FivGeeFuzz: Authorization-Aware API Fuzzing of 5G Core Service-Based Interfaces

Anqi Chen[1]✉[0009−0006−5439−3075]*, Riccardo Preatoni[2][0009−0006−6013−5283]*, Alessandro Brighente[2][0000−0001−6138−2995], Mauro Conti[2,3][0000−0002−3612−1934], and Cristina Nita-Rotaru[1][0000−0002−9649−6789]

[1] Northeastern University, Boston, USA {chen.anqi3, c.nitarotaru}@northeastern.edu
[2] University of Padova, Padua, Italy
riccardo.preatoni@phd.unipd.it, {alessandro.brighente, mauro.conti}@unipd.it
[3] University of Örebro, Örebro, Sweden

Abstract. 5G marks a major departure from previous cellular architectures, by transitioning from a monolithic design of the core network to a Service-Based Architecture (SBA) where services are modularized as Network Functions (NFs) communicating with each other via standard-defined HTTP-based APIs, called Service-Based Interfaces (SBIs). These NFs are deployed in private and public cloud infrastructures, and an OAuth-based access control framework restricts how they communicate with each other and obtain access to resources. Given the increased vulnerabilities of clouds to insiders, it is important to perform security testing of the 5G Core services to identify vulnerabilities that may allow attackers to use compromised NFs to obtain unauthorized access to resources.

We present `FivGeeFuzz`, a grammar-based fuzzing framework, coupled with authorization testing modules, to uncover security flaws in 5G Core SBIs. `FivGeeFuzz` fuzzes token requests, performs positive and negative authorization testing, and automatically derives grammars from the 3GPP API specifications to generate malformed, unexpected, or semantically inconsistent inputs.

We evaluate our approach on `free5GC`, the only open-source 5G core implementing Release 17-compliant SBIs with an access control mechanism. Using `FivGeeFuzz`, we discovered 134 invalid token requests accepted, and 14 previously unknown vulnerabilities in `free5GC`, leading to authorization privilege escalation, blocked benign authorization, runtime crashes, improper error handling, and other bugs. All bugs were disclosed to the `free5GC` team and patched.

Keywords: Core Network · 5G · Fuzzing.

* These authors contributed equally to this work.

1 Introduction

The core network of a 5G cellular network architecture manages connections, mobility, security, and services [3, 4]. While 4G relied on point-to-point interfaces and network entities for its core design, 5G marks a fundamental change by introducing a Service-Based Architecture (SBA). Specifically, the 5G Core implements its architectural elements as Network Functions (NFs), interacting one another and/or with consumer entities via standard-defined HTTP-based APIs, called Service Based Interfaces (SBIs). By moving away from proprietary hardware toward cloud-native architectures, SBA allows operators to reduce both capital and operational expenditures. The introduction of the SBA exposes the 5G Core network to new attack vectors as it expands the attack surface (more APIs, more endpoints), it exposes the core to cyberattacks (API abuse and DoS), and it decreases resilience (a breach in one NF can cascade to others NFs). To address some of these attacks, starting from Release 15 [3, 4], 3GPP provides basic security measures for the SBA, defining mechanisms for NF registration, authentication, and access control. For authorization, 3GPP suggests the use of OAuth2 to handle access to resources.

However, the deployment of the 5G Core in cloud-based infrastructures, especially public clouds[1], provides attackers with a novel and previously unavailable access to the core network from inside. Attackers can now assess the presence of NFs, monitor their traffic, interact with them without the need to access a walled infrastructure, and possibly exploit vulnerabilities to gain privileged access via lateral movements or privilege escalation. Therefore, although authorization may defend against external attacks, the 5G core may be vulnerable to attacks coming from legitimate or altered NFs. Given that a significant part of the 5G Core is deployed in clouds and considering the increased vulnerabilities of clouds to insiders, it is important to study the security of the 5G Core services for vulnerabilities that allow attackers to control some of these services.

Only few works studied the security of the 5G Core. Two recent works [9, 10] used formal methods to analyze the design of the 5G Core's access control mechanisms and identified several vulnerabilities. However, while formal methods provide great benefits in understanding protocol design and identifying design bugs and vulnerabilities, they do not guarantee that deployed services do work as intended. Security testing is a complementary approach that captures implementation details and deployment conditions to identify security issues and software bugs that attackers may exploit, such as accessing privileged resources or jeopardizing network availability. To the best of our knowledge, no work has focused on testing all the services in the 5G Core with the consideration of the authorization layer, whereas the ongoing work by Yang et al. [46][2] tested only one interface (out of 6) for only one out of the 10 Core NFs available in `free5GC` and did not consider the authorization testing.

[1] Telefonica Germany [45] and Boost mobile [29], migrated several core network services to public clouds such as AWS.

[2] The code is not publicly available as this is ongoing work.

Prior work on REST API testing has noted that many REST endpoints are restricted to authorized clients, and that supporting such endpoints remains a challenge for many REST API testing tools [22]. In the context of the 5G Core, this challenge is particularly relevant due to the use of an OAuth-based authorization framework that protects SBI services behind access tokens and scope checks. This means that a testing framework must not only generate adversarial inputs, but also obtain and manipulate valid access tokens in order to exercise protected endpoints and reach the NF business logic.

Our contribution. In this paper, we test a broad set of SBI API services of the 5G Core to identify access privilege escalation or core network disruption attacks. To this aim, we leverage fuzzing, a testing technique where the tester feeds a program with unexpected, random, or malformed inputs to cause crashes, misbehaviors, or expose security flaws. We design a fuzzing approach tailored to the 5G Core by solving three domain-specific needs. First, requests among NFs are regulated by an OAuth authorization framework. Their testing should hence not be solely limited to the use of valid authorization tokens, but also test for privilege escalation due to the improper handling of the authentication framework. Second, creating meaningful testing input requires knowledge of the expected input format. The 5G Core SBI defines a set of OpenAPI specifications regulating NF interactions. Thus, an effective fuzzer should leverage this standardized information. Third, identifying security issues and vulnerabilities should be automated and tailored to the specific operations performed by the fuzzer and its testing context.

We propose `FivGeeFuzz`, a grammar-based fuzzing approach to test 5G Core's SBIs. Our approach builds on the fundamental characteristics of the 5G Core SBA: a standard defined service and interaction model, and the use of an OAuth2 authorization framework. We design `FivGeeFuzz` starting from the entire set of 3GPP OpenAPI specifications, generating a grammar that describes the structure and constraints of all SBI requests. This grammar highlights protocol-relevant fields and enables the generation of inputs that conform to the expected formats, ensuring that tests reach the NF logic rather than being discarded due to malformed requests.

In addition, our framework dynamically derives and requests valid access tokens from the 5G Core OAuth server, using scopes extracted from the 3GPP OpenAPI specifications and tailored to each NF endpoint. This enables us to target endpoints that require authorization, both to test scope enforcement (using correctly scoped and deliberately mismatched tokens) and to reach the NF business logic for robustness analysis.

Building on these components, we develop three testing modules: i) an OAuth Token Issuance Testing, which targets bugs and security issues in token distribution; ii) an NF Producer Authorization Testing, which targets bugs and security issues in token handling; and iii) a Runtime Robustness Testing, which targets bugs leading to crashes or system performance degradation. To automate the identification and classification of bugs, we developed two bug Oracles tailored

to authorization and runtime bugs, respectively, that identify bugs based on responses received upon testing NFs.

To test the access control capabilities of the 5G Core, we need to consider implementations that leverage the standard-envisioned OAuth framework. We applied `FivGeeFuzz` to `free5GC`, an open source 5G Core implementation widely adopted as a reference implementation in 5G research and leveraged in industry [21, 39], and the most mature implementation compliant to 3GPP Release 17 with an access control mechanism by OAuth. Other available open-source reference implementations have not implemented the OAuth authorization, but our method can be easily adapted to them once they implement the OAuth authorization.

Thanks to `FivGeeFuzz`, we first uncovered a critical access control vulnerability in `free5GC`, that leads to an attack which we call Cross-Service Token Attack, where a malicious user can use a valid token issued for a service to obtain unauthorized access to resources managed by a different service. After a patch is introduced by the `free5GC` team to fix Cross-Service Token Attack, `FivGeeFuzz` identified another critical vulnerability under Unified Data Management (UDM) that blocks benign access to many UDM services, and another vulnerability that allows attackers to obtain an ideal token (for future misuse) simply by specifying a value for one field in the request to bypass checks. In summary, we identified 14 previously unknown vulnerabilities in `free5GC`: 3 in authorization logic, 7 in the UDM, 1 in the Network Slice Selection Function (NSSF), and 3 in the Policy Control Function (PCF). We responsibly disclosed them to the `free5GC` team: they acknowledged our findings, and created patches for all of them. We also reported these findings to MITRE, with 5 CVEs already assigned and others pending. We summarize our contributions as follows:

- **Authorization-aware 5G Core API fuzzer.** We design `FivGeeFuzz`, a grammar-based testing framework for 5G Core SBI API implementations, which jointly targets OAuth-based authorization and API robustness under unexpected inputs.
- **Two-phase authorization testing for 5G Core APIs.** We introduce a methodology that explicitly tests the 5G Core authorization mechanism, through an OAuth Token Issuance Testing phase on the NRF and an NF Producer Authorization Testing phase with positive and negative tests, evaluated by automated bug oracles.
- **Comprehensive grammar construction and large-scale evaluation.** We build grammars from 48 curated 3GPP OpenAPI specifications and use them to fuzz all 10 `free5GC` core network functions, dynamically deriving access tokens and scopes per endpoint.
- **Discovery of previously unknown authorization and robustness bugs.** `FivGeeFuzz` reveals 14 previously unknown authorization and robustness issues in `free5GC`, all confirmed and patched by the developers, and 5 CVEs assigned.

2 Background

In this section, we provide a brief overview of the 5G Core (5GC) architecture, its service-based interface (SBI), the access control mechanism, and the current 5GC open-source implementations. We then discuss security goals and previous vulnerabilities against 5GC.

2.1 5GC Network Overview

The 5G service architecture includes three main components: the User Equipment (UE), the Radio Access Network (RAN), and the 5GC. UEs are end-user devices, such as smartphones with a SIM card, that communicate with the RAN to access the core network. The 5G RAN, also called Next Generation Node B (gNB), consists of base stations deployed across many locations to connect UEs to the core. The 5GC handles connection and mobility management, user authentication, and subscriber data.

Service-Based Architecture (SBA). In 5G, the core network transitions from a hardware-based design to a Service-Based Architecture, where functionality is split across specialized NFs. Key NFs include the Network Repository Function (NRF), which maintains NF instance profiles and services; the Unified Data Management (UDM) for subscriber data; the Access and Mobility Management Function (AMF) for device registration and mobility; the Policy Control Function (PCF) for enforcing policy rules; and the Network Slice Selection Function (NSSF), which assigns user sessions to the appropriate network slice.

NFs interact by exposing their capabilities as services and communicating through Service-Based Interfaces. The SBI, defined by 3GPP, is a framework of RESTful APIs that includes interfaces such as `Nudm`, `Nnrf`, and `Nsmf`. Each interface groups related services specified using OpenAPI v3 [38] and exchanges JSON over HTTP/2 at the application layer.

NF Communication Model. Each NF can act either as a service producer, by exposing one or more APIs that implement specific capabilities, or as a service consumer, by invoking the services offered by other NFs. For example, the `Nudm` interface connects consumer NFs to the UDM, and includes services such as `Nudm_SDM` for subscription data management and `Nudm_UEAU` for user authentication. Also, each NF service can group many NF operations. For example, AMF provides an NF service `Namf_Communication`, under which there exist several operations like `UEContextTransfer`, `ReleaseUEContext` [5], etc. Both service-level scopes and operation-level scopes can be specified in a service request and response. NFs can communicate directly with each other or indirectly through a Service Communication Proxy (SCP), which acts as a middleware/broker.

2.2 5GC Access Control

While the SBA improves flexibility, scalability, and interoperability across vendors, it also increases the exposure of critical control plane functions to misuse.

Not all NFs should be able to access all services; only authorized NFs should invoke services they are allowed to.

To provide this access control and authorization mechanism, 3GPP selected OAuth 2.0 [24] as the standard for authorizing access to SBI services [3]. OAuth 2.0 supports several grant types for authorization purposes, and 5GC mandates the use of the Client Credentials Grant. The OAuth 2.0 roles are mapped as follows [3, 8]: the NF Service Producer acts as the OAuth 2.0 resource server, the NF Service Consumer acts as the OAuth 2.0 client, and the NRF acts as the OAuth 2.0 authorization server. In commercial products, the vendor documentation for Cisco Ultra Cloud Core [13] and Oracle Communications Cloud Native Core [36] indicates OAuth2 support, although it's not enabled by default in the former, and not specified in the latter.

When OAuth2.0 configuration is enabled, an NF Consumer requesting a service from an NF Producer needs to get an authorization token from the NRF. The access token should include a scope reflecting the request. Only with a valid matched access token, the NF Consumer can obtain access to the target services (Fig. 1).

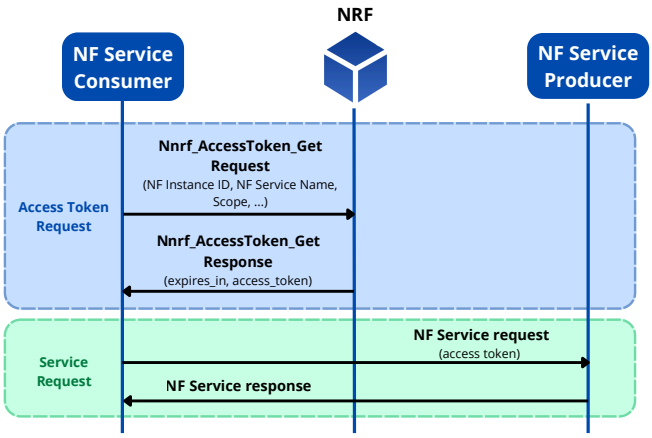

Fig. 1: Two-step authorization workflow in the 5GC.

Prerequisite Steps. Before requesting and getting an access token, the NF Service Consumer should be registered with the NRF through the `Nnrf_NFManagement_NFRegister` Request, using its `NFInstanceID` as the client ID during registration. The NF Service Producer should also be registered with the NRF, and the producer's NF Profile may indicate the resources and the service operations that are allowed per NF type of the NF Service Consumer, or per `NFInstanceID` of the consumer. The NRF and NFs should mutually authenticate each other, where NFs are identified by the `NFInstanceID` recorded in the public key cer-

tificate. The NRF and NF should also share the required credentials (e.g., the NRF's public key or a shared secret) [3, 8].

Access Token Request. Once the prerequisite conditions are satisfied, the NF Service Consumer obtains an access token from the NRF through the `Nnrf_AccessToken_Get` operation (Fig. 1). The request includes the consumer's `nfInstanceId` and `nfType`, as well as the `targetNfType` (or a specific `targetNf InstanceId`) and the requested `scope`, which identifies the service(s) the consumer intends to access. The NF Service Consumer may also include a list of `SNSSAIs` or a list of `NSI IDs` for the expected NF Service Producer instances in the access token request.

Access Token Issued. Upon receiving the request, the NRF checks whether the NF Service Consumer is authorized to access the requested services by comparing the request parameters against the target NF Service Producer's allowed NF types and permitted network slices (known from the NF registration phase). If the request is authorized, the NRF generates a signed access token and returns it to the consumer (Fig. 1).

In the 5GC, the access token, implemented as a JSON Web Token (JWT) [26], includes fields such as the issuer (the `NFInstanceID` of the NRF), the subject NF, the audience NF, the scopes, and the valid period. The integrity of the access token is secured by either a digital signature signed with NRF's private key or Message Authentication Codes (MAC) generated with a shared secret [3,8] based on JSON Web Signature (JWS) [25].

Access Token Validation and Usage. In the second step, the NF Service Consumer includes the access token in its request to the NF Service Producer (Fig. 1). The producer verifies the token's signature using either the NRF's public key or a shared secret. If the producer NF type has an associated list of `SNSSAIs` or `NSI IDs`, it checks that at least one matches its own. It also validates that the audience corresponds to its NF type or `NFInstanceID`, confirms the requested operation is allowed, and ensures the token is not expired [8]. If all checks succeed, the producer provides the requested service. Authorization granularity is determined by the token's scope claim. Service-level scopes authorize access to entire services for an NF type, while operation-level scopes restrict access to specific operations or data subsets. These finer-grained scopes are optional and depend on the NF Service Producer's configuration, enabling precise permissions and reducing over-privileged access in the 5GC.

2.3 Open-Source Implementations of Access Control in 5GC

Currently, there are three open-source 5GC implementations, each with different levels of functionality (see Table 1). OpenAirInterface (OAI) [34] is an open-source software project now maintained by the OpenAirInterface Software Alliance (OSA). `free5GC` [20] has become a Linux Foundation project in 2024. Finally, Open5GS is a community project, also commercialized by NewPlane [32]. `free5GC` [20] and Open5GS [42] are compliant with 3GPP Release 17 [38], while OAI [34] is only compliant with 3GPP Release 15 [3, 4]. `free5GC` has implemented the most core NFs (10), 2 more than Open5GS and 3 more than OAI.

Table 1: Comparison of open-source 5GC implementations.

Platform	Components	Specification	Access Control
`free5GC` [20]	**10 Core NFs** (AMF, AUSF, CHF, NEF, NRF, NSSF, PCF, SMF, UDM, UDR), N3IWF, TNGF, and UPF [19]	Release 17 [38]	Yes
Open5GS [42]	**8 Core NFs** (AMF, AUSF, NRF, NSSF, PCF, SMF, UDM, UDR), SCP, SEPP, BSF, and UPF [35]	Release 17 [38]	No
OAI [34]	**7 Core NFs** (AMF, AUSF, NRF, NSSF, SMF, UDM, UDR) and UPF [33]	Release 15 [4]	No

`free5GC` is the only publicly available 5GC implementation that includes a Service-Based Interface implementation compliant with a recent 3GPP Release 17 and also supports OAuth 2.0 as its access control framework (See Table 1 showing the differences). This makes it a suitable candidate for evaluating both access control enforcement and other robustness in state-of-the-art SBI deployments. While `free5GC` is the most mature implementation, it still lacks some features that are specified in the 3GPP specifications of release 17 [38], such as more sophisticated features including indirect communication where NFs communicate through a proxy (SCP), and roaming where the visitor NRF and home NRF need to coordinate between different network providers.

2.4 Security of 5GC

The security goals for 5G follow typical security properties: confidentiality, integrity, authentication, access control, availability, and anonymity. Some of these properties apply to all of its components, while others are component-specific. For example, integrity applies to all communication involving the UE, RAN, and Core, while privacy applies to the UE when users are roaming through other networks than the home network.

As the focus of this work is the 5GC network, below we describe how an attacker can target its security goals by leveraging the large attack surfaces for 5G represented by the UE, the RAN, and the Core itself, e.g., the SBA architecture.

Security goals of the 5GC. One of the main goals of the core is to protect the SBA architecture. While the communication takes place over HTTP/2, TLS is not mandatory [47], thus communication can take place unencrypted, allowing attackers with access to the cloud where SBA is deployed to monitor, intercept, and change the communication.

From the perspective of core network access control, an NF should only be allowed to access the resources that it is authorized to, and never gain any unauthorized access to other NF services that might cause a sensitive data breach or further attacks due to privilege escalation.

Further, the core network interactions should be robust against any unexpected messages from an attacker who compromised an NF or pretended its identity to interfere with the inter-NF communication.

Previous attacks. Previous work has used both testing and formal analysis to study attacks against the 5GC conducted through the UE or the SBA architecture. Specifically, previous work has shown how an attacker can send unexpected malformed messages from a UE via RAN to the core network to disrupt the service, achieve authentication bypass, or conduct billing fraud [12,15,27,43].

Very few works studied attacks conducted directly against the core. The work in [9,10] used formal methods to analyze the design of the 5GC itself, and found several attacks, including unauthorized access to sensitive information, illegitimate access to services, and denial of service. With respect to finding attacks in 5GC implementations, the only work we are aware of is the on-going work [46], which focused on limited testing of the robustness of the Event Exposure API of the AMF in `free5GC` and found 2 vulnerabilities (`free5GC` currently supports a total of 10 Core NFs that have not been assessed for vulnerabilities).

3 Threat Model

Attacker objectives. We assume that the attacker's goal is to access services they are not authorized to access or to undermine the availability of certain services. The attacker sits under the radar and tries to achieve their goals while interacting with the target service through the REST API standard interface.

Attacker knowledge. We consider an attacker having information about the general SBA architecture of the 5GC, and having access to the standard OpenAPI detailing the service model of the NFs. This information is publicly available [38] and easily obtainable by an adversary.

Attacker capabilities. The attacker can be a compromised NF or a program sending messages to a victim NF. We assume the attacker can detect the presence of a 5GC network in a cloud. As network operators have started deploying their core network in public cloud infrastructures [29,45], an attacker may easily identify the presence of a 5GC network via traffic analysis (e.g., targeting non-encrypted control information [48]), and successively target its NFs.

We assume the attacker has the capability to (1) send an access token request to NRF, (2) send a service request (attaching the token if successfully receiving a valid access token) to a victim NF. That can be possible if the attacker is a compromised NF, if the attacker can steal the identity of a valid NF to send an access token request, or if the communication between services is not running over TLS. Additionally, according to [47], TLS is optional; thus, an attacker can eavesdrop and modify the communication. However, we assume that the NRF is not compromised and the attacker can not change the token itself, as the token is digitally signed by the NRF.

An attacker can compromise an NF by taking advantage of the 5GC cloud deployment [29,45] and the potential risk of cloud container privilege escalation. The attacker may gain privileged access to NFs. For example, as the currently

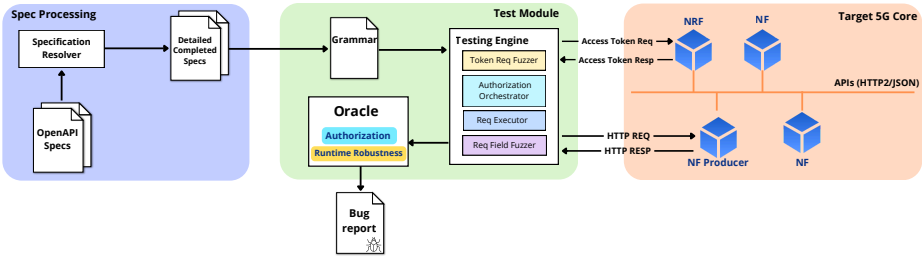

Fig. 2: Overview of `FivGeeFuzz`.

existing implementation leverages Kubernetes and Docker to implement NFs and their underlying network [20, 34, 42], the attacker may exploit known vulnerabilities for privilege escalation and lateral movements (e.g., CVE-2024-21626, CVE-2019-5736, CVE-2023-1260), gaining access to NFs and hence interacting with other NFs at a higher privilege level. This can provide the attacker access to valid authorization tokens.

4 `FivGeeFuzz` System Architecture

In this section we first describe the design goals of `FivGeeFuzz`. Then, we provide a detailed overview of its architecture.

4.1 Design Goals and Overview

`FivGeeFuzz` aims to test 5GC implementations that follow the 3GPP Release 17 [38] specifications. We want our testing platform to achieve several goals:

(G1:) Multi-Layer Authorization Testing. Requests among 5GC NFs leverage a secure authorization mechanism, i.e., OAuth 2.0. Beyond testing API functionalities, our fuzzer should validate the authorization chain in the 5GC, from token issuance to scope enforcement. This requires testing both the centralized authorization server (NRF) and individual NF producers to detect authorization bypass vulnerabilities at different layers of the security stack.

(G2:) Grammar-Based Fuzzing. All 5GC APIs are defined by 3GPP standards, which expose RESTful endpoints with well-structured URIs, payload schemas, and error codes. Our fuzzer should start from these specifications to derive input grammars that generate syntactically valid HTTP requests and derive useful information out of the received responses.

(G3:) Automated Context-Aware Oracles. Manual analysis of fuzzing results is time-consuming and error-prone, particularly when distinguishing between expected errors (e.g., input validation failures, business policy rejections) and security vulnerabilities (e.g., authorization bypass, runtime crashes). Our framework should incorporate automated bug oracles that classify responses based on the testing context to identify authorization bugs and runtime crashes.

The oracles should leverage knowledge of the system architecture, such as the expected ordering of authorization checks relative to input validation, to determine whether unexpected responses indicate security vulnerabilities or expected behavior. For instance, a `400 Bad Request` response when using mismatched authorization scopes may indicate that scope validation was not performed correctly before input parsing, revealing an authorization bypass vulnerability.

4.2 Details of `FivGeeFuzz`

We designed a grammar-based testing framework to systematically evaluate 5GC SBI APIs implementations along two security dimensions: OAuth-based authorization and API robustness under unexpected inputs. To achieve this, `FivGeeFuzz` combines a two-phase authorization testing campaign with a complementary runtime robustness analysis. The first phase, *OAuth Token Issuance Testing* (see Fig. 3), targets the NRF token endpoint and checks whether access tokens are issued for unauthorized or malformed scopes. The second phase, *NF Producer Authorization Testing* (see Fig. 4), tests individual NF APIs to assess how they validate token scopes through negative tests with mismatched scopes, and positive tests with correctly scoped tokens.

As illustrated in Fig. 2, `FivGeeFuzz` is built around a modular architecture that (i) collects and preprocesses the 3GPP OpenAPI specifications to align them with the concrete 5GC deployment, (ii) generates grammar-based test inputs while coordinating OAuth interactions (details explained in below paragraphs), (iii) interacts with a target 5GC deployment whose HTTP request–response traces are inspected by dedicated bug oracles. We did not use branch coverage as a fuzzing metric, as the authorization of 5GC in free5GC is distributed across multiple network functions (i.e., a docker container) and their SBI interactions (e.g., token issuance, service invocation, and token/scope validation). Therefore, branch coverage of an individual process would not faithfully capture whether authorization-relevant behavior was exercised.

Specification Collection and Pre-processing. The goal of this phase is to assemble and prepare the OpenAPI specifications required for automated fuzzing. We began by retrieving the official 3GPP Release 17 OpenAPI specifications for the SBI from the TS 29.x series [2, 38]. Each specification was then bundled [41] into a complete, self-contained document by resolving references and merging related definitions. As part of this process, we apply regex-based substitutions to replace some terms with deployment-specific values, including updating network-related references so that hostnames and ports matched the real topology. The resulting specifications, aligned with the actual testbed configuration, were stored in a dedicated directory and used as the input for the fuzzer. We present a YAML resolving example in Appendix C.

Compilation and Grammar Generation. Next, we process the pre-proces sed YAML specifications to derive input grammars that drive our testing campaign. For each 5GC service, the compilation stage produces a grammar that enumerates the available endpoints and methods, and captures the structure and

types of their parameters and payload fields. These grammars provide syntactically valid request templates that are later instantiated with adversarial values during testing, ensuring that generated inputs respect the basic typing and formatting rules imposed by the OpenAPI schema. 42 out of 48 specifications could be compiled automatically, and two more required some manual intervention: a dedicated grammar is generated for the OAuth token endpoint, which is used to construct token request messages for the *OAuth Token Issuance Testing* phase.

OAuth Token Issuance Testing. As shown in Fig. 3, this part tests the correctness of token issuance of the OAuth server (NRF) of the 5GC, i.e., whether the NRF issues access tokens for unauthorized or malformed scopes. Starting from the token-endpoint grammar, the Token Request Fuzzer generates access-token requests with syntactically or semantically invalid scope values.

To build the input space, the fuzzer uses valid scope values defined in the 3GPP specifications. A request becomes semantically invalid when one of these valid scopes is paired with an incompatible `targetNfType` (for example, using scope `nudr-dr` when the request's `targetNFType` is AMF). The fuzzer systematically explores these mismatched combinations to test whether the NRF incorrectly issues a token. In addition, this fuzzer mutates scope values to produce malformed or corrupted inputs, such as empty strings, random identifiers, or mutated valid scopes. For each generated request, the framework records the HTTP interaction with the NRF, and the Authorization Oracle checks whether the server correctly rejects malformed or semantically incompatible scopes instead of issuing an access token.

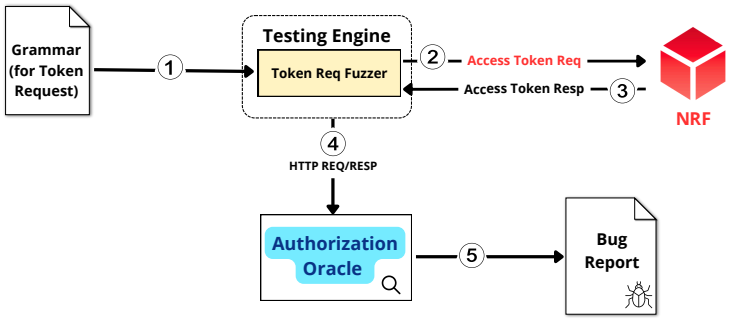

Fig. 3: OAuth Token Issuance Testing Flow.

NF Producer Authorization Testing. As illustrated in Fig. 4, this part tests the token validation at the NF producer side to see if a token with a wrong scope will be accepted and if a correct token will be accepted.

For a given target NF and endpoint, the scope extractor first inspects the OpenAPI specification to determine the scope that should be required and passes that information to the authorization orchestrator. With that, the orchestrator

issues negative and positive testing, dynamically requesting access tokens from the NRF for each test case.

In the *negative-testing* phase, given an endpoint whose expected scope is s_{exp}, the orchestrator iterates over alternative scopes s where $s \neq s_{\text{exp}}$, requests an access token for each such scope, and sends the corresponding API request to the Target NF Producer using that token. These tests reveal missing or incorrectly ordered authorization checks, for instance, when a request with a wrong-scoped-token is parsed or processed instead of being rejected at the authorization layer.

In the *positive-testing* phase, the orchestrator instead requests tokens with the expected scope and uses them against the corresponding API endpoints to check that correctly scoped tokens are accepted and to detect cases where valid tokens with the correct scope are rejected.

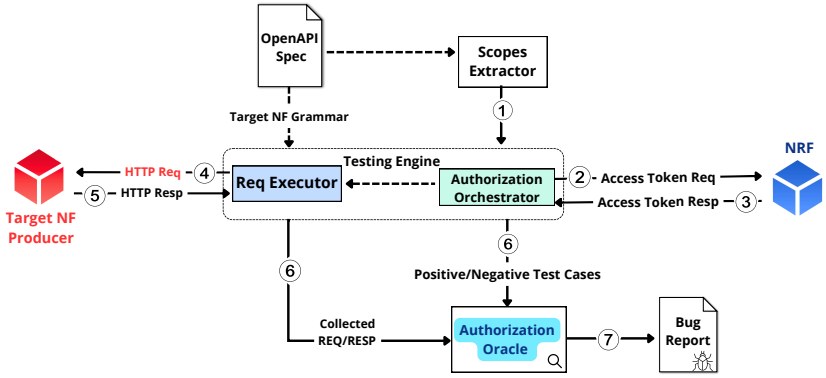

Fig. 4: NF Producer Authorization Testing Flow.

Authorization Bug Oracle. The authorization bug oracle analyzes the collection of HTTP request/response traces produced by both testing flows above and applies different approaches to identify authorization flaws. For the OAuth Token Issuance Testing flow, a bug is reported whenever the NRF returns a successful response and issues an access token for a request that carries an invalid, corrupted, or semantically incompatible `scope`–`targetNfType` combination.

For the NF Producer Authorization Testing flow, the oracle distinguishes between negative and positive tests. In the negative-testing phase, the oracle checks whether a request carrying a deliberately mismatched scope is rejected at the authorization layer. Since TS 29.500 does not mandate `401 Unauthorized` for all authorization failures [7], our oracle treats both `401 Unauthorized` and `403 Forbidden` as valid authorization-layer rejections. Accordingly, a bug is reported only when a request is not rejected at the authorization layer and instead progresses downstream. More concretely, any response other than `401`/`403` is treated as evidence that the request was not rejected at the authorization layer. In particular, `200`/`201` and `500` suggest business-logic reachability, whereas `400`/`404`

indicate that the request may have reached parsing or routing logic instead of being rejected by the authorization layer.

Runtime Robustness Testing. In addition to authorization testing, we have a Request Field Fuzzer that focuses on testing runtime robustness to see if crashes (5xx errors) and other issues occur. The execution flow in the positive-testing phase is reused by attaching a correct token to a fuzzed request to a target NF Producer. This is possible since the correct token allows passing the authorization layer and reaching the NF business logic.

In this phase, the Request Field Fuzzer instantiates the request templates' grammar by filling parameter positions with fuzzed values. Beyond simple fuzzing-dictionary-based value substitution, the fuzzer applies schema-based mutation rules to the request bodies, altering their structure, types, or values (e.g., dropping or duplicating fields, reordering elements, or introducing type mismatches). Through these mechanisms, the NFs are tested with inputs ranging from fully well-formed requests with semantically varied values to malformed or incomplete payloads that intentionally violate the API schema, increasing the chances of exposing robustness bugs.

Runtime Robustness Bug Oracle. The runtime robustness bug oracle inspects the HTTP responses produced during testing and reports a robustness bug whenever it observes a `500 Internal Server Error` status code. Each 500 response is treated as evidence of a potential unhandled exception in the NF implementation and is automatically recorded, together with the triggering request, for subsequent validation and root-cause analysis. In addition, the execution logs of NFs will be searched for keywords like warnings and errors.

Validation and Root-Cause Analysis. All reported issues undergo a manual validation step. This involves reproducing the fault, confirming that it represents an actual bug, and tracing it back to its root cause in the source code.

5 Evaluation

We present our experimental results. We first describe our focus and setup, and then describe the bugs we discovered using `FivGeeFuzz` on `free5GC`.

5.1 Our Focus

In our work, we focus on `free5GC` [20] as a core network implementation, hosted by the Linux Foundation as open-source code, and widely adopted [9,10,15,46]. Our work is conducted on `free5GC` v4.0.1, which is the most mature open-source 5GC implementation, conforming to a recent specification, 3GPP Release 17 [38] (the most up-to-date 3GPP specifications are Release 18 [5,6,8]), and is the only one that implements an OAuth 2.0 access control mechanism, different from Open5GS [42] and OAI [34] (See Table 1).

We want to find access privilege escalation attacks or core network disruption attacks by assuming a compromised NF sending malicious messages (or an attacker obtaining core network communication access to send messages, possible

in a no-TLS scenario). We choose a testing-based methodology against `free5GC`, focusing on the specific perspective of core-NF interactions, instead of UE-RAN attack surfaces chosen by many previous works [15, 43], and we want to systematically cover a broad range of core NF API functionalities, instead of only one API of one NF as done in a previous work [46].

5.2 Experimental Setup

Our evaluation was conducted on a server with 32 virtual CPUs and 64 GB of RAM, running on Ubuntu 22.04 LTS on an x86_64 architecture.

`free5GC` setup. We deployed `free5GC` [20] version 4.0.1 with the help of the free5gc-compose [18] repository, a Docker Compose–based distribution. The stack includes all major 5G Core NFs: AMF, AUSF, CHF, NEF, NRF, NSSF, PCF, SMF, UDM, and UDR. For our experiments, we removed the TNGF from the Docker Compose file, since TNGF is an access gateway function rather than a core NF, handles trusted non-3GPP access, and does not participate in the richer service interactions among core NFs. We adopted the default configurations of the NFs as suggested in the `free5GC` project. The environment uses MongoDB 4.4 [30] and a private Docker network for NF interconnectivity. The deployment was managed with Docker Engine v27.5.1 and Docker Compose v2.36.2.

Specification Corpus Preparation. We curated and bundled a set of 48 OpenAPI specifications, drawn from the 3GPP TS 29.x (Release 17) corpus [38], covering major SBI services across 10 core NFs (AMF, AUSF, CHF, NEF, NRF, NSSF, PCF, SMF, UDM, UDR).

`FivGeeFuzz` is implemented in Python and deployed as a Docker container connected to the `free5GC` private network for direct NF interaction. The framework leverages RESTler [28] (version 9.3.1) for low-level REST API fuzzing primitives, namely its grammar compiler and HTTP request execution engine, while implementing custom components for OAuth token management, authorization-aware fuzzing logic, and security oracles for automated bug classification.

5.3 Multi-Run Experiments

To assess repeated-run stability and the effect of seed selection on the fuzzing process, we repeated the evaluation campaign in three 24-hour runs using three different seed values. The runtime robustness results were stable across runs, with the same set of runtime robustness bugs consistently reproduced. We also manually inspected the reported findings and did not observe false positives in the reproduced bug set. For authorization testing, the identified bugs were reproducible under the corresponding software version and configuration in which each issue was observable. Overall, the authorization-testing results remained stable across the evaluated setups.

5.4 Vulnerabilities Found

Our fuzzing campaign uncovered 14 previously unknown vulnerabilities, falling into four categories: (1) severe authorization logic bugs; (2) unhandled exceptions

leading to runtime panics; (3) muted errors and 2 relevant bugs; (4) incorrect status-code mapping (Some bugs are in Appendix B). While tests included both syntactically malformed and syntactically correct inputs, the majority of vulnerabilities were triggered by syntactically valid inputs, exploiting implementation flaws like improper validation of optional parameters and unsafe type casting. The vulnerabilities are distributed across multiple NFs in `free5GC`.

5.5 Results: Authorization Bugs

(1) Cross-Service Token Attack: *"Any token works on accessing any service".* As described in Section 2.2, when OAuth 2.0 is enabled for inter-communication between NFs, a benign requesting NF must use a corresponding access token to visit the target NF and the target service (green part in Fig. 9 in Appendix C). However, the Authorization Bug Oracle finds that, in `free5GC`, the attacker can use an access token with unmatched scopes to access multiple target NF services (e.g., the yellow part in Fig. 9 in Appendix C).

For example, an attacker sends an access token request with the requester NF type set to AMF, the target NF Type set to UDR, and the scope set to `nudr-dr`. And the access token obtained from the NRF response message includes the scope `nudr-dr` (illustrated as the blue part in Fig. 9 in Appendix C). The attacker can then use the same token to request NRF's `nnrf-disc` service, with the requester NF type, target NF type, and target service scope not matching those inside the token, e.g., specifying UDR as the requester NF type, SMF as the target NF type, and the target service `nnrf-disc`. Despite the mismatch between the token's scope and the requested service, the request can be successfully accepted, and the attacker gets the detailed information about SMF in the response message (illustrated as the yellow part in Fig. 9 in Appendix C).

This shows that the access token validation fails to enforce proper scope checks, thereby allowing unauthorized NF operations. We identified the root cause being a bug in the `free5GC` source code related to error handling in the `VerifyOAuth` function. The relevant code snippet is in `openapi/oauth/oauth.go` (Fig. 5). The issue occurs in the following line: `return errors.Wrapf(err, "verify OAuth scope")`. In this context, `err` is the result of `jwt.ParseWith Claims`, not the `verifyScope` check. Therefore, if `verifyScope` returns false, indicating that the access token does not have the required scope, while assuming the JWT parsing succeeds, then the `err` variable will still be `nil`. This leads to a situation where the function incorrectly returns `nil`, silently ignoring the failed scope verification.

This bug results from forgetting to define a new error variable and assign the closest function's return value to the new error variable. Ensuring the error variable always tracks the latest function return value to avoid any unhandled errors is a common defensive coding routine in network-programming in Go.

Impact: The impact of this bug is broad and severe, since all the NFs rely on this OpenAPI library code for the OAuth 2.0 access control functionalities, specifically the access token scope verification. Due to the buggy logic, a valid access token with an arbitrary valid scope value will be accepted by any NF,

```
func VerifyOAuth(authorization, serviceName, certPath string,) error {
  ...
  token, err := jwt.ParseWithClaims(...)
  if err != nil {return errors.Wrapf(err, "verify OAuth parse")}
  if !verifyScope(token.Claims.(*models.AccessTokenClaims).Scope,
  serviceName) {return errors.Wrapf(err, "verify OAuth scope")}
  return nil
}
```

Fig. 5: OAuth Incorrect Verification from `free5GC`.

since that check will always pass. This could enable a huge number of scenarios of access privilege escalation and potential private data leakage.

(2) UDM Authorization Bug: ***Access blocked even with a valid token.*** After the `free5GC` team developed a patch for the Cross-Service Token Attack, the scope check provides a coarse layer security guard. The attack in the previous section is an example of Negative Authorization Testing (whether a token with a bad scope will be rejected by the NF producer). We also conducted the Positive Authorization Testing (whether a token with a correct scope will be accepted by the NF producer), which led to the finding of the UDM Authorization Bug: *Access blocked even with a valid token.*

First, a valid token is obtained for a UDM service, e.g., UDM SDM service, so the token includes the scope `nudm-sdm`, and the testing code uses it to access UDM SDM service with the ending URL address at `/supi/sm-data`. Our Authorization Bug Oracle observed the request was rejected, and the reply message said `"error": "OAuth scope verification failed: insufficient permissions"`. We observed the same behavior when trying to access other UDM services with a valid, correct token, like `nudm-mt, nudm-niddau`, etc.

We deep dived into the relevant code to analyse the root cause. With the help of customized logging, it turned out that the function `VerifyOAuth` (the patched version of the one in Fig. 5) was called with two arguments `scope` assigned as `nudm-sdm` and `serviceName` assigned as `nudm-ueid`, in the example incident described above. For other UDM services like `nudm-mt, nudm-niddau`, etc., the logging showed the same arguments passed to `VerifyOAuth`.

By following the function call stack, we identified the root cause in the `newRouter` function (Fig. 11 in Appendix C). In the context, the variable `routerAuthorizationCheck` is only defined once. Then, in the following different service branches, the variable is updated to the corresponding authorization check function, and finally updated by `routerAuthorizationCheck = util.NewRouterAuthorizationCheck(models.ServiceName_NUDM_UEID)`. So the variable always points to the check corresponding to `nudm_ueid`, and this explains why the `VerifyOAuth` function always got passed the same value `nudm-ueid` for the `serviceName` argument. Therefore, when a customer contacts UDM with a valid

access token with a scope other than `nudm_ueid`, e.g., `nudm_sdm`, UDM will claim the token cannot pass the authorization checks and reject the service.

Impact: The impact of this bug is broad and severe, since all the services under the UDM rely on the buggy code for the OAuth 2.0 access control functionalities. Due to the bug, even a valid access token with a scope value (not `nudm_ueid`) will be rejected by the UDM since the code always compares the scope with `nudm_ueid`. This could disable a huge number of possible scenarios of benign access and disrupt the normal services provided by UDM.

(3) Scope checks bypassed if `targetNF`==NRF: *"Setting `targetNF` as NRF gives the token in your dream".*

When a customer NF sends an access token request to NRF and specifies the `targetNF` and scope in the request, the relevant code will check if the customer NF is allowed to visit `targetNF` at the scope service. However, our OAuth Token Request fuzzing detected that when `targetNF` is set as NRF while the scope in the request is not an NRF-featured service (e.g.`udm-sdm`), the customer can still successfully obtain an access token. The danger is that the scope check is bypassed when deciding whether to issue the token. With such a malicious token, the attacker may access NFs and services it should not have access to, as the current logic in `free5GC` only checks the scope in the token [17]. For example, according to 3GPP specifications of the UDM [1], only a few NFs are allowed to access UDM, and UDR is not one of them. With our described vulnerability, the attacker with the role of UDR can first send an access token request to NRF, where the targetNF is set as NRF, and the scope is set as `udm-sdm`. Then it obtains an access token with the scope `udm-sdm`, and uses the scope to visit UDM. On the UDM end, the current logic in `free5GC` only checks the scope in the token [17] and finds that `udm-sdm` is allowed, so it decides to allow the service access. In this way, a privilege escalation happens, and private data leakage may result. The root cause is in the code [16] for handling access token requests: when `targetNF` is found to be NRF, it simply skips any future scope checks (Fig. 10 in Appendix C). Due to this bypass, we observed in our experiments 134 invalid token requests with "arbitrary" scope values accepted.

Impact: The impact of this bug is broad and severe, since attackers can get an access token to a service they should not have access to by setting NRF as `targetNF` in the access token request with "arbitrary" scopes. With the obtained token, they gained access to victim NFs. This could enable a huge number of possible scenarios of access privilege escalation and potential data leakage.

5.6 Results: Runtime Panics

Besides the authorization testing, our grammar-based fuzzing campaign revealed several vulnerabilities that cause runtime panics in the affected NFs. These failures, often originating from unhandled exceptions or unsafe type operations, terminate request processing and result in a `500 Internal Server Error` response.

(1) UDM — `Nudm_SDM` Shared Data Retrieval: omission of `supported-features` causes panic.

The `Nudm_SDM` service is responsible for providing UE subscription data to other NFs via the `Nudm` SBI interface. In the Shared Data Retrieval operation (`GET /nudm-sdm/v2/shared-data`), the `supported-features` query parameter is defined in the specification (Fig. 8) as optional (since `required: true` is not specified, and OpenAPI v3.0 treats all request parameters as optional by default [44]) and intended for feature negotiation between client and server. We found that omitting this parameter in an otherwise syntactically valid request causes the UDM to terminate with a runtime panic, returning a `500 Internal Server Error`. Such a condition can be triggered repeatedly to cause NF-level denial of service. Our code inspection revealed that the request is processed by the `HandleGetSharedData` function, which retrieves the parameter using `c.QueryArray("supported-features")` and immediately accesses the first element without verifying the array length. When the parameter is absent, this leads to an out-of-bounds access and an unhandled exception. This behavior violates the API specification, which allows the parameter to be omitted, and highlights the lack of input validation in the UDM implementation.

(2) NSSF — NSSAI Availability Subscription: omission of `expiry` field causes panic.

The `Nnssf_NSSAIAvailability` service enables other NFs to subscribe, via the `Nnssf` SBI interface, to notifications about the availability of network slices, identified by their NSSAI ID. In the Subscription Creation operation (`POST /nnssf-nssaiavailability/v1/nssai-availability/subscriptions`), the `expiry` field is specified as optional, indicating the subscription's expiration time. We found that omitting this field causes the NSSF to terminate with a runtime panic, returning a `500 Internal Server Error`. Our code inspection revealed that the request is handled by the `NssaiAvailabilitySubscriptionCreate` function, which unmarshals the request body into a structure where `expiry` is a pointer (to a `time.Time` object). The implementation immediately calls `.IsZero()` on this pointer without verifying whether it is `nil`. When the field is absent, the pointer remains `nil`, resulting in a nil pointer dereference and an unhandled exception. This behavior violates the API specification, which allows the omission of the `expiry` field, and underscores the lack of proper input validation in the NSSF implementation.

(3) PCF — BDT Policy Creation: unsafe type cast in request handling causes panic.

The `Npcf_BDTPolicyControl` service allows other NFs to request data transfer policies from the PCF via the `Npcf` interface. In the Policy Creation operation (`POST /npcf-bdtpolicycontrol/v1/bdtpolicies`), the request body is deserialized into a `BdtReqData` structure representing the transfer parameters. We found that sending a syntactically valid request triggers a runtime panic in the PCF, returning a `500 Internal Server Error`. The crash stems from an unsafe type assertion in the `HandleCreateBDTPolicyContextRequest` function, which directly casts the result of `deepcopy.Copy(requestMsg)` to `*models.BdtReqData` without checking its type.

(4) PCF — AM Policy Control: nil pointer dereference causes panic.

The `PCF_AMPolicyControl` service allows NFs to manage the access and mobility policies via the `Npcf` interface. In the Policy Association Deletion operation, the handler locates the corresponding UE context via `PCFUeFindByPolicyId` and, if found, deletes the associated policy state before returning a `204 No Content` response (Fig. 12 in Appendix C). We found that when `polAssoId` is not present, the `HandleDeletePoliciesPolsAssoId` function constructs a `ProblemDetails` error in the Gin context, and sends a JSON error response, but fails to abort or return from the handler. As a result, the code continues to execute and calls `delete(ue.AMPolicyData, polAssoId)` even when `ue` is nil, triggering a nil-pointer dereference and causing a runtime panic in the PCF.

Besides, the handler uses `c.JSON(http.StatusNoContent, nil)` for successful deletions, which violates RFC 7231's requirement that `204 No Content` responses must not include a message body. A correct line can be `c.Status(http.StatusNoContent)`.

Impact: The above runtime panics translate into the unavailability of the affected NFs and, in SBA deployments, the failure of a single NF can interrupt dependent procedures, amplifying service interruption. An attacker can repeatedly trigger this NF-level Denial of Service with minimal effort via malicious inputs. For completeness, we include additional bugs and supporting details in Appendix B. We also included several bug code snippets in Appendix C.

6 Related Work

OAuth testing. Existing OAuth tools (e.g., OAuth Tools by Curity [14], OAuth 2.0 Debugger [31], Google OAuth 2.0 Playground [23]) assume a conventional OAuth deployment model where a client sends requests to a single resource server and observes the response. 5G authorization testing requires multiple steps and components to cope with SBA: obtaining tokens from the NRF, and then checking whether producer Network Functions (NFs) enforce them correctly on standardized SBI APIs. Generic tools may support token handling, but they do not directly model this workflow in a 5G-aware way and require custom integration.

Formal Analysis. The work in [10] formally analyzed the OAuth-based access control mechanism of the 5G Core in the direct non-roaming mode as specified in 3GPP Release 17 [2], employed several techniques to construct the abstract model, and tested it against 55 properties. This led to the discovery of five vulnerabilities, including unauthorized access to sensitive information, illegitimate access to services, and denial of service. The work in [9] also worked on the formal analysis of the 5G Core access control system, including the indirect communication mode and 5G roaming. It analyzed 61 security properties across six model components and revealed 10 distinct types of access control attacks, including five newly discovered attacks.

UE and Base Station Input Fuzzing. The work in [12] fuzzed cellular interfaces accessible from a gNB or user device, compiled ASN.1 specifications into structure-aware fuzzing modules, and performed fuzzing on seven open-source

and commercial cores, revealing 119 vulnerabilities. The work in [15] used automata learning methods to infer the FSM of AMF, SMF, and their interaction. It developed a positive feedback cycle of iterative testing and FSM refinement, and evaluated on three open source and one commercial 5GC implementation, uncovering 15 exploitable vulnerabilities that could lead to DoS, authentication bypass, and billing fraud. The work in [43] designed a state transition feedback-guided fuzzing mechanism, tested on 3 5GC implementations (Open5GS [42], `free5GC` [20], and OAI-5GC [34]), and discovered 22 vulnerabilities. The work in [27] targeted only AMF fuzzing, since it exposed the attack surface for UE RAN inputs and also tested on the above 3 implementations.

5G Core Network API Fuzzing. Vanilla RESTler [11] assumes conventional API interactions without modeling the 5G SBA authorization workflow (token issuance from NRF followed by NF-to-NF API calls). Therefore, it cannot directly generate 5GC-meaningful authorization-aware test sequences. The ongoing work in [46] customized Restler [11], targeting to find flaws in 5GC inter-NF communication API implementations, and designed a state-aware feedback-driven fuzzing framework, where the authors use response codes and messages to detect issues or augment the fuzzing corpus with new messages. It was tested on the `free5GC` [20] AMF EventExposure API, and uncovered two previously unknown vulnerabilities. But this work didn't consider the authorization testing, nor did it fully cover all NFs and APIs.

Specification Inconsistency. The work in [40] used a few-shot learning mechanism on domain-adapted large language models, detected inconsistencies of the Non-Access-Stratum (NAS) and the security specifications of 4G and 5G networks, uncovered 157 inconsistencies, and validated them on three open-source implementations and 17 commercial devices.

7 Conclusion

The service-based architecture of the 5G Core network provides significant advantages to operators, but also opens up possibilities for attackers who compromise an NF or are able to obtain a valid token. To demonstrate that such attacks are possible, we proposed `FivGeeFuzz`, a grammar-based fuzzing approach with multi-phase authorization testing and oracles, to test SBIs for authorization failures, crashes, unhandled error messages, etc. We tested against the only open source 5G Core implementation supporting OAuth access control – `free5GC`.

We identified 14 previously unknown vulnerabilities, causing severe authorization bypass vulnerabilities, denial of service, and improper error handling. All bugs were acknowledged and patched by the developers. Our results show that tools for the automatic identification of vulnerabilities tailored to the 5G Core are essential to secure the cellular network critical infrastructure.

Acknowledgments This work was supported by the European Union - NextGenerationEU under the National Recovery and Resilience Plan (PNRR).

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

A Appendix: Ethical Considerations

The work was done on our local computing machines in a containerized environment, not to harm any other parties. Our research target is an open-sourced implementation, and we ran our own deployment for testing purposes. We have disclosed our findings to the `free5GC` team, and the team members have acknowledged all of them. All patches were completed already before the submission of this paper. We also reported our findings to MITRE to request the CVE IDs, we have obtained 5 CVEs, and waiting to hear back about the others.

We have open-sourced our code at the link `https://anonymous.4open.science/r/fivgeefuzz/`.

B Appendix: Additional Results

UDM — Nudm_SDM Session Management: malformed `single-nssai` causes runtime panic. In the Session Management data retrieval operation (`GET /nudm-sdm/v2/{supi}/sm-data`), the `single-nssai` query parameter is optional and, when present, is used to filter results. We found that putting a malformed value for this parameter causes the UDM to respond with `500 Internal Server Error`. The processing logic attempts to unmarshal the `single-nssai` and when that caused a JSON UNMARSHAL error, it simply claimed internal server error. When `single-nssai` is present but malformed, the service should have defensive code to detect that and return a client error (`400 Bad Request`). **Impact:** This attack reveals incorrect code logic: when faced with a malformed value, it fails to check that and classify it as client-side bad behavior, but fails with a server-side error. Propagating internal server errors might cause management intervention efforts to investigate the error root cause, resulting in unnecessary service downtime with the same effect as DoS attacks.

Muted Errors Due to Buggy Code. The Nudm_SubscriberDataManag-ement service exposes an optional `plmn-id` parameter on the `GET /nudm-sdm/v2/{supi}/sm-data` endpoint, which is supposed to be used to filter subscriber data. In our experiments, we noticed that invoking this operation without specifying `plmn-id` caused the UDM logging an internal warning about an unmarshal error. While for the same request with an invalid `plmn-id` value, UDM returned a `200 OK` despite a similar unmarshal error warning in logs.

The weird behavior is caused by two bugs in `getPlmnIDStruct` (Fig. 6). The function implies `queryParameters["plmn-id"] != nil` as "parameter present," but in fact the underlying value can be `[]string""` (earlier the function `HandleGetSmData` has the line `query.Set("plmn-id", c.Query("plmn-id"))` and assigned the empty string value), so that even though we did not include `plmn-id` in the request, the branch is still entered. The root cause is in forgetting the case of comparing `[]string""` with `nil`. Later, when the `plmn-id` JSON Unmarshal

```
func (s *Server) getPlmnIDStruct(queryParameters url.Values,) (
plmnIDStruct *models.PlmnId, problemDetails *models.ProblemDetails) {
  if queryParameters["plmn-id"] != nil {
    plmnIDJson := queryParameters["plmn-id"][0]
    plmnIDStruct := &models.PlmnId{}
    err := json.Unmarshal([]byte(plmnIDJson), plmnIDStruct)
    if err != nil {logger.SdmLog.Warnln("Unmarshal Error in
    targetPlmnListtruct: ", err)}
    return plmnIDStruct, nil
} ...}
```

Fig. 6: Muted errors in the UDM code.

fails, the code logs the error and forgets to abort but still returns a non-nil `PlmnId` pointer with `problemDetails` as nil. Therefore, both missing and malicious `plmn-id` values are allowed as valid input to be handled further while the errors are muted, and the request is accepted with a success status code, even though the invalid `plmnid` should be rejected with a `400 Bad Request`.

UDM — Nudm_SDM ID Translation: non-existent `ueId` yields 500 status code. In the ID Translation Result operation (`GET /nudm-sdm/v2/ueId/id-translation-result`), the request retrieves identity translation data for a given UE identifier. We found that when the `ueId` does not exist in the UDR, the UDM responds with `500 Internal Server Error` instead of propagating the `404 Not Found` returned by the UDR. The request is forwarded to the UDR, which correctly returns a `404`. However, the UDM does not handle the UDR's `404 Not Found` response correctly and instead returns a generic `500 Internal Server Error` with a `SYSTEM_FAILURE` cause. **Impact:** It fails to utilize the `404 Not Found` response from UDR as the correct reply, but claims the problem as a server-side error. Propagating internal server errors might cause intervention efforts to investigate the root cause, resulting in unnecessary service downtime with the same effect as DoS attacks.

UDM — Nudm_SDM Session Management: missing `single-nssai` yields improper 500. In the Session Management data retrieval operation (`GET /nudm-sdm/v2/{supi}/sm-data`), the `single-nssai` query parameter is optional and, when present, is used to filter results. We found that omitting this parameter causes the UDM to respond with `500 Internal Server Error`. The processing logic attempts to unmarshal the `single-nssai` value unconditionally without first checking whether the parameter is present. When it is absent (empty string), the unmarshal fails and the error propagates. In fact, if `single-nssai` is absent, since it is optional, the request should be processed normally without `NSSAI` filtering.

UDM — Nudm_SDM Shared Data Subscription: invalid parameters yield improper 500. In the Shared Data Subscription operation (`POST /nudm-sdm/v2/shared-data-subscriptions`), the request body contains parameters such as the NF instance identifier, a callback reference, and monitored

resource URIs. These are defined as strings in the specification but may require additional semantic validation by the service logic. We found that when the request body contains syntactically valid but semantically invalid JSON values, the UDM responds with `500 Internal Server Error`. The processing logic deserializes the request body without enforcing strict validation of parameter correctness; as a result, invalid values trigger failures during processing and lead to a server-side error.

C Appendix: Additional Figures and Examples

YAML resolving example. We show an example of how we use the different specifications to resolve dependencies. In Fig. 7, we show a fragment from the original `TS29503_Nudm_SDM.yaml` (the complete specification has around 4,000 lines of YAML code). As shown in Fig. 7, the `url` is incomplete since the actual server IP address is not filled, and the `schema` field refers to another YAML specification file. After the preprocessing, however, as shown in Fig. 8, the `url` address is filled with the IP of the UDM container used in our experiment setup, and the `schema` field is resolved by referring to the content in the self YAML file, since the relevant information from `TS29571_CommonData.yaml` has been merged into the self specification, now contains nearly 7,000 lines of YAML code.

```
servers:
  - url: '{apiRoot}/nudm-sdm/v2'
    variables:
      apiRoot: ...
...
paths:
  /shared-data:
    get:
      ...
           schema:
             $ref: >-
                TS29571_CommonData.yaml#/components/schemas/
                     SupportedFeature
```

Fig. 7: Nudm_SDM YAML specification [37] (partial, before resolving).

Cross-Service Token Attack. Fig. 9 shows one example attack procedure of Cross-Service Token Attack.

Buggy Code snippet. Fig. 11 shows the bug code of the (2) authorization bug in 5.5. Fig. 10 shows the bug code of the (3) authorization bug in 5.5. Fig. 12 shows the bug code of the (4) panic bug in 5.6.

```
servers:
- url: http://udm:8000/nudm-sdm/v2
...
paths:
  /shared-data:
    get:
        ...
        parameters:
        - ...
        - name: supported-features
          in: query
          description: Supported Features
          #required: false
          schema:
            $ref: >-
                #/components/schemas/SupportedFeatures
```

Fig. 8: Nudm_SDM YAML specification (partial, after resolving).

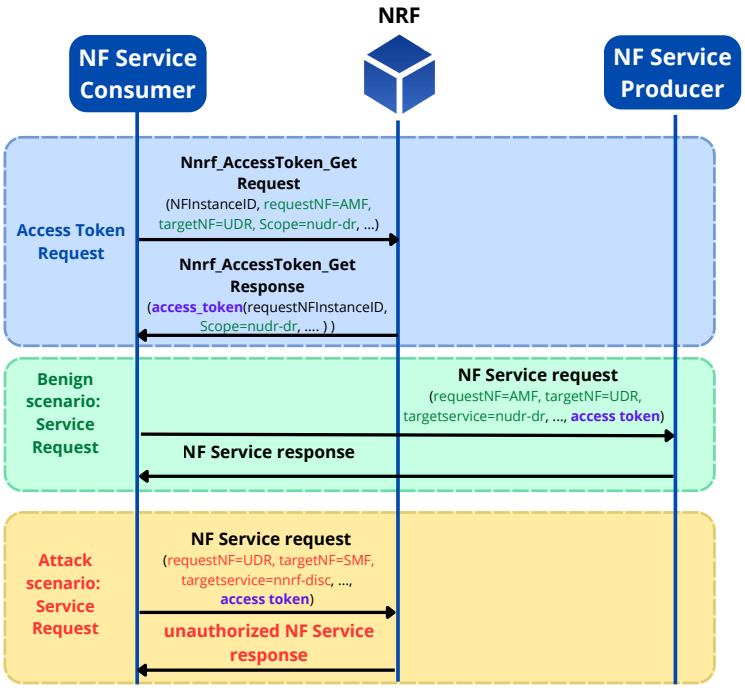

Fig. 9: Cross-Service Token Attack.

```
func (p *Processor) AccessTokenScopeCheck(req
models.NrfAccessTokenAccessTokenReq) *models.AccessTokenErr {
    // Check with nf profile ...
    // Verify NF's certificate with root certificate ...
    if reqTargetNfType == "NRF" { return nil }
    ...
}
```

Fig. 10: Access token generation code from free5GC.

```
func newRouter(s *Server) *gin.Engine {
  ...
  // EE...
  routerAuthorizationCheck := util.NewRouterAuthorizationCheck(
  models.ServiceName_NUDM_EE)
  udmEEGroup.Use(func(c *gin.Context) {routerAuthorizationCheck.Check(
  c, s.Context())})
  AddService(udmEEGroup, udmEERoutes)
  // other UDM services like SDM, ...
  // UEID...
  routerAuthorizationCheck = util.NewRouterAuthorizationCheck(
  models.ServiceName_NUDM_UEID)
  udmUEIDGroup.Use(func(c *gin.Context) {routerAuthorizationCheck.Check(
  c, s.Context())})
  AddService(udmUEIDGroup, udmUEIDRoutes)
  return router
}
```

Fig. 11: server.go in UDM from free5GC.

```
func (p *Processor) HandleDeletePoliciesPolAssoId(c *gin.Context,
polAssoId string,) {
  ...
  ue := p.Context().PCFUeFindByPolicyId(polAssoId)
  if ue == nil ||
  ue.AMPolicyData[polAssoId] == nil {
      problemDetails := util.GetProblemDetail("polAssoId not found in PCF"
          , util.CONTEXT_NOT_FOUND)
      c.Set(sbi.IN_PB_DETAILS_CTX_STR, problemDetails.Cause)
      c.JSON(int(problemDetails.Status), problemDetails)
  }
  delete(ue.AMPolicyData, polAssoId)
  c.JSON(http.StatusNoContent, nil)
}
```

Fig. 12: Buggy PCF am policy control code.

References

1. 3GPP: 5G; 5G System; Unified Data Management Services; Stage 3. Tech. rep., 3rd Generation Partnership Project (3GPP) (2019), https://www.etsi.org/deliver/etsi_ts/129500_129599/129503/15.02.01_60/ts_129503v150201p.pdf
2. 3GPP: Release 17 Description; Summary of Rel-17 Work Items. Tech. Rep. TR 21.917, 3rd Generation Partnership Project (3GPP) (2022), https://www.3gpp.org/ftp/Specs/archive/21_series/21.917/
3. 3rd Generation Partnership Project (3GPP): 5g; security architecture and procedures for 5g system. Tech. rep., 3GPP TS 33.501 (May 2019), note = https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.04.00_60/ts_133501v150400p.pdf,
4. 3rd Generation Partnership Project (3GPP): 5g; system architecture for the 5g system (5gs). Tech. rep., 3GPP TS 23.501 (Apr 2019), note = https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/15.05.00_60/ts_123501v150500p.pdf,
5. 3rd Generation Partnership Project (3GPP): 5g; procedures for the 5g system (5gs). Tech. rep., 3GPP TS 23.502 (May 2024), note = https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/18.05.00_60/ts_123501v180500p.pdf,
6. 3rd Generation Partnership Project (3GPP): 5g; system architecture for the 5g system (5gs). Tech. rep., 3GPP TS 23.501 (May 2024), note = https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/18.05.00_60/ts_123501v180500p.pdf,
7. 3rd Generation Partnership Project (3GPP): 5G; 5G System; Technical Realization of Service Based Architecture; Stage 3. Tech. rep., 3GPP TS 29.500 (Jun 2025), release 18. Available at: https://www.etsi.org/deliver/etsi_ts/129500_129599/129500/18.09.00_60/ts_129500v180900p.pdf
8. 3rd Generation Partnership Project (3GPP): 5g; security architecture and procedures for 5g system. Tech. rep., Technical Specification (TS) 33.501, ETSI (Apr 2025), note = https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/18.09.00_60/ts_133501v180900p.pdf,
9. Akon, M., Toufikuzzaman, M., Hussain, S.R.: From Control to Chaos: A Comprehensive Formal Analysis of 5G's Access Control . In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1081–1100. IEEE Computer Society, Los Alamitos, CA, USA (May 2025). https://doi.org/10.1109/SP61157.2025.00141, https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00141
10. Akon, M., Yang, T., Dong, Y., Hussain, S.R.: Formal analysis of access control mechanism of 5g core network. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. p. 666–680. CCS '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3576915.3623113, https://doi.org/10.1145/3576915.3623113
11. Atlidakis, V., Godefroid, P., Polishchuk, M.: RESTler: Stateful REST API Fuzzing. In: Proceedings of the 41st International Conference on Software Engineering (ICSE). IEEE/ACM (2019), https://patricegodefroid.github.io/public_psfiles/icse2019.pdf
12. Bennett, N., Zhu, W., Simon, B., Kennedy, R., Enck, W., Traynor, P., Butler, K.R.B.: Ransacked: A domain-informed approach for fuzzing lte and 5g rancore interfaces. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. p. 2027–2041. CCS '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3658644.3670320, https://doi.org/10.1145/3658644.3670320

13. Cisco: Ultra Cloud Core 5G Access and Mobility Management Function, Release 2025.04 - Configuration and Administration Guide: Chapter: OAuth2 Client Authorization Support to NRF. https://www.cisco.com/c/en/us/td/docs/wireless/ucc/amf/2025-04/config-and-admin/ucc-5g-amf-configuration-and-administration-guide-release-2025-04/m_5g-amf_oauth2-client-support-nrf.html (2025), accessed: 2026-03-31
14. Curity: The OAuth Laboratory: Curity. https://curity.io/oauth-tools/ (2026), accessed: 2026-03-31
15. Dong, Y., Yang, T., Ishtiaq, A.A., Rashid, S.M.M., Ranjbar, A., Tu, K., Wu, T., Mahmud, M.S., Hussain, S.R.: Corecrisis: Threat-guided and context-aware iterative learning and fuzzing of 5g core networks. In: 34th USENIX SECURITY Symposium. Usenix Security '25, USENIX Association (2025), https://www.usenix.org/system/files/conference/usenixsecurity25/sec25cycle1-prepub-1292-dong-yilu.pdf
16. Free5GC: Access token implementation code in Free5GC. https://github.com/free5gc/nrf/blob/main/internal/sbi/processor/access_token.go, accessed: 2025-8-18
17. Free5GC: Access token Usage Validation code in Free5GC. https://github.com/free5gc/openapi/blob/main/oauth/oauth.go, accessed: 2025-8-18
18. Free5GC: Free5GC Compose. https://github.com/free5gc/free5gc-compose, accessed: 2025-8-20
19. Free5GC: Open source 5G core network based on 3GPP R15. https://github.com/free5gc/free5gc (2025), accessed: 2025-08-22
20. free5GC Project: free5GC: An Open-Source 5G Core Network Implementation. https://free5gc.org/ (2025), accessed: 2025-07-21
21. Fujitsu: Consortium of Japan partners successfully promote domestic production and cost reduction for 5G core technology, the basis for next-generation communication standards. https://www.fujitsu.com/global/about/resources/news/press-releases/2022/1124-01.html?utm_source=chatgpt.com (2025), accessed: 2025-08-20
22. Golmohammadi, A., Zhang, M., Arcuri, A.: Testing restful apis: A survey. ACM Trans. Softw. Eng. Methodol. **33**(1) (Nov 2023). https://doi.org/10.1145/3617175, https://doi.org/10.1145/3617175
23. Google: Google OAuth 2.0 Playground. https://developers.google.com/oauthplayground/ (2026), accessed: 2026-03-31
24. Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749 (Oct 2012), https://datatracker.ietf.org/doc/html/rfc6749#section-4.4
25. Jones, M.B., Bradley, J., Sakimura, N.: JSON Web Signature (JWS). RFC 7515 (May 2015), https://datatracker.ietf.org/doc/html/rfc7515
26. Jones, M.B., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC 7519 (May 2015). https://doi.org/10.17487/RFC7519, https://www.rfc-editor.org/info/rfc7519
27. Mancini, F., Da Canal, S., Bianchi, G.: Amfuzz: Black-box fuzzing of 5g core networks. In: 2024 19th Wireless On-Demand Network Systems and Services Conference (WONS). pp. 17–24 (2024). https://doi.org/10.23919/WONS60642.2024.10449510
28. Microsoft: RESTler is the first stateful REST API fuzzing tool for automatically testing cloud services through their REST APIs and finding security and reliability bugs in these services. https://github.com/microsoft/restler-fuzzer (2025), accessed: 2025-08-20

29. Mobile, B.: The Boost Mobile Network. https://help.boostmobile.com/docs/boost-mobile-network, accessed: 2025-8-18
30. MongoDB: MongoDB 4.4 Release: MongoDB Database 4.4. https://www.mongodb.com/new-2020, accessed: 2025-8-21
31. Nate Barbettini: OAuth 2.0 Debugger Test OAuth 2.0 requests and debug responses. https://oauthdebugger.com/ (2026), accessed: 2026-03-31
32. NewPlane: Supercharge Open5GS Now. https://newplane.io/ (2025), accessed: 2025-08-25
33. Open Air Interface: OAI-CN 5G is an implementation of the 3GPP specifications for the 5G Core Network. https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-amf/-/wikis/home (2025), accessed: 2025-08-22
34. Open Air Interface: OpenAirInterface | 5G software alliance for democratising wireless innovation. https://openairinterface.org/oai-5g-core-network-project/ (2025), accessed: 2025-08-20
35. Open5GS: Open5GS Quickstart. https://open5gs.org/open5gs/docs/guide/01-quickstart/ (2025), accessed: 2025-08-22
36. Oracle: Cloud Native Core, Network Repository Function User Guide. https://docs.oracle.com/en/industries/communications/cloud-native-core/3.24.2/nrf_user_guide/introduction1.html#GUID-0E66ACCE-0D87-46B4-B518-F7268BDBD360 (2026), accessed: 2026-03-31
37. Partners, G.O.: RESTful APIs of main Network Functions in the 3GPP 5G Core Network (The API YAML specification for Nudm_SDM from GPP release 17). https://github.com/jdegre/5GC_APIs/blob/Rel-17/TS29503_Nudm_SDM.yaml, accessed: 2025-8-21, another resource link is at https://forge.3gpp.org/rep/all/5G_APIs/-/blob/REL-17/TS29503_Nudm_SDM.yaml?ref_type=heads
38. Partners, G.O.: OpenAPI Specification Files for 3GPP 5G Core Network (Release 17). https://github.com/jdegre/5GC_APIs/tree/Rel-17 (2024), accessed: 2025-08-14, , another resource link is at https://forge.3gpp.org/rep/all/5G_APIs/-/blob/REL-17/
39. Platform9: Running Free5GC on Platform9 Managed Kubernetes. https://platform9.com/blog/running-free5gc-on-platform9-managed-kubernetes/?utm_source=chatgpt.com (2025), accessed: 2025-08-20
40. Rahman, M.M., Karim, I., Bertino, E.: Cellularlint: a systematic approach to identify inconsistent behavior in cellular network specifications. In: Proceedings of the 33rd USENIX Conference on Security Symposium. SEC '24, USENIX Association, USA (2024)
41. Redocly: Redocly command: bundle. https://redocly.com/docs/cli/v1/commands/bundle, accessed: 2025-8-21
42. Sukchan Lee: Open5GS: Open Source implementation for 5G Core and EPC, i.e. the core network of LTE/NR network (Release-17). https://open5gs.org/ (2025), accessed: 2025-08-20
43. Sun, Y., Liu, X., Sun, Q., Wang, J., Tian, L., Liu, J.: 5gc-fuzz: Finding deep stateful vulnerabilities in 5g core network with black-box fuzzing. In: IEEE International Conference on Computer Communications (2025)
44. Swagger: OpenAPI Guide(version 3.0): Describing Parameters. https://swagger.io/docs/specification/v3_0/describing-parameters/#required-and-optional-parameters, accessed: 2025-8-21
45. o2 Telefónica: Our 5G Network. https://www.telefonica.de/network/5g-english.html#:~:text=Germany%20must%20be%20among%20the,out%20as%20quickly%20as%20this., accessed: 2025-8-18

46. Yang, T., S, S.K., Arumugam, A.S., Hussain, S.R.: Feedback-guided api fuzzing of 5g networks. In: Proceedings of the Workshop on Security and Privacy of Next-Generation Networks (FutureG). ISOC, San Diego, CA, USA (2025). https://doi.org/10.14722/futureg.2025.23071, https://dx.doi.org/10.14722/futureg.2025.23071
47. Zeidler, O., Sturm, J., Fraunholz, D., Kellerer, W.: Performance evaluation of transport layer security in the 5g core control plane. In: Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks. p. 78–88. WiSec '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3643833.3656140, https://doi.org/10.1145/3643833.3656140
48. Zhang, Y., Wan, T., Yang, Y., Duan, H., Wang, Y., Chen, J., Wei, Z., Li, X.: Invade the walled garden: Evaluating gtp security in cellular networks. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1159–1177. IEEE (2025)