistently applied. As a result, resource exhaustion is more likely to surface as a process crash than as controlled rejection or graceful degradation. Notably, despite their diversity, both flaws are captured by the same detection pattern.

Cross-protocol pattern distribution. To characterize the discovered vulnerabilities, we analyzed their distribution across pattern families, as summarized in Table 5. Notably, every pattern family recurs in both GTP-C and PFCP, suggesting that many vulnerabilities arise from shared implementation assumptions rather than protocol-specific semantics. Among all patterns, *Absent Field* is the most common, accounting for 31 of the 84 vulnerabilities, with a substantial fraction occurring in PFCP implementations (25/31). This result is consistent with PFCP handlers implicitly assuming the presence of mandatory information elements and proceeding along nominal control procedures without enforcing completeness checks. *Malformed Field* is another prevalent pattern, reflecting insufficient validation of length fields and structural counts in complex message formats. Compared with PFCP, GTP-C exhibits more *Malformed Field* vulnerabilities, likely because GTP-C messages rely on a richer and more diverse IE ecosystem, with extensive use of grouped IEs and nested length fields. This complexity makes GTP-C parsers more susceptible to desynchronization, which can in turn trigger memory-safety issues or logic errors. In addition, *Invalid Value* and *Invalid State* show a skewed distribution: *Invalid Value* is more prevalent in PFCP, whereas *Invalid State* occurs more often in GTP-C. This asymmetry may reflect the more complex state machines managed by GTP-C implementations, which are more susceptible to inconsistencies under unexpected message sequences. By comparison, *Invalid Reference* and *Resource Exhaustion* are more evenly distributed across the two protocols, likely because both protocols man-

Table 6: Vulnerability breakdown by CN components.

Component	Protocol		Total
	GTP-C	PFCP	
4G Core			
MME	2	N/A	2
SGW	32	1	33
PGW	3	0	3
5G Core			
SMF	N/A	9	9
UPF	N/A	37	37
Total	**37**	**47**	**84**

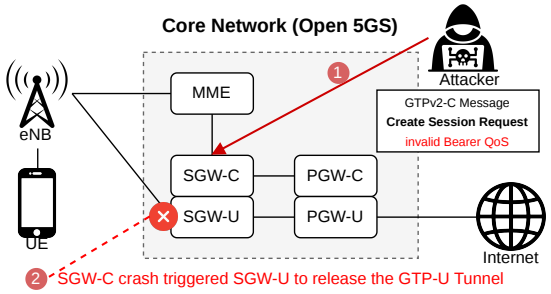

Figure 6: Example attack exploiting malformed GTPv2-C messages to trigger denial-of-service in a core network component by abusing missing validation checks.

age broadly comparable categories of resources and therefore exhibit similar exposure to *Invalid Reference* and *Resource Exhaustion* vulnerabilities.

Interestingly, although PFCP is a newer protocol originally introduced for CUPS and later widely adopted in 5G architectures, it exhibits more vulnerabilities than GTP-C in our study (47 versus 37). One plausible explanation is that PFCP handlers are often implemented as internal resource-management interfaces, where messages are more likely to be assumed well-formed and thus receive less rigorous validation. Overall, the cross-protocol recurrence of syntax-, semantics-, and resource-related pattern families suggests that implicit-trust assumptions can persist at the implementation level across protocol generations, leading to accumulated gaps in validation logic rather than isolated programming errors.

Component-level breakdowns. To understand where these implicit-trust assumptions most frequently surface in practice, we further examined how vulnerabilities are distributed across core network components, with results summarized in Table 6. The results also revealed a skewed distribution. In 4G systems, the Serving Gateway (SGW) accounted for the majority of GTP-C vulnerabilities (32/37). In 5G systems, the User Plane Function (UPF) accounted for the majority of PFCP vulnerabilities (37/47). Such observed concentration is because SGW/UPF components maintain long-lived protocol state across multi-step procedures, manage scarce and performance-critical resources, and mediate interactions among multiple peers while reconciling control-plane decisions with user-plane resource allocation. Such cross-plane coupling increases the complexity of state transitions and error handling, which can make missing validation and incomplete recovery paths more likely to surface. Moreover, the discovered flaws span both 4G and 5G, and this cross-generation coverage yet again supports the view that iTrue reflects a systemic weakness rather than isolated bugs in individual codebases: these components sit at the intersection of stateful control logic and resource management.

6.2 Case Studies

We present two case studies to demonstrate the security impact of the newly discovered iTrue vulnerabilities.

Denial of service. *iFinder* reported an iTrue vulnerability in SGW-C caused by missing syntactic validation of the Bearer Quality of Service (QoS) Information Element (IE) when parsing GTPv2-C *Create Session Request* messages. Specifically, a *Create Session Request* that carries a Bearer QoS IE with an invalid fixed-length encoding can trigger an assertion, causing SGW-C to crash and resulting in a DoS. As stated in our threat model, we consider a remote adversary with no direct control over cellular CN components, but who can send arbitrary GTPv2-C messages to trigger the vulnerability. In our study, we successfully executed an end-to-end DoS attack against Open5GS LTE in our testbed. As shown in Figure 6, the attack proceeds as follows: (1) the attacker sends a GTPv2-C *Create Session Request* containing a Bearer QoS IE with an invalid fixed-length encoding; (2) when SGW-C parses the malformed IE, it crashes immediately, triggering SGW-U to release the SGW-U-eNB GTP-U tunnels.

Session hijacking. *iFinder* detected an iTrue vulnerability in the UPF caused by missing uniqueness validation of Packet Detection Rule (PDR) IDs when processing PFCP *Session Modification Request* messages. Specifically, by injecting a duplicate PDR ID with a lower `Precedence` value (i.e., higher priority), an attacker can overrule the legitimate rule. This logic flaw causes the UPF to match the malicious PDR first during packet processing, resulting in session hijacking.

In line with the threat model, we assume an Internet attacker with no direct control over core-network components who, through a cloud-native misconfiguration that exposes the N4 interface, can send arbitrary PFCP messages to the UPF to trigger a vulnerability. We demonstrated an end-to-end session-hijacking attack on OAI 5G in our testbed. As shown in Figure 7, the attack proceeds as follows: (1) the attacker sends a PFCP *Association Setup Request* to the UPF; (2) the victim UE initiates an attach, triggering the SMF to send a PFCP *Session Establishment Request* to the UPF; (3)

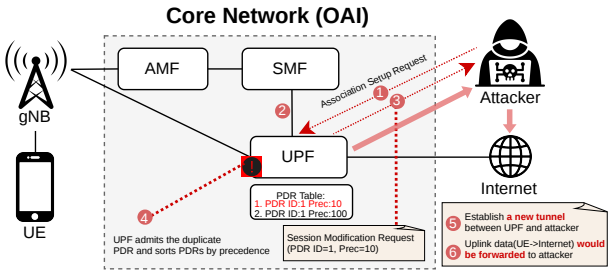

Figure 7: Example attack exploiting duplicate PDR IDs in PFCP *Session Modification Request* messages to trigger session hijacking in UPF by abusing missing uniqueness validation.

the attacker then issues a PFCP *Session Modification Request* that reuses the victim's PDR ID with a lower `Precedence` value (higher priority) and binds it to a malicious Forwarding Action Rule (FAR); (4) the UPF admits the duplicate PDR and sorts PDRs by precedence, placing the malicious rule ahead of the legitimate one; and (5) during packet processing, the UPF matches the malicious PDR first, and establishes a new forwarding tunnel between UPF and attacker; and (6) as a result, the victim's uplink traffic is forwarded to the attacker rather than to the Internet. Beyond the OAI testbed, we further validated this flaw on two commercial 5G core networks. In collaboration with a partner vendor, we reproduced the attack in the vendor's lab environment under default configurations. In both cases, the same root cause (missing uniqueness validation of PDR IDs) was present, and the attack exhibited the same session-hijacking behavior observed on OAI. Specifically, the duplicate PDR with a lower `Precedence` value (higher priority) overruled the legitimate rule for forwarding the victim's uplink traffic to the Internet, redirecting it to the attacker instead. One vendor has fixed the issue, and a CVE has been assigned (CVE-2026-8233); the other, a major 5G carrier, is still in the remediation process.

7 Discussion

Vulnerabilities in CNs. Our research brings to light the pervasiveness and severity of iTrue vulnerabilities, which we attribute to a systemic specification-to-implementation gap. Requirements for protocol-structure validation, protocol-semantics validation, and resource-availability checks are dispersed across 3GPP specifications, increasing the likelihood of incomplete enforcement in implementations and leading to recurring weaknesses such as iTrue. Crucially, these gaps threaten not only availability (e.g., denial of service) but also integrity (e.g., session hijacking), as demonstrated in our research, indicating that iTrue flaws are not isolated implementation bugs but systematic concerns.

Mitigating iTrue risks requires coordinated action across specifications, implementation, and deployment. At the specification level, 3GPP should update its threat model to reflect cloud-native deployments in which internal interfaces may be exposed, and should mandate mutual authentication on internal interfaces. At the implementation level, network-function developers should enforce strict syntactic and semantic validation at protocol boundaries, maintain uniqueness invariants for session and rule identifiers, and adopt bounded resource allocation with graceful rejection rather than process-ending assertions. At the deployment level, operators should adopt a zero-trust posture by removing unintended reachability, enforcing net-segmentation and least-privilege access between components, and continuously auditing cloud configurations to detect misconfigurations before they become exploitable.

We hope that the findings of this study will help catalyze such efforts, contributing to defense-in-depth protections for future CNs in particular, and carrier networks more broadly.

Limitations of iFinder. *iFinder* does not guarantee complete coverage of all iTrue vulnerabilities. The Discovery Agent is bounded by our seed pattern set, and vulnerabilities that do not match these patterns may be missed. Moreover, although our pipeline leverages procedure context and message schemas to reduce false positives, the LLM may still make incorrect feasibility judgments when exploitability hinges on subtle specification semantics, implementation-specific state transitions, or implicit invariants that are not captured by message structure alone. Future work should explore additional implicit assumptions underlying CNs and incorporate deeper, system-level analysis to further reduce false positives and broaden coverage.

Agent-based security analysis on carrier networks. Our results demonstrate that multi-agent systems can automate vulnerability discovery across multiple protocols, achieving a level of scalability and accuracy that is difficult to attain with manual auditing or conventional, less intelligent tools. Of particular importance is the agentic system's ability to perform multimodal analysis, jointly reasoning over code, specifications, and other artifacts, and to do so smoothly across different programming languages and protocols.

These capabilities are especially valuable for carrier-network security analysis, where the ecosystem is complex and heterogeneous, and where both the codebases and the supporting documentation are massive. Looking ahead, we expect agentic techniques to see broader adoption and tighter integration into pipelines for both specification development and system implementation, ultimately improving the security posture of this critical information infrastructure.

8 Related Work

Cellular Network Security. A large body of research has examined the security of cellular networks across different protocol layers and interfaces. At the UE and RAN side, prior work has investigated baseband vulnerabilities [35–37],

lower-layer attacks [38, 39], eavesdropping [40], privacy violations [41], and protocol compliance testing [6, 28, 42–44]. However, these efforts predominantly target the air interface between UE and RAN or the NAS/RRC signaling between UE and core, leaving the *internal interfaces* of the core network relatively underexplored. Recent work has started to examine core-network security from specific angles: formal verification of access-control mechanisms [18, 45], provenance-based attack detection [46], context-integrity testing of the LTE core [1], and GTP-U tunnel abuse that demonstrates external attackers can reach internal CN interfaces [4, 33]. Motivated by the growing reachability of internal interfaces in cloud-native deployments, our work takes a complementary direction: we systematically study implementation vulnerabilities across PFCP and GTP-C protocols and characterize *iTrue* as a recurring type of security flaw, where core-network components blindly trust internal messages and skip critical validations, leading to severe consequences including DoS and session hijacking.

Automated Vulnerability Discovery in Cellular Networks. Fuzzing has been applied to cellular implementations from UE-RAN side testing [5, 28, 43] to core-network fuzzing [7–9, 47], but these approaches predominantly uncover memory-safety bugs and crash-inducing issues, while many high-impact core-network failures are logic vulnerabilities that do not manifest as crashes. Specification-based analysis mines vulnerabilities from 3GPP documents through NLP [48], change-request analysis [49], automated reasoning [50], undefined-behavior studies [51], and protocol state-machine synthesis [52–54]; however, these methods target design flaws at the specification level and cannot automatically validate whether identified issues are exploitable in real implementations. In contrast, iFinder distills recurring vulnerability patterns from previously discovered core network flaws and uses them to guide an LLM-driven multi-agent system, enabling efficient discovery of both memory-safety and deep logic vulnerabilities while closing the loop with code-specification cross-checking and runtime-verified PoC generation.

Multi-Agent for Vulnerability Discovery. LLMs have recently been applied to automate vulnerability discovery through agentic and multi-agent pipelines. Previous works have used LLMs to enhance existing static-analysis tools, e.g., improving the efficiency of CodeQL [55] or customizing CodeQL for JavaScript vulnerability detection [56]. Others build general-purpose multi-agent pipelines to discover bugs in general programs [57, 58]. However, these systems target general-purpose code and are not tailored to stateful protocols, so they are ineffective for 5G core networks and only report vulnerability candidates without confirming exploitability. In contrast, *iFinder* employs a pattern-guided multi-agent system that performs code-specification cross-checking to recover procedure context and iteratively refines PoCs with feedback, enabling it to detect iTrues in 5G core network implementations and confirm their exploitability.

Input Validation and Specification-Driven Testing. Some iTrues are related to input-validation vulnerabilities, a broad class that includes XSS [59], SQL injection [60], and, more recently, prompt injection [61], each with distinct root causes and mitigation strategies. However, many other iTrues do not fall under input validation: they are triggered by well-formed inputs that arrive in incorrect system states. Because PFCP and GTP-C are long-lived, stateful protocols, inconsistencies between internal states and otherwise valid inputs can introduce security risks. For example, a duplicate PDR ID may be syntactically valid yet still cause session hijacking. This distinction differentiates many iTrues from input-validation vulnerabilities. Moreover, although iTrues are not fundamentally new, their detection cannot be effectively addressed by existing techniques. Specification-driven approaches, including model-based testing [62], primarily target design flaws, while stateful protocol testing [63] and stateful network fuzzing [26] often rely on crash-based signals and therefore struggle to capture logic flaws that do not manifest as crashes.

9 Conclusion

As cloud-native development erodes the physical isolation traditionally afforded to cellular CNs, We use *implicit trust errors* (iTrue) to capture a recurring type of security flaw. We designed *iFinder*, an LLM-driven multi-agent system, and applied it to discover new iTrues in CN implementations. Running iFinder on seven prominent open-source CN implementations, we discovered 84 previously unknown vulnerabilities. Among them, 81 have been confirmed with 81 CVE assignments already awarded. Importantly, a session-hijacking flaw has already been confirmed on real-world commercial 5G core networks. Our findings highlight the pervasiveness of iTrues across the cellular CNs and the urgent need for elevated protection within the original trust domains. We plan to release iFinder as a public service to help enhance the security of cellular core networks.

Acknowledgments

We thank the shepherd, all anonymous reviewers for their valuable comments. This research is supported by the National Research Foundation, Singapore, and the Cyber Security Agency of Singapore under the National Cybersecurity R&D Programme and the CyberSG R&D Programme Office (Award CRPO-GC3-NTU-001 and Award CRPO-GC6-NTU-001), and NTU startup grant (025559-00001). Any opinions, findings, conclusions, or recommendations expressed in these materials are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, the Cyber Security Agency of Singapore, or the CyberSG R&D Programme Office.

Open Science

We make the artifacts required to evaluate our contributions available at this link:https://zenodo.org/records/20534406. These artifacts include the implementation of the iFinder framework, the curated dataset of publicly reported iTrue vulnerabilities used as ground truth, and scripts and configuration files necessary to reproduce the main experimental results in controlled environments. To prevent misuse prior to widespread patch adoption, we do not release exploit-grade proof-of-concept code for newly discovered vulnerabilities. The released artifacts are sufficient to reproduce our evaluation and validate the reported findings.

Ethical Considerations

We structure the ethical considerations discussion by linking our stakeholder analysis to the impacts generated during two distinct phases: the *research process* (vulnerability discovery and validation) and the *publication of results* (disclosure and artifact release). We then detail *mitigations* (specifying which stakeholder group each measure protects) and conclude with the *justification* for conducting this research.

Stakeholder Analysis and Process Impact. Our work involves six primary stakeholder groups. (1) *Open-source CN Maintainers*: the developer teams of the seven evaluated implementations (Open5GS/5G, Open5GS/LTE, free5GC, OAI/5G, OAI/LTE, SD-Core, and eUPF). They rely on the research process to receive clear, reproducible, and responsibly timed vulnerability reports that enable effective patching. (2) *Commercial CN Vendors*: two commercial core-network vendors whose products we additionally tested under vendor-sanctioned conditions. They rely on confidential disclosure channels to protect their customers during coordinated remediation. (3) *Network Operators and Deployers*: telecom operators, private-5G tenants, and research deployments running the affected stacks; their operational security depends on whether discovered flaws are responsibly disclosed and remediated before public release. (4) *End Users*: although no user was involved in our experiments, subscribers are the ultimate stakeholders whose confidentiality, availability, and session integrity are at stake when CN flaws are exploited. (5) *3GPP/GSMA*: their specifications encode the implicit-trust assumptions targeted by iFinder, and they rely on research findings to inform future threat models. (6) *Researchers*, who rely on methodologically sound and ethically scoped procedures. During the research process, all experiments were conducted in isolated, self-hosted testbeds using open-source implementations or vendor-authorized evaluation units. No production operator network, live subscriber, or real user data was accessed, and no stakeholder experienced availability or privacy impact from our experimentation.

Impact of the Research. The publication of this work has both positive and negative impacts, affecting the stakeholders above.

Positive Impacts. (1) Strengthening Deployed Infrastructure (Impact on Maintainers, Vendors, Operators & End Users): iFinder produced 84 previously unknown vulnerability reports, of which 70 have been confirmed and 60 have been assigned CVEs, directly improving the security of widely deployed CN stacks. As a concrete indicator that affected stakeholders view the work as net-beneficial, one of the evaluated projects has proactively requested continued access to iFinder for long-term automated scanning. (2) Informing Standards Bodies (Impact on 3GPP/GSMA): by naming and characterizing iTrue as a recurring specification-to-implementation gap rather than isolated bugs, we equip 3GPP/GSMA, operators, and defenders with a reusable lens to audit their stacks. (3) Reproducibility and Open Science (Impact on Researchers): we release the iFinder framework and the curated ground-truth dataset to allow independent replication, extension, and scrutiny.

Negative Impacts. (1) Misuse of Automated Discovery (Impact on Operators & End Users): the iFinder pipeline lowers the cost of finding similar flaws in other stateful protocol stacks and could be repurposed by malicious actors if released without safeguards. (2) Facilitating Analogous Attacks (Impact on Operators & End Users): public characterization of the vulnerabilities may help adversaries search for analogous bugs, especially in deployments that have not yet applied upstream fixes.

Mitigation. To address the risks outlined above, we have implemented the following measures, each tied to the stakeholder it protects. (1) *Coordinated Disclosure for Maintainers & Operators*: every confirmed vulnerability was disclosed to the respective upstream project or vendor prior to submission. As of the response deadline, 60 CVEs have been assigned, and 58 of 70 confirmed vulnerabilities in open-source implementations have been patched; the per-implementation status is summarized in Table 7. Vulnerabilities on the two commercial CNs were disclosed directly to the vendors.

One affected commercial 5GC vendor has fixed the issues, and CVE identifiers have been assigned, including session hijacking (CVE-2026-8233) and DoS (CVE-2026-8232). The second commercial 5GC vendor, which is a major 5G carrier, is still in the remediation process. (2) *Reducing Misuse Risk for Operators*: although iFinder includes an Exploitation Agent for validating candidate findings in controlled testbeds, the public artifact will not include a collection of PoCs, and we will not provide PoCs or reproduction details for unpatched vulnerabilities. (3) *Protecting Commercial Vendors*: for any vendor still under coordinated remediation, we anonymize vendor identity. (4) *Controlled Experimental Scope for End Users*: all experiments were executed in isolated testbeds with no interaction with production networks, commercial operator infrastructure, or real subscribers.

Justification for Research. This work addresses a critical and systemic security challenge in carrier core networks: as

Table 7: Coordinated-disclosure status per open-source implementation.

Implementation	#Discovered	#Confirmed	#CVE	#Fixed
Open5GS 5G	10	9	9	9
free5GC	14	14	12	12
OAI 5G	11	11	11	0
SD-Core	7	7	7	7
eUPF	5	5	5	0
Open5GS LTE	30	30	30	30
OAI LTE	7	7	7	0
Total	**84**	**83**	**81**	**58**

CNs migrate to cloud-native deployments, the long-standing implicit-trust assumption between internal network functions no longer holds, exposing end users and operators to risks that have received far less scrutiny than RAN or UE components. Conducting this research is essential because: (1) it leads to the discovery of unknown vulnerabilities in open-source implementations and facilitates the enhancement of their security; (2) characterizing iTrue as a recurring type of security flaw, rather than reporting isolated bugs, enables standards bodies, vendors, and defenders to generalize mitigations beyond the specific cases we found; (3) releasing the iFinder framework fosters transparency and community-driven improvements, reducing the risk of opaque or unilateral vulnerability discovery. Under these constraints, we believe the benefits of systematically identifying and mitigating recurring implicit-trust failures in critical network infrastructure outweigh the residual risks.

References

[1] Mincheol Son, Kwangmin Kim, Beomseok Oh, CheolJun Park, and Yongdae Kim. Citesting: Systematic testing of context integrity violations in lte core networks. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 276–290, 2025.

[2] 3GPP. Ip network layer security. Technical Report TS 33.210, 3rd Generation Partnership Project, 08 2020.

[3] Sergio Barrachina-Muñoz, Miquel Payaró, and Josep Mangues-Bafalluy. Cloud-native 5g experimental platform with over-the-air transmissions and end-to-end monitoring. In *2022 13th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pages 692–697. IEEE, 2022.

[4] Yiming Zhang, Tao Wan, Yaru Yang, Haixin Duan, Yichen Wang, Jianjun Chen, Zixiang Wei, and Xiang Li. Invade the walled garden: Evaluating gtp security in cellular networks. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1159–1177. IEEE, 2025.

[5] Matheus E Garbelini, Zewen Shang, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. Towards automated fuzzing of 4g/5g protocol implementations over the air. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 86–92. IEEE, 2022.

[6] Syed Khandker, Michele Guerra, Evangelos Bitsikas, Roger Piqueras Jover, Aanjhan Ranganathan, and Christina Pöpper. Astra-5g: automated over-the-air security testing and research architecture for 5g sa devices. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 89–100, 2024.

[7] Yu Sun, Xinyu Liu, Qian Sun, Jiaming Wang, Lin Tian, and Jianwei Liu. 5gc-fuzz: Finding deep stateful vulnerabilities in 5g core network with black-box fuzzing. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2025.

[8] Yilu Dong, Tianchang Yang, Abdullah Al Ishtiaq, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Md Sultan Mahmud, and Syed Rafiul Hussain. {CoreCrisis}:{Threat-Guided} and {Context-Aware} iterative learning and fuzzing of 5g core networks. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 5287–5306, 2025.

[9] Nathaniel Bennett, Weidong Zhu, Benjamin Simon, Ryon Kennedy, William Enck, Patrick Traynor, and Kevin RB Butler. Ransacked: A domain-informed approach for fuzzing lte and 5g ran-core interfaces. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2027–2041, 2024.

[10] CyRC Global Research Team. Security advisory: DoS via PFCP message handling in Open5GS. https://www.blackduck.com/blog/cyrc-advisory-open5gs.html, 2022. Accessed: 2026-02-01.

[11] Siddharth Sharma. Technical advisory: Open5GS stack buffer overflow during PFCP session establishment on UPF (CVE-2021-41794). https://www.nccgroup.com/research-blog/technical-advisory-open5gs-stack-buffer-overflow-during-pfcp-session-establishment-on-upf-cve-2021-41794/, October 2021. Accessed: 2026-02-01.

[12] Black Duck Software. Defensics fuzz testing. https://www.blackduck.com/fuzz-testing.html, 2026. Accessed: 2026-02-01.

[13] 3rd Generation Partnership Project (3GPP). Network architecture. Technical Specification (TS) 23.002, 3GPP, 5 2022.

[14] 3rd Generation Partnership Project (3GPP). System architecture for the 5g system (5gs). Technical Specification (TS) 23.501, 3GPP, 2023. Release 17.

[15] Ng-ran; ng application protocol (ngap). Technical Report TS 38.413, 3rd Generation Partnership Project (3GPP), December 2024. Release 18.

[16] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. Lteinspector: A systematic approach for adversarial testing of 4g lte. In *Network and Distributed Systems Security (NDSS) Symposium 2018*, 2018.

[17] Evolved general packet radio service (gprs) tunnelling protocol for control plane (gtpv2-c). Technical Report TS 29.274, 3rd Generation Partnership Project (3GPP), April 2021.

[18] Mujtahid Akon, Tianchang Yang, Yilu Dong, and Syed Rafiul Hussain. Formal analysis of access control mechanism of 5g core network. In *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security*, pages 666–680, 2023.

[19] Patrick Donegan. Ipsec deployment strategies for securing lte networks. *White Paper, Heavy Reading, USA*, 2011.

[20] Interface between the control plane and the user plane nodes. Technical Report ETSI TS 129 244, European Telecommunications Standards Institute (ETSI), July 2024. Release 18.

[21] 3GPP. Gprs tunnelling protocol user plane (gtpv1-u). Technical Report TS 29.281, 3rd Generation Partnership Project, December 2023. Release 18.

[22] George Amponis, Panagiotis Radoglou-Grammatikis, Thomas Lagkas, Wissam Mallouli, Ana Cavalli, Dimitris Klonidis, Evangelos Markakis, and Panagiotis Sarigiannidis. Threatening the 5g core via pfcp dos attacks: the case of blocking uav communications. *EURASIP Journal on Wireless Communications and Networking*, 2022(1):124, 2022.

[23] 3GPP. Gprs tunnelling protocol (gtp) across the gn and gp interface. Technical Report TS 29.060, 3rd Generation Partnership Project, July 2020.

[24] Ze Sheng, Zhicheng Chen, Shuning Gu, Heqing Huang, Guofei Gu, and Jeff Huang. Llms in software security: A survey of vulnerability detection techniques and insights. *ACM Computing Surveys*, 58(5):1–35, 2025.

[25] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 2123–2138, 2018.

[26] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. Aflnet: A greybox fuzzer for network protocols. In *2020 IEEE 13th international conference on software testing, validation and verification (ICST)*, pages 460–465. IEEE, 2020.

[27] Roberto Natella. Stateafl: Greybox fuzzing for stateful network servers. *Empirical Software Engineering*, 27(7):191, 2022.

[28] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. Touching the untouchables: Dynamic security analysis of the lte control plane. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1153–1168. IEEE, 2019.

[29] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. {PentestGPT}: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 847–864, 2024.

[30] Saad Ullah, Praneeth Balasubramanian, Wenbo Guo, Amanda Burnett, Hammond Pearce, Christopher Kruegel, Giovanni Vigna, and Gianluca Stringhini. From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves. *arXiv preprint arXiv:2509.01835*, 2025.

[31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[32] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[33] Altaf Shaik, Robert Jaschek, and Jean-Pierre Seifert. Uncovering hidden paths in 5g: Exploiting protocol tunneling and network boundary bridging. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 231–245, 2025.

[34] MITRE FiGHT. Network boundary bridging: Gtp-u abuse (fgt1599.505). https://fight.mitre.org/techniques/FGT1599.505/, 2024. Accessed: 2024-05-22.

[35] Eunsoo Kim, Dongkwan Kim, CheolJun Park, Insu Yun, and Yongdae Kim. Basespec: Comparative analysis of baseband software and cellular specifications for l3 protocols. In *NDSS*, 2021.

[36] Kai Tu, Abdullah Al Ishtiaq, Syed Md Mukit Rashid, Yilu Dong, Weixuan Wang, Tianwei Wu, and Syed Rafiul Hussain. Logic gone astray: A security analysis framework for the control plane protocols of 5g basebands. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3063–3080, 2024.

[37] Dominik Maier, Lukas Seidel, and Shinjo Park. Basesafe: Baseband sanitized fuzzing through emulation. In *Proceedings of the 13th ACM conference on security and privacy in wireless and mobile networks*, pages 122–132, 2020.

[38] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Breaking lte on layer two. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1121–1136. IEEE, 2019.

[39] Hojoon Yang, Sangwook Bae, Mincheol Son, Hongil Kim, Song Min Kim, and Yongdae Kim. Hiding in plain signal: Physical signal overshadowing attack on {LTE}. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 55–72, 2019.

[40] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Call me maybe: Eavesdropping encrypted {LTE} calls with {ReVoLTE}. In *29th USENIX security symposium (USENIX security 20)*, pages 73–88, 2020.

[41] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy attacks to the 4g and 5g cellular paging protocols using side channel information. *Network and distributed systems security (NDSS) symposium2019*, 2019.

[42] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1082–1099, 2021.

[43] CheolJun Park, Sangwook Bae, BeomSeok Oh, Jiho Lee, Eunkyu Lee, Insu Yun, and Yongdae Kim. {DoLTEst}: In-depth downlink negative testing framework for {LTE} devices. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1325–1342, 2022.

[44] Tianchang Yang, Syed Md Mukit Rashid, Ali Ranjbar, Gang Tan, and Syed Rafiul Hussain. {ORANalyst}: Systematic testing framework for open {RAN} implementations. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1921–1938, 2024.

[45] Mujtahid Akon, Md Toufikuzzaman, and Syed Rafiul Hussain. From control to chaos: A comprehensive formal analysis of 5g's access control. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1081–1100. IEEE, 2025.

[46] Harsh Sanjay Pacherkar and Guanhua Yan. Prov5gc: Hardening 5g core network security with attack detection and attribution based on provenance graphs. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 254–264, 2024.

[47] Tianchang Yang, Sathiyajith K_S, Ashwin_Senthil Arumugam, and Syed_Rafiul Hussain. Feedback-guided api fuzzing of 5g network. 2024.

[48] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm game: Automatic discovery of lte vulnerabilities through documentation analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1197–1214. IEEE, 2021.

[49] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Dongfang Zhao. Seeing the forest for the trees: Understanding security hazards in the {3GPP} ecosystem through intelligent analysis on change requests. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 17–34, 2022.

[50] Yi Chen, Di Tang, Yepeng Yao, Mingming Zha, XiaoFeng Wang, Xiaozhong Liu, Haixu Tang, and Baoxu Liu. Sherlock on specs: Building {LTE} conformance tests through automated reasoning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3529–3545, 2023.

[51] Daniel Klischies, Moritz Schloegel, Tobias Scharnowski, Mikhail Bogodukhov, David Rupprecht, and Veelasha Moonsamy. Instructions unclear: undefined behaviour in cellular network specifications. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3475–3492, 2023.

[52] Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon, Rui Zhang, et al. Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4445–4462, 2024.

[53] Mirza Masfiqur Rahman, Imtiaz Karim, and Elisa Bertino. {CellularLint}: A systematic approach to identify inconsistent behavior in cellular network specifications. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5215–5232, 2024.

[54] Ke Xie, Xingyi Zhao, Yiwen Hu, Munshi Saifuzzaman, Wen Li, Shuhan Yuan, Tian Xie, and Guan-Hua Tu. Cellsecinspector: Safeguarding cellular networks via automated security analysis on specifications. *arXiv preprint arXiv:2512.24682*, 2025.

[55] Le Wang, Chan Chen, Junyi Zhu, Rufeng Zhan, and Weihong Han. Cqllm: A framework for generating codeql security vulnerability detection code based on large language model. *Applied Sciences*, 16(1):517, 2026.

[56] Jonah Ghebremichael, Saastha Vasan, Saad Ullah, Greg Tystahl, David Adei, Christopher Kruegel, Giovanni Vigna, William Enck, and Alexandros Kapravelos. Multi-agent taint specification extraction for vulnerability detection. *arXiv preprint arXiv:2601.10865*, 2026.

[57] Zihan Wu, Jie Xu, Yun Peng, Chun Yong Chong, and Xiaohua Jia. Mulvul: Retrieval-augmented multi-agent code vulnerability detection via cross-model prompt evolution. *arXiv preprint arXiv:2601.18847*, 2026.

[58] Hanzhi Liu, Chaofan Shou, Xiaonan Liu, Hongbo Wen, Yanju Chen, Ryan Jingyang Fang, and Yu Feng. Synthesizing multi-agent harnesses for vulnerability discovery. *arXiv preprint arXiv:2604.20801*, 2026.

[59] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *Proceedings of the 30th international conference on Software engineering*, pages 171–180, 2008.

[60] Salim Al Wahaibi, Myles Foley, and Sergio Maffeis. {SQIRL}:{Grey-Box} detection of {SQL} injection vulnerabilities using reinforcement learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6097–6114, 2023.

[61] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pages 79–90, 2023.

[62] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software testing, verification and reliability*, 22(5):297–312, 2012.

[63] Jinsheng Ba, Marcel Böhme, Zahra Mirzamomen, and Abhik Roychoudhury. Stateful greybox fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3255–3272, 2022.