CoreCrisis: Threat-Guided and Context-Aware Iterative Learning and Fuzzing of 5G Core Networks

Yilu Dong, Tianchang Yang, Abdullah Al Ishtiaq, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Md Sultan Mahmud, Syed Rafiul Hussain
The Pennsylvania State University
{yiludong, tzy5088, abdullah.ishtiaq, szr5848, aranjbar, kjt5562, tvw5452, mqm7099, hussain1}@psu.edu

Abstract

We develop CoreCrisis, a stateful black-box fuzz-testing framework for 5G core network (5GC) implementations. Unlike previous stateful security analysis efforts of cellular networks which rely on manually-crafted, static test inputs and are limited to identifying only logical errors, CoreCrisis employs a dynamic two-step approach. Initially, CoreCrisis builds an initial finite state machine (FSM) representation of the 5GC's implementation using only benign (i.e., positive) inputs with its efficient and scalable *divide-and-conquer* and *property-driven equivalence checking* learning. During fuzzing, it utilizes the learned FSM to target underexplored states and introduces state-aware mutations to generate and test attacking (i.e., negative) inputs. Based on the responses observed from the core network, CoreCrisis continuously refines the FSM to better guide its exploration and find vulnerabilities. Evaluating CoreCrisis on three open-source and one commercial 5GC implementations, we identified 7 categories of deviations from the technical specifications and 13 crashing vulnerabilities. These logical and crashing vulnerabilities lead to denial-of-service, authentication bypass, and billing fraud.

1 Introduction

Unlike the monolithic architectures of previous generations of cellular networks, 5G introduces a service-based architecture (SBA) for its core networks. This SBA divides the core network's functionalities into various network functions (NFs). Each NF executes a distinct set of operations as an individual component, providing Application Programming Interface (API) access to other NFs. This modular approach allows operators to distribute NFs with more flexibility. The 5G technical specifications [13–16] defined by the 3rd Generation Partnership Project (3GPP) aim to provide guidelines for implementing the 5G core networks. However, previous research [18, 23–25, 37, 39, 45, 56, 60] found that these specifications are sometimes ambiguous and underspecified. Furthermore, the absence of a reference implementation often results in misinterpretations of the standards. These ambiguities, underspecifications, and misinterpretations can result in exploitable logical bugs, including authentication bypass and billing fraud in the 5G core network implementations. Additionally, flaws or oversights in implementations could also lead to errors that result in system crashes due to missing checks or mishandling of unexpected inputs, causing Denial-of-Service (DoS) for all user devices connected to the core network and requiring a complete restart of the affected NF to restore service. Previous research has shown that 4G core network implementations are prone to these logical and crashing issues [39, 45, 50]. However, due to the unique challenges posed by 5GC's software-centric distributed design, no systematic security analysis of 5GC implementations has been conducted before. To bridge this gap, we aim to answer the following research question in this work: *can we develop a systematic framework that analyzes and uncovers implementation flaws in 5G core network implementations?*

5GC is typically closed from external traffic, preventing attackers from gaining direct access through its internal APIs. Instead, by obtaining a valid Subscriber Identification Module (SIM) card and gaining root control of the user device, the attacker can send arbitrary malicious messages to the core network. These messages then trigger the internal API interactions between NFs. Since access to the internal details of these implementations is often limited in commercial settings, testing can only be performed without access to the source code, compiled binaries, or runtime environments of the NFs. Therefore, we take a black-box fuzzing approach, analyzing feedback based on responses observed directly from the UE. We target the 5G Non-Access Stratum (NAS) protocol between user devices and the core network, as it is crucial for user authentication, security context setup, and session management.

Existing works analyzing the security of cellular protocols generally take two directions, *stateless* or *stateful* testing. Stateless analyses often struggle to understand the highly stateful cellular protocols and are inefficient at exploring the state space. These methods either depend on complete man-

ual examination [45] or utilize predefined test cases [63]. In contrast, some stateful testing approaches [24,39,42,60] leverage automata learning algorithms to construct a finite state machine (FSM) representation of the implementation. These approaches require a predefined static set of input symbols to construct and test the FSM. As a result, they are limited to identifying vulnerabilities detectable within the predefined symbol set, and assembling these symbol sets requires manual efforts from human experts. Additionally, the limited number of tests these approaches can employ restricts them to detecting only logical errors. A recent approach [12] aiming to identify logical vulnerabilities addresses this limitation and enables the fuzzer to test symbols beyond the initial symbol set. However, it cannot dynamically update the initial FSM to explore deeper protocol states. Moreover, none of these existing approaches can detect memory safety violations and crashes, which usually require testing a wide range of inputs. Conversely, approaches that simultaneously generate diverse inputs and learn states [52] use ad-hoc learning methods that depend on randomness, resulting in poor state coverage. Furthermore, these methods lack protocol semantics, restricting them to only finding memory bugs but not protocol violations.

Proposed approach. To overcome the limitations of previous methods, we design and implement CoreCrisis, a context-aware, black-box approach employing a *two-step* iterative learning and fuzzing. In the first step, CoreCrisis constructs an initial state representation, i.e., a skeleton of the implementation's state machine learned using only benign symbols, i.e., messages that are correctly formed and expected by the core network. To address the challenges of high learning time and the complexity of learning intricate 5GC implementations, CoreCrisis introduces two novel methods: (1) *divide-and-conquer*, which partitions the FSM learning process into smaller, more manageable segments based on the functional division of NFs; and (2) *property-driven equivalence checking*, which leverages key *compliance properties* of 5G control-plane procedures to efficiently refine the hypothesis FSM and accelerate the learning process.

During the second step, CoreCrisis references and refines the learned FSM to generate state-aware mutated test inputs (i.e., attacking symbols) that can trigger unexpected behavior and explore previously underexplored state spaces. Since this second stage does not impose any restrictions on the number of attacking symbols CoreCrisis tests, it can explore a wider search space and find deeply rooted vulnerabilities. To avoid generating test sequences that result in meaningless cycles or early rejections, CoreCrisis prioritizes less-explored or potentially vulnerable states and constructs state-specific test messages to maximize coverage. It also designs a grammar-aware message mutation strategy to ensure generated test messages are well-formed, meeting both syntactic (e.g., type, length, structure) and semantic (e.g., value constraints, inter-field dependencies) correctness. CoreCrisis utilizes a 5G-specific, side-channel-based probing method to collect feedback and detect vulnerabilities from the limited black-box response the core network provides to the UE. If CoreCrisis observes a previously unseen response, it infers that the corresponding test message has either discovered a new transition to an existing state or uncovered a new state, and updates the FSM to represent this finding. The reference FSM is used to guide CoreCrisis's test generation, while results from these tests in turn are used to refine the FSM, creating an *iterative cycle of testing and refining*.

Findings. Applying CoreCrisis to 3 open-source and 1 commercial 5GC implementations, we uncovered 3 types of logical violations that can lead to authentication bypass and billing fraud, 7 additional types of protocol deviations, and 13 crashing vulnerabilities, leading to denial-of-service (DoS). Eight CVEs have been assigned to 14 of our findings.

Responsible disclosure and open-sourcing. We reported all identified issues to corresponding developers and are working with them to identify root causes and design fixes. Patches for 11 of the 16 identified vulnerabilities are already merged in the latest release. We open-sourced CoreCrisis at the following link: https://doi.org/10.5281/zenodo.14735880 and also on GitHub [4].

Contributions. We make the following contributions:

- We develop CoreCrisis, the first stateful systematic fuzzing framework for 5G core network implementations.
- We design an efficient *divide-and-conquer* and *property-driven equivalence checking* style automata learning methods to infer the FSM of 5G core network functions.
- We develop a positive feedback cycle of *iterative testing and refining* and novel *state-aware mutation* techniques to effectively generate stateful test cases for 5GC.
- We evaluated CoreCrisis on three open-source and one commercial 5GC implementation, uncovering 15 exploitable vulnerabilities (both logical and crashing) and 7 additional types of logical deviations. These issues could lead to DoS, authentication bypass, and billing fraud.

2 Preliminaries

5G core network (5GC). A 5G system consists of three main components (Figure 1): User Equipment (UE), the Radio Access Network (RAN), and the 5G core network (5GC). UEs are end-user communication devices, such as smartphones. The 5G RANs, also known as Next Generation Node B (gNBs), are base stations that enable connections between 5G UEs and the core network. The 5G core network serves as the back end of the RAN, providing services including connectivity and mobility management, authentication and authorization, and subscriber data handling. Unlike its monolithic predecessors, 5GC separates its functionalities into several components called Network Functions (NFs). This service-based architecture (SBA) provides more flexibility and scalability. However, it has also increased the overall complexity due to the increased number of components and the intricate

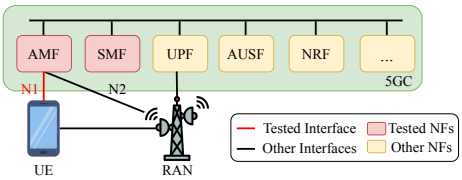

Figure 1: 5G system architecture

interactions between them. Among these NFs, control-plane traffic originating from a UE is primarily handled by two NFs: *Access and Mobility Management Function (AMF)* and *Session Management Function (SMF)*. The AMF directly communicates with UEs via the 5G Non-Access Stratum (NAS) protocol over a logical N1 interface [16]. Although NAS messages on the N1 interface are transmitted through the RAN, the RAN acts as a conduit, forwarding these messages to 5GC without processing or interpreting them. The AMF is responsible for key control-plane tasks, including authenticating and registering UEs, maintaining security contexts, and managing handovers. The SMF handles data-plane-related NAS traffic, responsible for establishing, maintaining, and releasing Protocol Data Unit (PDU) sessions, enabling data (e.g., Internet) connectivity for UEs.

Automata learning. Automata learning aims to infer a system's abstract model based on either logged system traces (passive automata learning [49]) or actively sent queries (active automata learning [19]). Active automata learning typically consists of two alternating phases—*hypothesis construction* and *validation* [19, 40]. During hypothesis construction, the learner sends *membership queries* to the system under learning and generates a hypothesis model based on the responses. After that, to validate the hypothesis model, the learner sends *equivalence checking (EC)* queries to check if the generated hypothesis model is valid or not. EC queries revealing discrepancies between the hypothesis model and the system-under-learning are saved as counterexamples, which are utilized to further refine the hypothesis. However, because of the vast search space, finding counterexamples requires a large number of queries and reduces the overall efficiency of active automata learning.

3 Overview

3.1 Threat Model

We consider compromised UEs controlled by attackers that transmit arbitrary messages with malicious intent, potentially disrupting 5G services for benign users or even compromising the entire network's stability. For example, by gaining root access to a user device, an attacker could seize complete control of the device and interface it with a software-defined radio [9, 11] to craft and dispatch arbitrary uplink NAS messages to the 5G core network. The rogue UE equipped with operator-provided SIM has the full capability of sending perfectly encrypted, improper encrypted, or unencrypted NAS messages of arbitrary message type and content to the core network through a benign gNB. We assume that the rogue UE does not have any control over or access to gNBs interfacing with the core network. The primary objective of the attacker is to exploit its capability to craft an unexpected input sequence that can lead to the core network crashing or transiting to an unexpected state that leads to logical errors. Our threat model does not consider cases where malicious inputs are sent directly through the API of each NF, or the gNB directly initializes malicious or malformed traffic since both gNB and NFs are deployed and controlled by network operators and are unlikely to intentionally act maliciously.

Testing scope. Conceptually, CoreCrisis can be applied to test any protocol layers of cellular networks and any NFs in 5GC. In this work, we specifically focus on the 5G Non-Access Stratum (NAS) layer at the 5GC, testing the N1 interface (Figure 1). The NAS protocol messages are critical for security (e.g., authentication and security context establishment) and voice/data session management in cellular networks, with complex message structures and protocol states that make thorough exploration challenging. We consider both Subscription Permanent Identifier (SUPI)-based and Globally Unique Temporary Identifier (GUTI)-based registration modes. The AMF and SMF directly process NAS messages UEs send, though processing these messages often requires the AMF and SMF to interact with other NFs within the 5GC, creating dependencies that often extend the impact of these messages beyond AMF and SMF (Section 7).

3.2 Motivation for CoreCrisis

Testing commercial 5GC implementations poses unique challenges where access to the source code, compiled binaries, or the run-time environment is generally unavailable. Meanwhile, the 5G NAS protocol implementation in the 5G core is inherently stateful, requiring an understanding of the implementation's state machine to effectively navigate its behavior and identify deeply rooted vulnerabilities. In the absence of direct access to the implementation, one may resort to a manually constructed reference state machine [37, 38] from the technical specifications to approximate the expected state space. However, relying solely on a single reference state machine is insufficient for guiding the testing of all implementations, primarily for two reasons. *First,* the granularity of state implementations can vary between the actual implementations and the specifications. For instance, a state defined in the specification might be further divided into multiple substates in a practical implementation. As an example, Figure 3 demonstrates the state space of a 5GC implementation, OpenAirInterface [9], in which states `a0`, `a3`, and `a4` all represent a single *deregistered state* in the 5G specification [14]. *Second,* implementations may not strictly adhere to the specifications, resulting in discrepancies that a single reference model can-

not capture. For example, state a7 in Figure 3 represents a vulnerable state that deviates from technical specifications (P3 in Table 2). Therefore, to effectively guide the testing process, the reference state machine must capture variations in different implementations instead of utilizing a uniform state machine. This signifies that accurate state inference based on the implementation is critical.

Two primary approaches are commonly used for inferring state machine from implementations: *run-time inference* of states [52] and *automata-learning* [24, 39, 60]. Run-time state inferences often suffer from inaccuracies and incompleteness, primarily due to their reliance on ad-hoc state learning techniques. The poor quality of state guidance in these approaches results in input generation that lacks an understanding of the protocol's context and semantics. Additionally, these methods are largely limited to detecting memory errors and are incapable of automatically identifying logical errors in the protocol's behavior.

On the other hand, approaches that use black-box automata learning to construct state machines also face significant limitations when applied to testing 5GC. (1) These approaches require a predefined static set of symbols for state machine construction. Consequently, the learning phase must include both *benign* symbols (i.e., valid protocol messages) and *malicious* symbols (e.g., invalid messages, such as those containing an invalid message authentication code) to simultaneously capture regular protocol behavior and enable negative testing [50]. (2) Since there can be an arbitrarily large number of malicious symbols (e.g., each bit mutation of a message creates a new symbol), increasing the symbol set to address the first limitation results in significant growth in the time and resources required for automata learning. This often extends the learning process to weeks, imposing strict upper limits on the number of symbols to keep learning feasible within reasonable time and resource constraints. (3) Due to the static nature of automata learning algorithms, adding new symbols requires existing approaches to re-learn the entire state machine from scratch. (4) After the state machine is constructed, these approaches employ differential testings to compare the learned state machines from multiple implementations to find vulnerabilities. However, the effectiveness of such methods heavily depends on the availability of multiple implementations for comparison. While this approach has shown success in UE testing, where state machines from dozens of commercial UEs can be compared [39, 60], it is significantly less effective in the 5GC context. With only three publicly accessible 5GC implementations [5, 8, 9], the ability to identify inconsistencies or anomalies through differential testing is greatly limited.

3.3 Challenges of CoreCrisis

To facilitate effective testing and address these complexities, we develop CoreCrisis, a context-aware, threat-driven, learn-and-fuzz-based black-box testing framework. CoreCrisis is designed to navigate the complex and stateful nature of 5GC implementations by leveraging an evolving state model that represents a behavioral abstraction of the implementation under test. Its workflow is divided into two stages: an *initial FSM learning* stage, which constructs a basic skeleton FSM of the implementation using only benign symbols, and a subsequent *dynamic testing* stage, where CoreCrisis utilizes the extracted state machine to guide further fuzzing and exploration of the core network's internal state space. Despite its systematic approach, applying CoreCrisis to 5GC implementations presents unique challenges. We next present a few major challenges that CoreCrisis addresses.

C_1 **Scalability and efficiency issues of initial automata learning.** Following the black-box design of CoreCrisis, it learns the *initial FSM* (also denoted as *skeleton FSM*) of the 5GC implementation through black-box automata learning [19, 40]. The automata learning uses the input/output messages sent/received from a user device to the core network to infer the skeleton FSM. However, applying vanilla automata learning, similar to prior works [24, 39], to construct a monolithic FSM that encompasses all NFs, even if it represents only the regular behavior of the 5G core network, leads to significant scalability issues. Managing the large state space in a single FSM also poses challenges in scheduling states for targeted testing and difficulties in efficiently triaging once vulnerabilities are detected. To make matters worse, existing automata learning approaches suffer from the slow discovery of counterexamples during the equivalence checking (EC) phase [24, 39]. This phase randomly generates numerous queries to the target system before eventually finding a counterexample that can be used to refine the hypothesized FSM. The untargeted generation and testing of these queries, especially for remote targets like 5GC systems, lead to significant delays in discovering counterexamples, resulting in inefficient FSM learning that could take days to months to complete [39].

C_2 **Sequence-level mutations.** For existing stateful testing of protocol implementations [52, 57], each sequence of messages is perturbed similarly to message-level mutations by blindly inserting, removing, or shuffling messages within the sequence or by concatenating sequences without regard to the logical flow or context/state requirements. This is due to the inherent difficulty of inferring valid or meaningful sequences without detailed guidance from a state machine or domain knowledge. This method often leads to the generation of uninteresting or ineffective test sequences. For example, it generates redundant cycles within the state machine instead of exploring new states. It can also produce non-compliant sequences that result in early rejection in the first few messages. Additionally, such naive sequence-level mutations also result in unbalanced testing across states, where some states are explored more often by inputs while others are neglected.

C_3 **Acquiring fuzzing feedback & oracles in black-box**

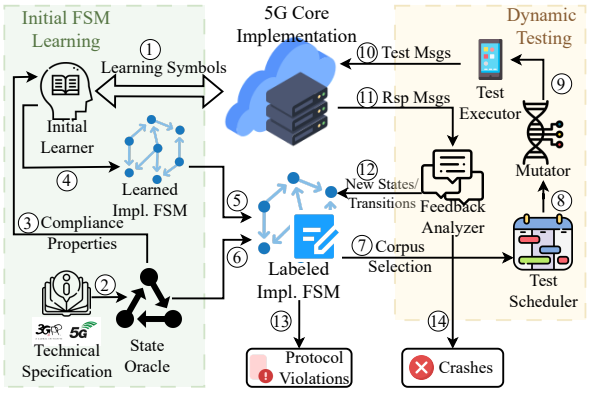

Figure 2: CoreCrisis architecture

An overview of CoreCrisis's approach is presented in Section 3.4

settings. Obtaining detailed feedback is essential for uncovering new input coverage, e.g., identifying new states and transitions discovered by test inputs, and for guiding input generation, e.g., retaining these inputs for further mutation to progressively explore the 5GC. However, CoreCrisis's black-box approach restricts its access to the source code or the machines hosting the 5GC, limiting its use of white- or gray-box feedback methods [20,52], such as code coverage or dynamic tracing. Since CoreCrisis can only observe over-the-air (OTA) input and output messages, detecting crashes or logical deviations in the core network after each test is also challenging.

3.4 High-Level Approach

Figure 2 presents the workflow of CoreCrisis, including an *initial learning* phase and the subsequent *dynamic testing* phase. To address the scalability and to significantly reduce the time and resources needed to learn this initial FSM (challenge $\boxed{C_1}$), CoreCrisis incorporates two novel techniques, *divide-and-conquer* and *property-driven equivalence checking* in its initial FSM learning phase (① in Figure 2). Divide-and-conquer allows CoreCrisis to learn the FSMs of each NF within the core network individually, thereby mitigating the significant increase in learning time that typically occurs when constructing a large monolithic FSM. On the other hand, property-driven equivalence checking bootstraps the learning process, using the intuition that the implementation FSM should follow the key transitions defined in the technical specifications. Therefore, we first extract the expected behavior requirements for each key state as described in the technical specifications [13] (②). These expected behaviors are formulated as *compliance properties* that help generate queries for equivalence checking, which can more effectively refine the learned FSM (③). Initial learning is detailed in Section 4.1.

In the second stage, CoreCrisis leverages the extracted state machine (④ ⑤) and the state oracle (⑥) to guide the exploration of the core network's internal state space by generating targeted malicious symbols to uncover flaws in the deep protocol states of the implementation. To address the challenge of sequence-level mutations $\boxed{C_2}$, CoreCrisis employs a context-aware fuzzing and state space exploration strategy (detailed in Section 4.2). This approach prioritizes less-explored or potentially vulnerable states and constructs state-specific test messages to maximize coverage (⑦ ⑧ ⑨). To address the challenge of acquiring feedback and oracles in black-box settings $\boxed{C_3}$, CoreCrisis utilizes the observable black-box feedback, specifically response (i.e., output) messages of the tested inputs. If CoreCrisis detects previously unseen responses, signifying the discovery of new states or transitions, it dynamically refines the guiding FSM to incorporate these updates (⑩ ⑪ ⑫). The refined FSM is then used to better guide the generation of subsequent test inputs (Section 4.3). During this iterative process, CoreCrisis automatically identifies *weird states* (⑬) by identifying states that deviate from expected behaviors defined in the technical specifications and may lead to exploitable logical vulnerabilities, and detects crashes using side-channel-based probing messages (⑭) (Section 4.4). Through this iterative mechanism, test case generation and FSM refinement continually inform and enhance each other, enabling a systematic and efficient traversal of the implementation's entire state space.

The two-step approach of CoreCrisis offers four key advantages. (1) It allows for the exploration of a large number of benign symbols during the initial learning stage, enabling a comprehensive examination of benign protocol behavior. (2) Since the total number of benign symbols is still significantly smaller than the combined set of benign and malicious symbols used in traditional automata learning approaches, our learning phase is faster. Additionally, we incorporate *divide-and-conquer* and *property-driven equivalence checking* to drastically improve the efficiency and scalability of this learning phase. (3) The dynamic testing stage can easily extend the tested symbols by introducing different malicious inputs, uncovering more vulnerabilities. This stage imposes no upper limit on the number of tested symbols as our mutation can generate unlimited test symbols. (4) If a test symbol reveals new states or transitions not identified during initial learning, our approach dynamically updates the FSM during testing without re-learning the entire state machine. This updated FSM then further guides fuzz testing to uncover additional vulnerabilities in the newly discovered states.

4 Detailed Design of CoreCrisis

4.1 Initial FSM Learning

Before dynamic testing, CoreCrisis learns a skeleton state machine using benign symbols (i.e., positive inputs), which serves as a foundation to guide the fuzz testing process. However, off-the-shelf state learning algorithms face scalability issues, resulting in the learning phase not completing when

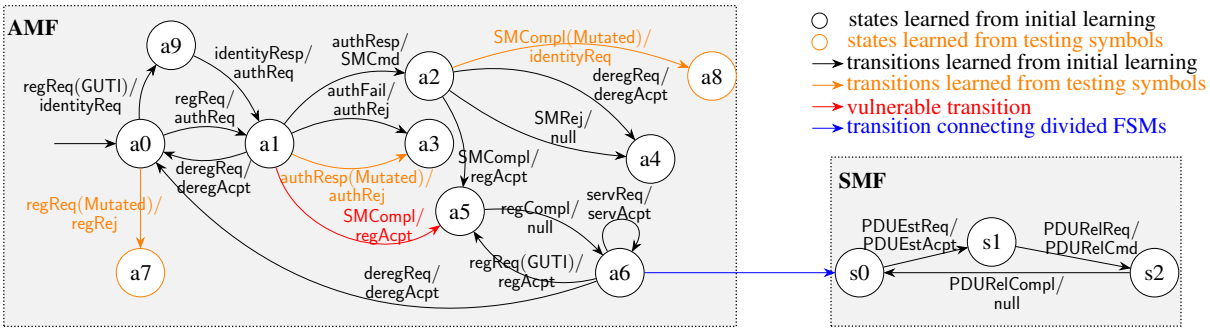

Figure 3: Simplified representation of the learned state machine of a 5GC implementation [9]
Black states and transitions in the diagram represent those learned during the initial FSM learning phase using only benign symbols (§4.1). CoreCrisis references the learned FSM to guide the dynamic testing of 5GC by focusing on underexplored states (§4.2). Newly identified transitions or states (states and transitions in orange and red) are used to refine the FSM during testing (§4.3).

directly applied to 5GC implementations. To address this challenge, CoreCrisis incorporates two novel techniques, *divide-and-conquer* and *property-driven equivalence checking* in its initial learning.

4.1.1 *Divide-and-conquer* FSM learning

CoreCrisis leverages domain-specific knowledge of NF dependencies as described by the technical specifications to take advantage of the SBA of 5GC to divide the monolithic FSM into multiple smaller FSMs, each representing a single NF. This division aligns naturally with the functional separation of NFs, making the learning process more intuitive and manageable. For instance, the technical specifications explicitly state the AMF is responsible for handling UE authentication and registration procedures, while the SMF manages session-related tasks after the UE has successfully registered with the core network [16]. Following this functional split, CoreCrisis divides the learning of the NAS procedure into two stages: first, learning the UE registration procedures, and then ensuring the UE has successfully registered (e.g., reaching state a6 of the AMF in Figure 3, which corresponds to the initial state s0 of the SMF) before proceeding to learn session management procedures. By independently learning FSMs for the AMF and SMF, CoreCrisis reduces the complexity of constructing a monolithic FSM into a more manageable task of linearly combining smaller FSMs. After learning, CoreCrisis sequentially merges the divided FSMs, applying the insight that session management procedures can only occur after successful registration (detailed in Section 4.2). The combined FSM is then used for further targeted testing.

4.1.2 *Property-driven* equivalence checking

CoreCrisis also incorporates domain knowledge of 5GC procedures to generate more targeted queries that are more likely to uncover meaningful counterexamples during equivalence checking (EC). The key insight is that implementations must adhere to the high-level transitions and the properties

PS	State Description	Expected Behaviors
D	Deregistered	Only RegistrationReq accepted
N	Registration-Initiated without authentication or security context	Only plaintext messages accepted ServiceRequest not accepted
A	Registration-Initiated with authentication without security context	Only plaintext and protected msgs SecurityModeComplete accepted ServiceRequest not accepted
S	Registration-Initiated with security context	Only protected messages accepted ServiceRequest not accepted
R	Registered	Only protected messages accepted

Table 1: Expected behaviors for AMF's protocol state (PS)

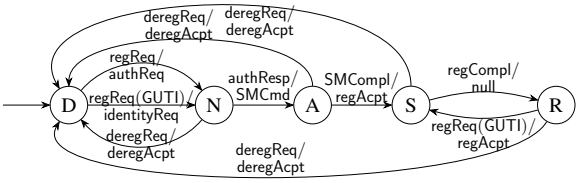

Figure 4: AMF's protocol FSM with key transitions

of the regular protocol behavior specified in technical documents, especially when testing benign messages during this initial learning phase. To leverage this insight, we first observe that the technical specifications [14, 15] define several key and high-level transitions, and each key state between these transitions has distinct requirements. One can extract these critical protocol states and their transitions to construct a protocol FSM representing expected key states defined in the specifications. Table 1 summarizes the key protocol states we manually extracted for AMF, along with the expected positive behavior for each state. Figure 4 illustrates the main transitions between these key protocol states. Table 6 and Figure 11 in Appendix C presents the key protocol states and transitions for SMF.

Using these key transitions, we derive *compliance properties* that guide the generation of EC queries. Testing these queries is more likely to uncover counterexamples used for hypothesized model refinement and increase the likelihood of faster discovery of discrepancies between the hypothesis and the actual implementation-under-testing. This is because

these queries target critical transitions that all 5GC implementations are expected to follow according to the specifications. For example, transitions D → N and N → A in Figure 4 can be translated into the property that states *authentication must be completed before a successful security mode command*. Formally, the property is expressed in regular expression as: $\boxed{\langle \mathsf{regReq/authReq} \rangle \cdot (\neg \langle \mathsf{deregReq/deregAcpt} \rangle)^* \cdot \langle \mathsf{authResp/SMCmd} \rangle}$ The regular expression format allows for the insertion of random new symbols in the generated sequences, enabling the detection of small, unanticipated transitions between the key transitions that are not specified in the technical documents but may exist in actual implementations. For example, this property might guide the generation of multiple EC query sequences, including $\langle \mathsf{regReq/authReq}, \mathsf{authResp/SMCmd} \rangle$, $\langle \mathsf{regReq/authReq}, \mathsf{servReq/null}, \mathsf{authResp/SMCmd} \rangle$, $\langle \mathsf{regReq/authReq}, \mathsf{identityResp/null}, \mathsf{authResp/SMCmd} \rangle$. By following this, the AMF's FSM presented in Figure 4 can be translated to 9 compliance properties, as shown in Table 8 in Appendix C.

Since the regular expression format of these properties can generate a large number of possible queries, we cap the number of generated EC queries during each learning iteration. If a counterexample is identified, the hypothesis is refined, and the learning proceeds to the next iteration. If a counterexample is not found within this limit, we heuristically terminate the learning process. This strategy of capping the number of queries performed may result in an incomplete but observationally correct initial FSM. This is an inherent limitation in black-box automata learning [19]. To address this, CoreCrisis actively updates this learned FSM by the behaviors observed during the dynamic testing phase, which discovers states that are missed at this stage (Section 4.3).

4.2 Test Sequences Generation

To address challenge $\boxed{\mathbf{C_2}}$, CoreCrisis implements a *two-phased mutation* strategy, which is designed to optimize the testing process by taking advantage of the guiding FSM to focus on specific states of interest. *Initially*, CoreCrisis selects a target state that it aims to test. From the various sequences of messages that reach this state, CoreCrisis selects an under-explored sequence and executes it to set the system to the desired state. This sequence may include both benign symbols, which are identified during the initial FSM learning stage, and mutated malicious symbols, which are introduced during the testing stage. CoreCrisis employs a power scheduling algorithm as discussed in Appendix A.

To illustrate, consider testing at state a7 in Figure 3, a state identified during the fuzzing stage. At this phase, we send a mutated regReq to first set the core network to state a7. *In the second phase*, we introduce a mutated symbol into the sequence to probe the program exploring new states and transitions from this set state. Following each test, CoreCrisis assesses whether the mutated message has led to the discovery of a new state/transition (Section 4.3), or a vulnerability (Section 4.4). To support this two-phased approach, CoreCrisis maintains a two-level corpus. It saves all unique accessing sequences to each state, which represent different paths from the initial state to the target state. At each state, it keeps a separate message corpus, which contains all messages that lead to unique transitions from the current state.

To support the separated FSM learned using divide-and-conquer (Section 4.1), CoreCrisis sequentially combines the divided FSMs in this dynamic testing phase by leveraging our knowledge of NF interdependencies. CoreCrisis first identifies connecting states between separated NFs as target states and then selects sequences that access these target states within each divided FSM. Subsequently, these accessing sequences are concatenated with the connecting transitions between the divided FSMs to form the complete accessing sequence. For example, in Figure 3, to test state s1 of SMF, CoreCrisis first identifies the connecting states between AMF and SMF, i.e., a6 and s0, respectively. CoreCrisis then constructs separate accessing sequences for the two divided FSMs: from a0 to a6 (e.g., a0 → a1 → a2 → a5 → a6) and from s0 to s1 (e.g., s0 → s1). These sequences are then combined to form a single sequence that accesses s1 from a0, specifically a0 → a1 → a2 → a5 → a6(s0) → s1.

4.2.1 Constructing syntactically and semantically valid test messages

Existing fuzzing approaches [48, 52, 57] typically employ byte-level mutations that do not understand the underlying message structure. Performing simple byte-level mutations on highly structured NAS messages results in messages that are almost always rejected by 5GC, as these mutations often violate the syntax (e.g., type, length, or structure) and semantics (e.g., value constraints, inter-field dependencies) requirements. Additionally, existing grammar-aware testing tools [3] often rely on the manual construction of message format. Extracting the complex grammar defined in natural language within specifications is challenging, as it requires thorough manual parsing of the specifications, which is both time-consuming and error-prone. Moreover, these messages require complex protocol-specific computations such as message authentication code (MAC), encoding, and encryption to construct a valid binary message. Documentations for these processes are scattered across various documents [1,14]. Even with correct grammar, the complexity of these procedures is challenging to implement for valid NAS message generation.

To overcome these challenges, we harness existing open-source software UE implementations [34] that already contain predefined structure definitions for each NAS message type and function implementations to handle operations like MAC calculation, encryption, and encoding. The original software UE implements these codes to construct valid request messages to communicate with the core network. We

manually identify the relevant structure definitions and corresponding computation functions and transform the existing message construction code into message mutation functions. This grammar-aware capability enables CoreCrisis to generate messages that are either valid or intentionally malformed across three levels. (1) Semantically, such as omitting encryption where encryption is expected, or using an incorrect security key for computing the MAC; (2) Syntactically, by including incorrect field types or unexpected enum values; and (3) Structurally, by mutating the encoded byte message. These capabilities provide the foundation for CoreCrisis to conduct both positive testing (i.e., testing valid messages to observe their acceptance) and negative testing (i.e., testing invalid messages to ensure their rejection). CoreCrisis's detailed mutation strategy is presented in Appendix B.

4.3 Feedback & New State Learning

As discussed in challenge $\boxed{C_3}$ in Section 3.3, obtaining detailed feedback given CoreCrisis's black-box design is challenging but essential for guiding input generation and uncovering new system states and transitions during testing. To address this challenge, we draw on the insight that black-box automata learning algorithms can infer different states by analyzing the response messages received in reaction to test messages. CoreCrisis adopts this approach in its testing stage and enhances it by performing a detailed analysis of response messages to extract as much information as possible. This feedback guides targeted input generation, enabling CoreCrisis to discover and incorporate new states and transitions to its reference state machine.

4.3.1 Analysis and comparison of responses

To enhance the granularity of the feedback we collect from such a limited source (i.e., response to a message), we analyze not only the type and error code of the response message but also the contents of the response body, which may include more detailed error messages. Specifically, any response that differs from previously observed responses at the current testing state (different type, unique response code, or different message in the response body) is used as an indication of a new state or new transition exploration. The quality of the feedback depends on the implementation being tested. If an implementation offers detailed error messages, we can discern more fine-grained states, enhancing CoreCrisis's understanding of the system's behavior. For example, some implementations provide a 5GMM Cause error message when it receives unexpected messages. Based on different cause values, CoreCrisis can differentiate the states. While comparing with the previous responses, CoreCrisis ignores fields having different values in each response, such as sequence numbers and MACs, to prevent adding unnecessary states and transitions. CoreCrisis observes regular network traffic during initial learning and identifies such fields that are different for all messages.

4.3.2 Dynamic FSM refinement

CoreCrisis's initial automata learning process (Section 4.1) is designed to incorporate only benign symbols to ensure scalability. During testing, each mutated message represents a symbol not encountered in the initial learning and may elicit a new type of response from the core network. For example, consider the FSM in Figure 3. If a UE sends a RegistrationRequest message with an invalid user identity, the core network may respond with a RegistrationReject message ($a0 \rightarrow a7$) instead of the expected AuthenticationRequest that a benign request would generate ($a0 \rightarrow a1$). This divergence suggests a previously unknown behavior in the implementation. However, existing FSM learning algorithms [19, 40] require a static, predefined alphabet set, making them inflexible to the introduction of new symbols during or after the learning process. In contrast, CoreCrisis's mutation engine generates arbitrary messages, requiring the FSM to be dynamically refined with newly discovered symbols. This iterative refinement enables CoreCrisis to explore refined states and find vulnerabilities that can only be triggered through multiple mutations.

To refine the FSM dynamically, if CoreCrisis encounters a test input at state s_i producing a previously unseen response, instead of re-learning the entire automata, CoreCrisis first creates a hypothetical state s_h and a transition t from s_i to s_h. To verify if s_h is a new state or an existing state in the FSM, CoreCrisis tests all benign input symbols on s_h and observes the resulting responses. If all input and output pairs align with those of an existing state s_j in the FSM, CoreCrisis empirically confirms that s_h is the same state as s_j. It merges s_h with s_j and updates the FSM to include a new transition t from s_i to s_j. If no existing state matches the request/response pairs of s_h, CoreCrisis formally introduces s_h into the FSM as a new state, establishing the transition t from s_i to s_h. At this point, the newly found state s_h contains only one incoming transition edge t, and transitions from s_h to other states in the FSM will be learned in future iterations following the same approach. Once the FSM is updated, sequences leading to the new state or transition with the mutated message are stored and used in subsequent tests as accessing sequences to further explore this new state or transition.

4.3.3 Resetting core network

CoreCrisis's two-phased testing methodology requires each testing run to start from the initial state. However, restarting the entire core network or clearing all data significantly stalls the testing process (e.g., $\sim$ 10 seconds to restart free5gc [5]). CoreCrisis instead uses a domain-informed solution of assigning a new UE identity (i.e., SUPI) for each

new testing sequence. This allows it to simulate a fresh user context for each test, effectively resetting the network without time-consuming restarts. After each test, the connection is terminated, leaving only a minimal amount of residual context (e.g., ∼1KB in free5gc [5]), which does not interfere with the target's normal operation. While this method may lead to error propagation between user contexts, such issues would indicate security vulnerabilities that CoreCrisis aims to detect. However, no such issues were observed during our testing. To ensure that findings are not influenced by residual SUPI contexts, all results were validated on a freshly initialized core network and confirmed to be reproducible.

4.4 Testing Oracles

Defining effective test oracles to detect crashes and logical errors presents another challenge in CoreCrisis's black-box testing approach. To overcome this limitation, CoreCrisis leverages protocol side-channels to identify crashes and utilizes key protocol transitions to compare the states of the learned implementation FSM to detect logical deviations.

4.4.1 Protocol side-channel crash detection

First, CoreCrisis employs a probing strategy leveraging protocol side channels in the 5G NAS layer. CoreCrisis sends messages that, under normal conditions, guarantee a response according to the technical specifications [14, 15]. For instance, to check the status of AMF, CoreCrisis sends a RegistrationRequest with the last-used SUPI; for SMF, it sends a PDUSessionEstablishmentRequest. If CoreCrisis receives corresponding responses (AuthenticationRequest and PDUSessionEstablishmentAccept, respectively), it confirms that the corresponding NF is operational. If no response is observed, it infers the component is non-responsive, suggesting that it has crashed or entered a problematic state. This probing method is reliable because the technical specifications mandate responses to these specific messages, ensuring that any unresponsiveness is indicative of an issue. These messages are also straightforward to identify due to their clear and detailed documentation in the specifications.

4.4.2 Logical errors detection

5GC's expected behaviors specified in the technical documents vary at different protocol states. For example, the specification mandates that ServiceRequest messages should only be accepted after the UE has successfully completed registration with the core network [13]. Accepting such requests before a successful registration signifies a logical error. However, implementation FSMs often differ significantly due to variations in design and implementation details. Without access to source code or detailed implementation information, mapping an implementation FSM to the states defined in the

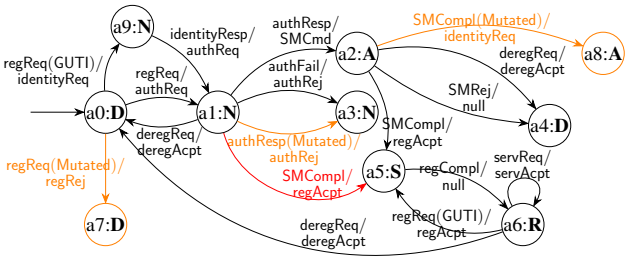

Figure 5: Labeled FSM of the AMF presented in Figure 3
Description of each key protocol state is presented in Table 1

specifications becomes challenging. To address this, CoreCrisis leverages the protocol FSM extracted from technical specifications (illustrated in Figure 4), which was previously used to guide equivalence checking during the initial implementation FSM learning (Section 4.1). This protocol FSM encapsulates the key high-level protocol states defined in the specifications, the expected behaviors of each state (summarized in Table 1), and the critical transitions between key states. CoreCrisis simplifies the task of mapping states in the learned FSM to their corresponding states in the protocol FSM by focusing on key protocol states and transitions. This approach is based on the intuition that key states can only be accessed through specific key transitions. By traversing and analyzing the transitions in the learned FSM, CoreCrisis labels each state with its corresponding key protocol state. Once labeled, CoreCrisis subsequently verifies the specified requirements for each labeled state, enabling systematic detection of logical errors.

Key state labeling. CoreCrisis automatically labels states in the learned FSM by analyzing the transitions (i.e., input/output pairs) associated with each state. Consider a sequence of transitions in the learned FSM, $i_1/o_1, i_2/o_2, i_3/o_3, \ldots$, CoreCrisis compares each transition with the key protocol transitions defined in the protocol FSM. If the input/output pair i_1/o_1 matches a transition in the protocol FSM's initial state p0 to another state p1, CoreCrisis consumes i_1/o_1 and labels the subsequent implementation states as corresponding to protocol state p1. The following input/output pairs are then compared against transitions originating from p1. If, for example, i_3/o_3 matches the transition from p1 to p2, CoreCrisis updates the current label to p2. After consuming all the symbols in the sequence leading up to each state in the learned FSM, each learned state is labeled with a corresponding protocol state, providing a clear mapping between the learned FSM and the protocol FSM. Figure 5 illustrates the labels for AMF's FSM in Figure 3, and Figure 12 in Appendix C shows the labels for SMF's FSM.

To illustrate, consider state a5 in Figure 3. The initially learned FSM, containing only states and transitions learned from benign symbols (states and transitions in black), has a path from a0 to a5: a0 → a1 → a2 → a5. State a0 is identified as the initial key state "Deregistered" (D). Along this path, a key transition regReq/authReq occurs between a0 and a1, re-

sulting in a1 being labeled as key state "Registration-initiated w/o authentication or security context" (N). The transition from a1 to a2 involves the key transition authResp/SMCmd, so a2 is labeled as key state "Registration-initiated w/ authentication w/o security context" (A). Finally, the transition from a2 to a5 includes the key transition SMCompl/regAcpt, leading to a5 being labeled as key state "Registered" (S).

Error identification. CORECRISIS identifies logical errors through a two-level checking process. ***First***, during the key state labeling phase, each path from the initial state to a state under consideration generates a label for that state. CORECRISIS analyzes the labels across all paths leading to each state to detect discrepancies. If a state is reachable from only one path, CORECRISIS initially assumes the label is correct. As new transitions are discovered, uncovering additional paths to the same state, CORECRISIS reevaluates the labels for consistency. CORECRISIS reports an error when a label discrepancy is found, and we then manually triage the issue to assign the correct label, ensuring the state machine accurately reflects the protocol's behavior for subsequent tests. For instance, consider again the example state a5 in Figure 3. During testing, CORECRISIS discovers a new transition from a1 to a5, indicated by the red arrow, revealing a new path from a0 to a5: a0 $\rightarrow$ a1 $\rightarrow$ a5. Here, a0 is labeled as key state D and a1 as N. Following this path, transition a1 $\rightarrow$ a5: SMCompl/regAcpt does not match either the condition or the action of any key transition outgoing from key state N, thus a5 retains the label N. However, path a0 $\rightarrow$ a1 $\rightarrow$ a2 $\rightarrow$ a5 labels a5 as key state S, resulting in a discrepancy. Manual analysis concludes that state a5 represents key state S, and the problematic path a0 $\rightarrow$ a1 $\rightarrow$ a5 leads to an authentication bypass vulnerability, presented in Section 6.1.1.

The ***second*** level of checks CORECRISIS performs is on top of the labeled FSM, where each state is annotated with its corresponding key protocol state. We preform protocol verification specific to each labeled protocol state, based on the expected behaviors for each key state extracted from technical documents, outlined in Table 1. For instance, after a 5G common-procedure, the protocol regulates that only protected messages should be processed by the core network. Scenarios where unprotected messages are accepted are identified as violations. This protocol-level check ensures compliance with the technical specifications, and any deviations are recorded as potential logical errors for manual validation. Deviation behaviors D1–D7 in Table 2 show the vulnerabilities CORECRISIS identifies from this check.

5 Implementation

We implement CORECRISIS in three components: *Message Adapter*, *State Inference Module*, and *Guided Testing Module*.

Message Adapter is developed utilizing UERANSIM [34], an open-source software simulator implementing two 5G components, UE and gNB. We modified the software UE with around 4,000 lines of code (LoC) to implement abstract symbol interpretation and input mutation.

State Inference Module is implemented on top of LearnLib [41] to perform the initial state learning (Section 4.1). We use TTT [40] as the learning algorithm and develop *propery-driven EC* as the equivalence-checking algorithm. *State Inference Module* generates queries from a predefined set of benign symbols. These symbols (i.e., input messages) and their corresponding field values are selected for initial learning from messages and values commonly used in standard 5GC procedures. The selection is based on UERANSIM's default configurations, presented in Table 7 in the Appendix. Each generated query is sent to *Message Adapter*, which constructs and forwards corresponding concrete messages to the core network. It then collects the response and converts it to output symbols.

Guided Testing Module is implemented in Python3 with around 1,200 LoC, including the main fuzzing loop, power scheduling, feedback processing, and protocol violations oracles. It orchestrates fuzzing by communicating with *Message Adapter* and reading state information from the learned FSM.

6 Evaluation

With the following research questions, we evaluate CORECRISIS's effectiveness when testing 5GC implementations.

- **RQ1.** How effective is CORECRISIS in finding protocol errors and crashes in 5GC implementations? (§6.1)
- **RQ2.** How does CORECRISIS perform compared to state-of-the-art fuzzing frameworks? (§6.2)
- **RQ3.** How effective are the novel techniques employed in CORECRISIS? (§6.3)

Experiment setup. We evaluated CORECRISIS on 1 commercial 5GC implementation, Amarisoft 5GC [2], which we only have access to the executable binaries, and 3 open-source implementations, Open5GS [8], free5GC [5], and OpenAirInterface 5G core network (OAI-CN-5G) [9].

6.1 Identified Vulnerabilities (RQ1)

To address **RQ1**, we present and analyze the logical deviations and crashes detected by CORECRISIS in Table 2. In the table, *Stateful* indicates issues that cannot be detected in the initial state without the guidance of the state machine, while *Refine* refers to issues that can only be discovered through FSM refinement (i.e., in dynamically learned states or by dynamically found transitions). In summary, CORECRISIS uncovers 3 exploitable protocol violations, **P1-P3**, 7 additional categories of protocol deviations from the technical specifications, shown as **D1-D7**, and 13 new unique crashes triggered by unexpected payloads that could lead to AMF or SMF crashes, listed as **C1-C13**. Below, we detail a few discovered logical

Flaw	Target	Vulnerability/Deviation	Comp.	Impact	CVE Number	New	Stateful	Refine
C1	Open5GS	assertion failure	AMF	DoS	CVE-2024-34476	Y	Y	N
C2	Open5GS	assertion failure	AMF	DoS	CVE-2024-34475	Y	Y	N
C3	Open5GS	assertion failure	SMF	DoS	CVE-2024-33232	Y	Y	N
C4	Open5GS	assertion failure	SMF	DoS	CVE-2024-33232	Y	Y	N
C5	Open5GS	assertion failure	SMF	DoS	CVE-2024-33232	Y	Y	N
C6	free5GC	array index out of bound	AMF	DoS	CVE-2024-22728	Y	N	N
C7	free5GC	array index out of bound	SMF	DoS	CVE-2024-31838	Y	Y	N
C8	OAI-CN-5G	null pointer dereference	AMF	DoS	CVE-2024-33236	Y	N	N
C9	OAI-CN-5G	null pointer dereference	AMF	DoS	CVE-2024-33236	Y	Y	N
C10	OAI-CN-5G	heap buffer overflow	AMF	DoS	CVE-2024-33236	Y	Y	N
C11	OAI-CN-5G	heap buffer overflow	AMF	DoS	CVE-2024-33236	Y	Y	N
C12	OAI-CN-5G	invalid memory access	AMF	DoS	CVE-2024-33236	Y	Y	N
C13	Amarisoft	segmentation fault	AMF	DoS	N/A	Y	N	N
P1	Open5GS	sink state	AMF	DoS	CVE-2024-33233	Y	N	N
P2	OAI-CN-5G	Incorrect message handling	AMF	A.B.	CVE-2024-33241	Y	Y	Y
P3	free5GC	Missing validation check of IMEI/IMEISV	AMF	I.S.	-	Y	Y	Y
D1	Open5GS, OAI-CN-5G	Incorrect transition	AMF	None	-	Y	Y	Y
D2	free5GC, OAI-CN-5G	Respond to invalid SHT (≥ 5)	AMF	None	-	Y	N	N
D3	Open5GS, OAI-CN-5G, Amarisoft	Respond to protected messages before SMCompl	AMF	None	-	Y	Y	Y
D4	Open5GS, OAI-CN-5G	Accept wrong SHT in SMCompl	AMF	None	-	Y	Y	N
D5	Open5GS, free5GC, OAI-CN-5G, Amarisoft	Accepts integrity-protected only messages when the selected algorithm is both integrity-protected and ciphered	AMF	None	-	Y	Y	N
D6	Open5GS, free5GC, OAI-CN-5G, Amarisoft	Respond to wrong SHT protected messages	AMF	None	-	Y	Y	N
D7	Open5GS, OAI-CN-5G	Respond to wrong SHT plaintext messages	AMF	None	-	Y	Y	N

Table 2: Implementation flaws found in our testing. C: Crashes; P: Protocol Violations; D: Logical Deviation
Impact: A.B: **A**uthentication **B**ypass, I.S.: **I**dentity **S**poofing

vulnerabilities and deviations in Section 6.1.1 and discuss the crashes CoreCrisis found in Section 6.1.2.

6.1.1 Identified Protocol Violations

Authentication bypass due to incorrect message handling (P2). TS 24.501 [14] mandates that the *registration* procedure is performed only after the core network has successfully authenticated the UE and finished the security mode control procedure, i.e., has derived encryption keys to protect further communications. CoreCrisis discovers a critical vulnerability in OAI-CN-5G that violates this requirement, illustrated in Figure 6. After sending an AuthenticationRequest message (① ②), the core network is expected to wait for AuthenticationResponse from the UE before initiating the security mode control procedure (③ ④). However, at this state, if an attacker sends a plaintext SecurityModeComplete message from the UE (⑤), OAI-CN-5G accepts this message and responds with a plaintext RegistrationAccept message (⑥). The attacker can then send back a plaintext RegistrationComplete message to complete the UE registration with the core network (⑦). This unauthorized access allows attackers to impersonate legitimate users, bypassing the authentication procedure, and potentially leading to billing fraud and privacy breaches (⑧ ⑨). If the attacker spams its unauthorized access and registers multiple illegitimate UEs, it may also degrade network performance and service quality for legitimate benign users. At an extreme, the attacker may operate a zombie network of malicious UEs registering to the core network and consuming all its resources, thereby causing DoS for legitimate users. The detection of this authentication bypass vulnerability relies on the dynamically learned transition from a1 to a5 (Figure 3). The plaintext SecurityModeComplete message is a mutated message that is not in the initial benign symbol set since this plaintext message should not be accepted in any state of the registration procedure. Without this dynamically learned transition, this vulnerability cannot be detected.

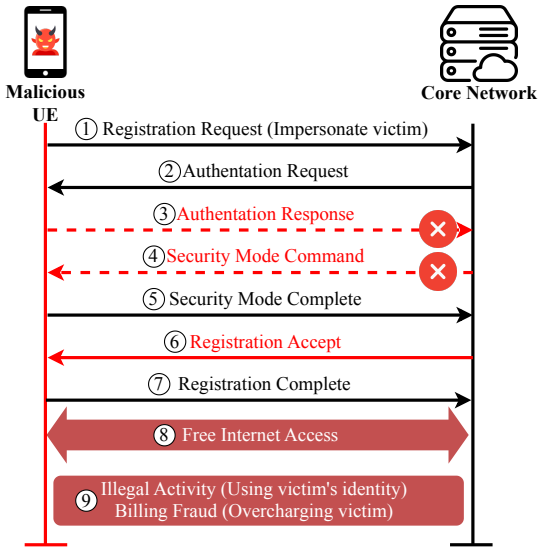

Figure 6: Illustration of authentication bypass attack (P2)

Identity spoofing due to missing validation check of IMEI/IMEISV (P3). During testing, CoreCrisis identified a vulnerability in free5GC where invalid IMEI/IMEISV

values are accepted during user registration. This issue was discovered in a dynamically learned state following state a2 in Figure 3. The fuzzer identified it after sending a mutated securityModeComplete message to the core network. Usually, a securityModeComplete message should include a registrationRequest containing the user's identity. However, in the mutated message, this field was removed, prompting the core network to send an identityRequest to request the user's identity again. This dynamically discovered state is represented in Figure 3 as state a8. Subsequent testing of this state revealed that when CoreCrisis sent a mutated message containing an invalid IMEI value, free5GC accepted the identifier and completed the registration process. IMEI/IMEISV is a unique identifier for mobile devices, providing manufacturer information and enabling user tracking. It has a fixed length and includes a checksum for validation. In this vulnerability, free5GC fails to validate the IMEI/IMEISV field properly. The impact of this oversight depends on how the network operator utilizes this identifier in its systems. Exploiting this vulnerability, an attacker could compromise any system that relies on IMEI values. For example, operators often maintain IMEI blacklists to block stolen devices from accessing their networks [7, 35]. Attackers could forge IMEIs to bypass these restrictions. Additionally, systems relying on user-provided IMEI values may be exposed to injection attacks if malicious input is supplied, potentially leading to the compromise of internal databases. Furthermore, operators may rely on IMEI values to enforce network access restrictions or grant privileged access. Exploiting this vulnerability, an attacker could inject arbitrary identifiers to gain unauthorized access to privileged systems, potentially leading to privilege escalation, data leakage, and network disruptions.

Fingerprinting (D1-D7) CoreCrisis also uncovers some behaviors that deviate from technical specifications but do not lead to directly exploitable vulnerabilities. The attacker may utilize these behaviors to fingerprint 5GC implementations to launch targeted attacks, such as sending specific payloads to trigger DoS on the core network (Section 6.1.2) or combine these deviations with other vulnerabilities to launch attacks. In addition, some deviations may also allow attackers to evade intrusion detection systems. Below we discuss several noticeable deviations.

• ***Incorrect state transition (D1).*** CoreCrisis identified an incorrect state transition issue in Open5GS. If the core network receives an IdentityResponse message without first receiving a RegistrationRequest message, it responds with an AuthenticationRequest message. According to TS 24.501 [14], only the RegistrationRequest message can initiate a state transition from the 5GMM-DEREGISTERED state to the 5GMM-COMMON-PROCEDURE-INITIATED state.

• ***Early acceptance of protected message (D3).*** We found Open5GS, OAI-CN-5G, and Amarisoft 5GC accept protected messages before receiving a SecurityModeComplete message, while free5GC only accepts protected messages after receiving this message from UE. After further investigation, we conclude that this issue is caused by the ambiguity of the specifications in TS 24.501 [14] clauses 4.4.2.5 and 5.4.2.1.

6.1.2 Identified Crashes (C1-C13)

A commercial 5GC may provide services to thousands of UEs. These vulnerabilities lead to the crashing of NFs, resulting in DoS attacks against the core network and disrupting the connectivity of all the user devices registered to the network. In mission-critical services such as defense, manufacturing, and emergency treatment, such disruptions can have severe and potentially life-threatening consequences. When these NFs crash, they require a complete restart to restore services, which may also require UEs to re-establish connectivity through new registration requests. This process leads to wasted resources and increased downtime. Additionally, an attacker could repeatedly exploit these vulnerabilities to cause recurring crashes, amplifying the impact.

Improper handling of Security Hearde Type (SHT). Among the thirteen crashes, five (**C1, C6, C8, C9, C10**) are related to inappropriate handling of Security Header Type (SHT). Open5GS, free5GC, and OAI-CN-5G assume an incoming NAS message with a non-zero security header type must have a message Authentication code (MAC). Under this assumption, these implementations perform byte operations on the message directly without input validation. For example, Figure 10 in the Appendix shows a crashing input that exploits vulnerability **C6** in Table 2. The crashing input, `0x7E025F74`, represents a plaintext SecurityModeReject message with a security header type 2 (integrity-protected and ciphered) containing only 4 octets. Assuming the message contains a MAC, the decoder attempts to access the plaintext NAS message starting from octet 8, without checking for the message length, resulting in a crash.

Malformed message field. Other crashes (**C2, C3, C4, C5, C7, C11, C12, C13**) are triggered by some malformed fields inside the message. For example, in Amarisoft 5GC, if the attacker sends a RegistrationRequest with an invalid SUCI that exceeds its maximum length, the decoder cannot decode the message, resulting in a segmentation fault that crashes the AMF (C13 in Table 2). In addition, we found that the message handlers for the PDUSessionModificationRequest message in Open5GS and free5GC's SMFs miss length checks for some fields, leading to four vulnerabilities (**C3, C4, C5, C7**).

6.2 Comparison with Existing Methods (RQ2)

6.2.1 Quantitative Comparison with Existing Fuzzers

We conduct a comparative analysis with state-of-the-art protocol testers. None of these tools can support 5GC testing or generating NAS messages off-the-shelf, and we summarize the modifications we made to extend their implementations.

Boofuzz [3] is a widely-used protocol fuzzer and a successor to Sulley [10]. To make Boofuzz suitable for testing 5GC, we take a similar approach as other research [29, 62] by defining and generating each message as a hexadecimal string.

Fuzzowski [6] is another successor to Sulley. Similar to Boofuzz, we generate hexadecimal strings as test input.

AFLNet [52] is a state-of-the-art stateful fuzz testing tool for protocol implementations. It utilizes the error/response code received from SUT to identify the program state and guide the mutation process. To support black-box 5GC testing, we extend AFLNet to enable communication with UERANSIM, and use message identities as a response code.

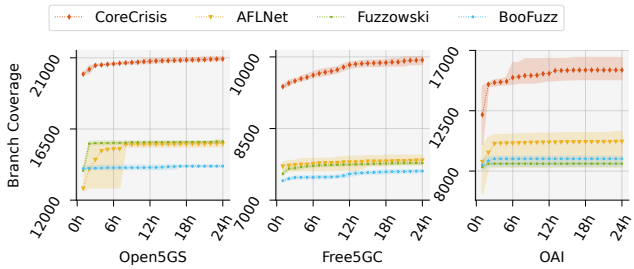

Figure 7: Discovered branches in comparative testing

Comparison results. We evaluated each tool on three open-source implementations for 24 hours, using the same initial corpus set. For fairness, all compared fuzzers use the same reset method (Section 4.3) to minimize the overhead associated with restarting the core network. Each experiment was repeated 10 times to ensure statistical significance [22]. Experiment results are detailed in Table 3 as averages and visualized in Figure 7, where the lines represent the average values and shaded regions indicate the minimum and maximum range.

We evaluate with the following metrics: (i) branch coverage, (ii) line/block coverage, (iii) number of generated corpora, (iv) protocol violations/crashes discovered, and (v) the testing speed. Coverage metrics differ by implementation due to the use of different languages and coverage tools. For mutation fuzzers, a corpus is saved if it explores new code regions, so the metric roughly represents the fuzzer's ability to generate meaningful inputs. For generation-based fuzzers (i.e., Boofuzz and Fuzzowski), the corpus metric reflects the total number of inputs tested. Fuzzowski completed testing all generated test cases within 24 hours, resulting in the same value across all implementations. In contrast, Boofuzz finished testing all test cases for Open5GS but did not complete testing for free5GC and OAI. In a black-box setup, AFLNet struggles to generate interesting inputs that can find new feedback, as shown by its low corpus count across all targets. CoreCrisis shows the slowest testing speed among the evaluated tools due to its intricate feedback processing, state inference, and state refinement mechanisms. For fairness in comparison, the reported speed metric is measured in messages per second across all tools. However, each stateful test sequence CoreCrisis generates consists of multiple messages (e.g., the sequence speed for Open5GS is only 0.218 sequences/second). Despite its slower speed, CoreCrisis achieves the highest code coverage and identifies the most vulnerabilities, demonstrating its superiority in uncovering critical issues. CoreCrisis also detects 3 logical vulnerabilities, while all other compared tools can only find crashes.

6.2.2 Feature Comparison with Other Methods

Many prior works testing UEs or core networks only support previous generations of mobile networks and cannot easily be adapted to 5GC. Therefore, we conducted additional experiments comparing the supported capabilities of CoreCrisis with existing works listed in Table 4. Among the works compared, DoLTEst [50], DIKEUE [39], and Chlosta et al. [24] can only detect protocol violations but not crashes. AFLNet [52] and BooFuzz [3] are state-of-the-art fuzzers designed to find crashes but cannot uncover protocol violations. Additionally, AFLNet's gray-box design requires program binaries for its instrumentation and feedback collection, while BooFuzz cannot understand the program states or perform state-guided exploration of the target.

6.3 Ablation Studies of CoreCrisis (RQ3)

6.3.1 Effectiveness of property-driven EC

To evaluate the impact of property-driven equivalence checking (introduced in Section 4.1), we conducted FSM extraction with and without this technique (with the divide-and-conquer technique in both cases) on two targets—free5GC [5] and Open5GS [8]. From the 9 compliance properties (shown in Table 8 in Appendix), we empirically limit the number of queries generated for equivalence checking (EC) to 400. In free5GC, these 400 queries learn 20 states, taking 1815 queries in total (1415 learning queries and 400 EC queries). Without this technique, to learn the same number of 20 states, 15104 queries are required (1497 learning and 13607 EC).

For Open5GS, 15 states are learned with these EC queries, taking 1,169 queries in total (769 learning and 400 EC). Without this technique, 7,753 queries are required to learn 15 states (877 learning and 7851 EC). In conclusion, property-driven equivalence checking reduces the number of total queries required to learn the same number of states by 87.98% and 84.92% in free5GC and Open5GS, respectively, and reduces the number of EC queries required by 97.06% and 94.91%, respectively.

6.3.2 Effectiveness of fuzzing components

We performed ablation studies to evaluate the contributions of our fuzzer's individual components to its overall effectiveness. Specifically, we compared CoreCrisis's performance

5GC Impl.	Open5GS						free5GC						OAI-CN-5G					
Fuzzer	branch	line	# corpus	PV	crash	speed	branch	block	# corpus	PV	crash	speed	branch	line	# corpus	PV	crash	speed
AFLNET	15582.8	32054.9	21.0	0.0	0.0	13.01	7838.2	14744.2	21.0	0.0	0.0	11.19	10208.8	15183.8	21.0	0.0	0.0	11.53
BooFuzz	14148.7	28734.6	174408.0	0.0	0.0	3.83	7610.0	14206.9	117086.0	0.0	1.0	1.36	8917.6	13299.2	174408.0	0.0	2.0	3.15
Fuzzowski	15707.4	32387.4	23626.0	0.0	0.0	2.15	7775.8	14567.1	23626.0	0.0	1.0	2.14	8560.8	12880.2	23626.0	0.0	1.0	1.78
CoreCrisis	20925.6	43559.5	461.7	1.0	2.0	1.08	9933.3	18738.7	594.3	1.0	1.0	1.01	15534.5	24330.7	597.4	1.0	3.3	0.87

Table 3: Comparative testing results. PV: Protocol Violation. Speed is in messages/second.

Approach	Stateful	Black-box Testing	Identify Protocol Violations	Identify Crashes	Dynamic Test Cases	Context-Aware Mutation
DoLTEst [50]	●	●	●	○	○	○
DIKEUE [39]	●	●	●	○	○	○
BooFuzz [3]	○	●	○	●	○	○
AFLNet [52]	●	○	○	●	●	○
Chlosta et al. [24]	●	●	●	○	○	○
5GBaseChecker [60]	●	●	●	○	○	○
CoreCrisis	●	●	●	●	●	●

Table 4: Comparison with existing testing approaches
●: Supported. ○: Unsupported.

under the following conditions: (i) without feedback collection, (ii) without FSM guidance, (iii) without using message grammar for test input generation, (iv) relying solely on the protocol FSM extracted from technical specifications (e.g., Figure 4), and (v) restarting the core network after each experiment instead of using a new identifier to avoid restarts. Each experiment was repeated 10 times, for 24 hours in each run, using the same initial corpus set.

Figure 8 shows the coverage growth over time for two targets, Open5GS and free5GC. The lines represent the average coverage, while the shaded regions indicate the minimum and maximum values. Figure 9 summarizes the number of discovered issues (both crashes and logical deviations) over time. In this figure, the lines represent the average time at which a specific number of issues were discovered, and the shaded regions indicate the minimum and maximum times for discovering those issues. The results of these ablation studies demonstrate that CoreCrisis consistently achieves the highest coverage and discovers more crashes and protocol deviations significantly faster than the other configurations. Our incorporation of stateful testing contributes the most to CoreCrisis's effectiveness, as performance without FSM or with only the specification FSM shows the weakest results.

Additionally, of the 23 identified issues presented in Table 2, 18 (78.3%) are stateful, meaning they cannot be detected in the initial state without the guidance of the state machine. Furthermore, 4 vulnerabilities are only identifiable in refined states or transitions, highlighting the importance of CoreCrisis's feedback and refinement mechanism. Besides refinement, logical error detection also relies on understanding the state space associated with mutated inputs, as the requirements for protocol behavior vary across different states (explained in Section 4.4). This makes it essential to determine whether mutated inputs uncover new states or transitions and incorporate these discoveries into the guiding state machine. Without this comprehensive feedback oracle and refinement mechanism, CoreCrisis would be unable to identify any logical errors in its dynamic testing (i.e., P1-P3 and D1-D7), as shown in Figure 9. Overall, CoreCrisis finds the most vulnerabilities, which is the most critical metric for security testing [22].

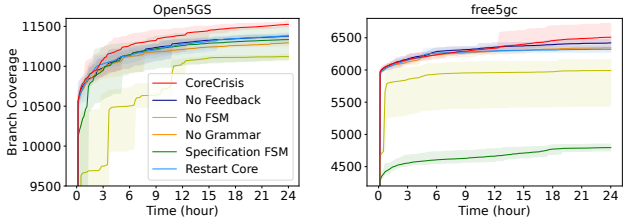

Figure 8: Ablation studies coverage evaluation

6.3.3 Crash detection accuracy

We evaluate the accuracy and effectiveness of CoreCrisis's crash probes (Section 4.4) using the AMF of Open5GS and free5GC, where ground truth is available through runtime logs. Over 10 runs, the logs reveal an average of 27.3 crashes in Open5GS (2 unique, C1 and C2 in Table 2) and 471.2 crashes in free5GC (C6), all of which are successfully detected by CoreCrisis. In contrast, an alternative approach could infer crashes based solely on the absence of response messages, but it has significant limitations. Some messages, such as regCompl, normally do not produce responses, resulting in missed detections of crashes (i.e., false negatives) if the message triggers a crash. Additionally, mutations may produce test inputs that are dropped without any responses, despite the NF has not crashed, leading to false positives. Our evaluation shows that this response-based approach yields average false negatives of 4.3 in Open5GS (C1) and 267.5 in free5GC (C6), and false positives of 8,433 in Open5GS

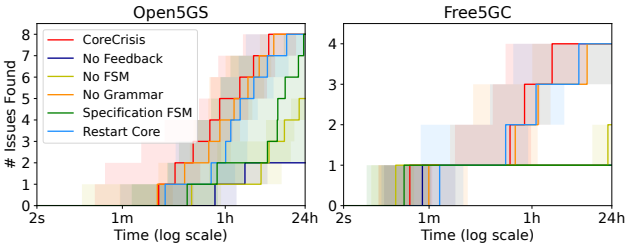

Figure 9: Unique crashes/logical deviations found over time

and 3,592.4 in free5GC per run. Furthermore, CoreCrisis demonstrates its effectiveness in black-box testing by successfully detecting a crash in a commercial Amarisoft 5GC deployment (C13).

7 Discussion

Multi-NF interactions. CoreCrisis maintains state machines only for AMF and SMF, leveraging domain knowledge for dividing the FSM of these NFs (Section 4.1). However, as CoreCrisis conducts black-box testing over the N1 interface (Section 3.1), it cannot directly observe or infer the internal division and organization of NFs within the 5GC. Handling UE requests involves interactions among multiple NFs beyond just AMF and SMF, but many of these internal interactions and states are not directly observable over N1. For example, AMF's retrieval of subscriber information from UDM is an internal operation abstracted in the transition a1 → a2/a3 in Figure 3. While these interactions are not directly observable, they may result in different behaviors over N1. For example, if a subscriber_info_not_found_in_UDM error occurs, it could lead to an observable AMF response to the UE, such as a registration_reject_with_cause_illegal_UE message. In such cases, the AMF FSM would be updated to reflect the observable outcome, even though the underlying transition originates from a different NF.

Manual effort. We manually extract the key protocol states (Table 1) from 3GPP specifications [14]. The construction of *Message Adapter* (§ 5) and message grammar (§ 4.2) require manual work. However, these efforts are one-time and can be reused for all subsequent tests across all implementations, which are completely automated. The triaging of crashes and verification of protocol violations are manual.

Correctness of learned FSM. The TTT automata learning algorithm [40] used by CoreCrisis guarantees the resulting FSM is observationally correct but not complete [19], which motivates CoreCrisis's dynamic refinement (§ 4.3). Evaluating the correctness and completeness of learned FSM beyond observational correctness is challenging, especially in a black-box setting, and previous works [24, 60] do not address it either. We leave it as future work.

Further enhancement of CoreCrisis. CoreCrisis focuses on NAS messages and AMF/SMF in the core network (Section 3.1). Extending CoreCrisis's methodology to test other messages (e.g., RRC messages) and other NFs (e.g., NRF and UDM) is an interesting future direction. Additionally, applying program analysis techniques could automate the construction of message grammar and mutation functions.

8 Related Works

Fuzzing. Among existing protocol fuzzers, SGFuzz [20], Nyx-Net [57], and Bleem [46] apply different approaches to infer protocol states and increase test efficiency. However, they cannot effectively generate valid test cases to test 5GC implementations, and their exploration cannot cover the complex 5GC state space. Learn&Fuzz [33] tests PDF parsers employing statistical machine learning techniques to generate inputs more likely to expose faults. Our method employs automata learning to develop an implementation state machine, which allows for a guided exploration of highly stateful cellular protocol implementations. Additionally, existing methods combining state learning and fuzzing [51] cannot dynamically update FSM during fuzzing to reflect new states and miss vulnerabilities caused by multiple mutated inputs (e.g., P3 in Table 2). In contrast, CoreCrisis employs a novel two-step approach that iteratively performs testing and FSM-refining.

Automata learning-based testing. Among passive automata learning approaches, Prospex [26] learns botnets' states by analyzing observed network traffic, while Hsu et al. [36] passively learn an automaton of message formats for fuzz testing. In contrast, active state machine learning has been applied to analyze various network protocols, including TLS [28], DTLS [31], TCP [30], IoT [59], OpenVPN [27], QUIC [53], SSH [32], and Bluetooth [42]. In the area of cellular networks, Hussain et al. [39] and Chlosta et al. [24] have applied automata learning to LTE protocols. Additionally, 5GBaseChecker [60] applies automata learning and differential testing to find logical vulnerabilities in commercial 5G UEs. However, 5GBaseChecker's FSM learning requires both positive and negative symbols, which results in a slow learning process. It can only detect vulnerabilities triggered by messages in this initially constructed symbol set. In contrast, CoreCrisis employs a scalable divide-and-conquer and property-driven equivalence-checking style learning approach, taking advantage of the service-based architecture and the properties of regular protocol behavior of 5GC. Also, CoreCrisis requires only a small benign symbol set for its initial FSM learning. The subsequent mutation and FSM refinement stage systematically explore the target to discover vulnerabilities triggered by messages that are not in the initial benign set. Additionally, CoreCrisis's compliance properties automatically identify logical vulnerabilities, whereas 5GBaseChecker's differential testing is ineffective with a small number of targets and cannot find vulnerabilities that exist in all implementations. To our knowledge, this work is the first to combine automata learning with mutation-based testing that iteratively refines the learned model.

Cellular network security. Most of the previous works in cellular network security [39, 43, 44, 47, 54–56, 58, 60–62] focus on attacks on the UE or RAN side. Existing efforts on core network security generally employ formal verification [17, 21, 37, 38], which rely on the protocol specifications, target specification errors, and cannot catch implementation flaws. Other efforts focusing on the security and privacy of the core network rely entirely on manual [25] or semi-automated stateless [45] methods. Only Chlosta et al. [24] adopt an au-

tomata learning approach for LTE core networks and uncovered several implementation flaws. However, their approach aims to identify logical bugs and cannot find crashing bugs.

9 Conclusion

In this work, we present CoreCrisis, a context-aware black-box testing framework for 5G core network implementations. CoreCrisis utilizes a dynamic two-step approach, first constructing an initial FSM using only benign symbols, and then refining this FSM through targeted dynamic testing. Through this approach, CoreCrisis dynamically generates and refines test inputs, ensuring comprehensive coverage of state spaces and effective identification of potential vulnerabilities. Our evaluation across various 5G core network implementations demonstrates that CoreCrisis excels in identifying both protocol violations and component crashes, significantly improving test coverage compared to existing approaches.

Acknowledgements

We thank the anonymous reviewers and the shepherd for their feedback and suggestions. We also thank the corresponding developers for cooperating with us during our responsible disclosure. This work has been supported by the NSF under grants 2145631, 2215017, and 2226447, the Defense Advanced Research Projects Agency (DARPA) under contract number D22AP00148, and the NSF and Office of the Under Secretary of Defense—Research and Engineering, ITE 2326898, as part of the NSF Convergence Accelerator Track G: Securely Operating Through 5G Infrastructure Program.

Ethics Considerations

All conducted experiments use either open-source 5GC implementations or commercial implementations with explicit permission. Our testing is performed exclusively in controlled environments using our own testbed, ensuring that no commercial networks are involved. We have reported all identified issues, as summarized in Table 2, to the respective developers. We are actively collaborating with them to triage the root causes and develop appropriate fixes. Currently, patches for 11 of the 16 identified vulnerabilities have already been merged into the latest releases.

Open Science

We made the source code of CoreCrisis and the modifications of all other fuzzers evaluated in Section 6.2 at the following link: https://doi.org/10.5281/zenodo.14735880, and also on GitHub [4]. This will ensure that our work is accessible and reusable by the broader research community.

References

[1] *5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 17.5.0 Release 17)*. [Online]. Available: http://www.3gpp.org/dynareport/33501.htm.

[2] Amarisoft. https://www.amarisoft.com/.

[3] boofuzz: Network protocol fuzzing for humans. https://github.com/jtpereyda/boofuzz.

[4] *CoreCrisis*. https://github.com/SyNSec-den/CoreCrisis.

[5] Free5gc. https://www.free5gc.org/.

[6] fuzzowski: the network protocol fuzzer that we will want to use. https://github.com/nccgroup/fuzzowski.

[7] Lost or stolen phone: What to do and how to protect your data.

[8] Open5gs. https://open5gs.org/.

[9] OpenAirInterface. https://openairinterface.org/.

[10] Openrce/sulley: A pure-python fully automated and unattended fuzzing framework. https://github.com/OpenRCE/sulley.

[11] srsran. https://www.srsran.com/.

[12] Aglfuzz: Automata-guided fuzzing for detecting logic errors in security protocol implementations. *Computers & Security*, page 103979, 2024.

[13] 3GPP. 5g security assurance specification (scas); access and mobility management function (amf). Technical Specification (TS) 33.512, 3rd Generation Partnership Project (3GPP), 2023. Version 18.0.0.

[14] 3GPP. Non-access-stratum (nas) protocol for 5g system (5gs); stage 3. Technical Specification (TS) 24.501, 3rd Generation Partnership Project (3GPP), 2023. Version 18.3.0.

[15] 3GPP. Procedures for the 5g system (5gs). Technical Specification (TS) 23.502, 3rd Generation Partnership Project (3GPP), 2023. Version 18.1.1.

[16] 3GPP. System architecture for the 5g system (5gs). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 2023. Version 18.2.0.

[17] 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.

[18] 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. *arXiv preprint arXiv:2310.04381*, 2023.

[19] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

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

[21] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1383–1396, 2018.

[22] Marcel Böhme, László Szekeres, and Jonathan Metzman. On the reliability of coverage-based fuzzer benchmarking. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1621–1633, New York, NY, USA, 2022. Association for Computing Machinery.

[23] 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.

[24] Merlin Chlosta, David Rupprecht, and Thorsten Holz. On the challenges of automata reconstruction in lte networks. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 164–174, 2021.

[25] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. Lte security disabled: misconfiguration in commercial networks. In *Proceedings of the 12th conference on security and privacy in wireless and mobile networks*, pages 261–266, 2019.

[26] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *2009 30th IEEE Symposium on Security and Privacy*, pages 110–125. IEEE, 2009.

[27] Lesly-Ann Daniel, Erik Poll, and Joeri de Ruiter. Inferring openvpn state machines using protocol state fuzzing. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 11–19. IEEE, 2018.

[28] Joeri De Ruiter and Erik Poll. Protocol state fuzzing of tls implementations. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 193–206, 2015.

[29] Xiaotao Feng, Ruoxi Sun, Xiaogang Zhu, Minhui Xue, Sheng Wen, Dongxi Liu, Surya Nepal, and Yang Xiang. Snipuzz: Black-box fuzzing of iot firmware via message snippet inference. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 337–350, 2021.

[30] Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. Combining model learning and model checking to analyze tcp implementations. In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II 28*, pages 454–471. Springer, 2016.

[31] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri De Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. Analysis of dtls implementations using protocol state fuzzing. In *29th USENIX Security Symposium, Online, August 12–14, 2020*, pages 2523–2540, 2020.

[32] Paul Fiterău-Broştean, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits Vaandrager, and Patrick Verleg. Model learning and model checking of ssh implementations. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, pages 142–151, 2017.

[33] Patrice Godefroid, Hila Peleg, and Rishabh Singh. Learn&fuzz: Machine learning for input fuzzing. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 50–59. IEEE, 2017.

[34] Ali Güngör. Aligungr/ueransim: Open source 5g ue and ran (gnodeb) implementation. https://github.com/aligungr/UERANSIM.

[35] Abida Haque, Varun Madathil, Bradley Reaves, and Alessandra Scafuro. Anonymous device authorization for cellular networks. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 25–36, New York, NY, USA, 2021. Association for Computing Machinery.

[36] Yating Hsu, Guoqiang Shu, and David Lee. A model-based approach to security flaw detection of network protocol implementations. In *2008 IEEE International Conference on Network Protocols*, pages 114–123. IEEE, 2008.

[37] 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.

[38] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 669–684, 2019.

[39] 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.

[40] Malte Isberner, Falk Howar, and Bernhard Steffen. The ttt algorithm: a redundancy-free approach to active automata learning. In *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings 5*, pages 307–322. Springer, 2014.

[41] Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib: a framework for active automata learning. In *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*, pages 487–495. Springer, 2015.

[42] Imtiaz Karim, Abdullah Al Ishtiaq, Syed Rafiul Hussain, and Elisa Bertino. Blediff: Scalable and property-agnostic noncompliance checking for ble implementations. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1082–1100. IEEE Computer Society, 2022.

[43] 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.

[44] Hongil Kim, Dongkwan Kim, Minhee Kwon, Hyungseok Han, Yeongjin Jang, Dongsu Han, Taesoo Kim, and Yongdae Kim. Breaking and fixing volte: Exploiting hidden data channels and mis-implementations. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 328–339, 2015.

[45] 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.

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

[47] 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.

[48] Roberto Natella. Stateafl: Greybox fuzzing for stateful network servers. *CoRR*, abs/2110.06253, 2021.

[49] José Oncina and Pedro Garcia. Identifying regular languages in polynomial time. In *Advances in structural and syntactic pattern recognition*, pages 99–108. World Scientific, 1992.

[50] 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.

[51] Andrea Pferscher and Bernhard K. Aichernig. Stateful black-box fuzzing of bluetooth devices using automata learning. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods*, pages 373–392, Cham, 2022. Springer International Publishing.

[52] 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.

[53] Abdullah Rasool, Greg Alpár, and Joeri de Ruiter. State machine inference of quic. *arXiv preprint arXiv:1903.04384*, 2019.

[54] David Rupprecht, Kai Jansen, and Christina Pöpper. Putting lte security functions to the test: A framework to evaluate implementation correctness. In *WOOT*, 2016.

[55] 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.

[56] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Call me maybe: Eavesdropping encrypted lte calls with revolte. In *USENIX Security Symposium*, pages 73–88, 2020.

[57] Sergej Schumilo, Cornelius Aschermann, Andrea Jemmett, Ali Abbasi, and Thorsten Holz. Nyx-net: network fuzzing with incremental snapshots. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, pages 166–180, New York, NY, USA, 2022. Association for Computing Machinery.

[58] Altaf Shaik, Ravishankar Borgaonkar, N Asokan, Valtteri Niemi, and Jean-Pierre Seifert. Practical attacks against privacy and availability in 4g/lte mobile communication systems. *arXiv preprint arXiv:1510.07563*, 2015.

[59] Martin Tappler, Bernhard K Aichernig, and Roderick Bloem. Model-based testing iot communication via active automata learning. In *2017 IEEE International conference on software testing, verification and validation (ICST)*, pages 276–287. IEEE, 2017.

[60] 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, Philadelphia, PA, August 2024. USENIX Association.

[61] 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, Santa Clara, CA, August 2019. USENIX Association.

[62] 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, Philadelphia, PA, August 2024. USENIX Association.

[63] Chuan Yu, Shuhui Chen, Ziling Wei, and Fei Wang. Secchecker: Inspecting the security implementation of 5g commercial off-the-shelf (cots) mobile devices. *Computers & Security*, 132:103361, 2023.

Appendix

A Power Schedule

CoreCrisis uses a two-layer power schedule algorithm to select the most suitable state and message for testing. Each corpus c is associated with a specific message and the state from which the message is sent (i.e., the sequences of messages needed to reach that state). The probability of selecting a corpus c for further mutation and testing is determined by its associated energy e_c:

$$P_c = \frac{e_c}{\sum_{c' \in C} e_{c'}}$$

This approach ensures that seeds with higher assigned energy are more likely to be selected in subsequent fuzzing rounds.

All corpora are initialized with a base energy e_0. If mutations of a corpus lead to the discovery of new states/transitions (i.e., finding a new corpus), its energy is significantly increased; otherwise, it is slightly decreased. All newly discovered corpora are initialized with e_0. The underlying intuition is that if CoreCrisis discovers a new response using a particular state and message, the corresponding corpus likely represents unexplored state space and should be prioritized.

To prevent starvation, CoreCrisis employs rules to adjust the energy of seeds dynamically. First, it tracks the visit count for each seed. If a seed in the corpus has been visited less frequently than the average visit count (i.e., is_rare(c) is true), additional energy is assigned to it:

$$e'_c = e_c + e_{rare} \times \text{is\_rare}(c)$$

Here, e_{rare} is computed as a fraction of the average energy across all seeds, ensuring fairness while prioritizing underexplored seeds:

$$e_{rare} = \alpha \times \frac{\sum_{c \in C} e_c}{|C|}$$

where α is a scaling factor that determines the extent of prioritization for rare seeds.

To mitigate extreme bias, CoreCrisis caps the adjusted energy of each seed to ten times the lowest original energy in the corpus:

$$e'_c = \min(e_c, 10 \times \min(E))$$

The adjusted energy values are recalculated before each seed selection to ensure up-to-date prioritization.

B Mutation Operators

The format of 5G NAS messages is defined in 3GPP TS 24.501 [14], with strict syntactical (i.e., structural) and semantic (i.e., value-based) constraints. Randomly generated inputs are highly likely to be rejected by the decoder due to these restrictions. To address this, CoreCrisis employs a grammar-aware mutation approach to systematically explore the input space.

Structure-aware mutation operator. Structure-aware mutation is applied only to syntactically-valid corpus messages. The mutation strategies incorporated in CoreCrisis are summarized in Table 5. Here, semantically-valid values refer to values defined as acceptable by the technical specifications (e.g., valid enum values and values within constrained ranges), while semantically-invalid values refer to disallowed or unexpected values. CoreCrisis randomly selects an Information Element (IE) within the corpus for mutation. Additionally, it generates unused fields to explore code paths that are not reachable with the initial corpus. For further exploration, CoreCrisis also mutates the value of the Security Header Type and encodes messages with and without a MAC and with or without encryption. Some unexpected values in certain message fields may cause incorrect state transitions or even trigger crashes, enabling CoreCrisis to detect potential vulnerabilities.

Type	Available Mutations
Security Header Type	Generate a random value Choose a semantically-valid value Choose a semantically-invalid value
Encryption	Randomly select between plaintext, integrity-protected, and both integrity-protected and ciphered message
Fixed-size IEs	Generate a random value with the same size Choose a semantically-valid value Choose a semantically-invalid value
Octetstring IEs	Generate a string with the same length Generate a string with a random length Generate an empty string Modify a random byte Delete a random byte Insert a random byte
Full NAS message	Modify a random byte Delete a random byte Insert a random byte

Table 5: Mutation strategies

Byte-level Mutation Operator. To explore flaws in the NAS message decoder, CORECRISIS also applies a probabilistic byte-level mutation operator to break syntactical constraints. Byte-level mutations are randomly chosen from three operations: (1) generate a random byte at a random location, (2) insert a byte at a random location, or (3) delete a byte at a random location. These operations result in syntactically invalid messages, allowing CORECRISIS to uncover deeper implementation flaws.

Both structure-aware and byte-level mutations are performed prior to any MAC calculation and/or ciphering operations. This ensures that the core network can correctly decode integrity-protected and ciphered test messages.

C Figures & Tables

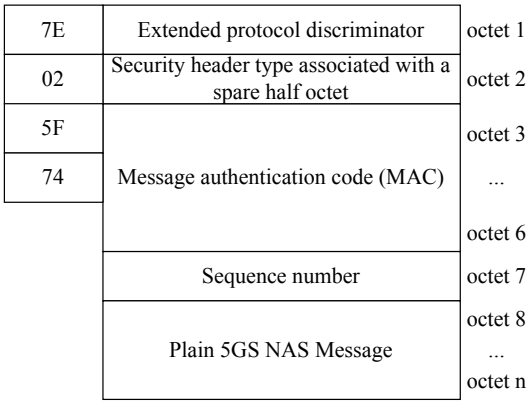

Figure 10: Example of a malformed message

PS	State Description	Expected Behaviors
NE	PDU session not established	Only PDUEstReq accepted
ES	PDU session established	PDURelReq and PDUModReq can be accepted
RI	PDU session release initiated	Only PDURelCompl accepted
MI	PDU session modification initiated	Only PDUModCompl accepted

Table 6: Expected behaviors for SMF's protocol state (PS)

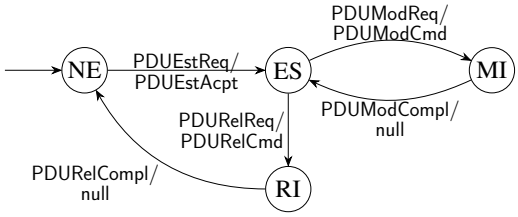

Figure 11: SMF's protocol FSM with key transitions

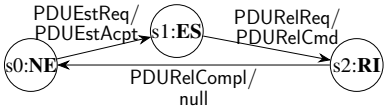

Figure 12: Labeled FSM of the SMF presented in Figure 3

Symbol Used	5G NAS Message Name	Direction
regReq	RegistrationRequest	Uplink
regReq(GUTI)	RegistrationRequest with GUTI	Uplink
regRej	RegistrationReject	Downlink
regAcpt	RegistrationAccept	Downlink
regCompl	RegistrationComplete	Uplink
idReq	IdentityRequest	Downlink
idResp	IdentityResponse	Uplink
authReq	AuthenticationRequest	Downlink
AuthRej	AuthenticationReject	Downlink
AuthResp	AuthenticationResponse	Uplink
AuthFail	AuthenticationFailure	Uplink
SMCmd	SecurityModeCommand	Downlink
SMCompl	SecurityModeComplete	Uplink
SMRej	SecurityModeReject	Uplink
CUCmd	ConfigurationUpdateCommand	Downlink
CUCompl	ConfigurationUpdateComplete	Uplink
SMRej	SecurityModeReject	Uplink
servReq	ServiceRequest	Uplink
servAcpt	ServiceAccept	Downlink
servRej	ServiceReject	Downlink
deregReq	DeregistrationRequest	Uplink/Downlink
deregAcpt	DeregistrationAccept	Uplink/Downlink
PDUEstReq	PDUSessionEstablishmentRequest	Uplink
PDUEstAcpt	PDUSessionEstablishmentAccept	Downlink
PDURelReq	PDUSessionReleaseRequest	Uplink
PDURelCmd	PDUSessionReleaseCommand	Downlink
PDURelCompl	PDUSessionReleaseComplete	Uplink
PDUModReq	PDUSessionModificationRequest	Uplink
PDUModCmd	PDUSessionModificationCommand	Downlink
PDUModCompl	PDUSessionModificationComplete	Uplink
null	No response from Core	N/A

Table 7: Symbols used in initial learning

Property ID	Description
A1	When the UE is deregistered, if the AMF receives a regReq message, it should send an authReq message.
A2	If the authentication is successful, the AMF should send a SMCmd message after it receives an authResp message.
A3	If the security context is established, the AMF should send a regAcpt message after it receives a SMCompl message.
A4	If the security context is established and the AMF has sent a regAcpt message, the AMF should move to registered state if it receives a regCompl message.
A5	After the AMF has initiated the authentication procedure, if it receives a deregReq request message, it should respond with a deregAcpt message and not respond to anything other than regReq.
A6	After authentication is successful and the AMF has initiated the security mode control procedure, if it receives a deregReq request message, it should respond with a deregAcpt message and not respond to anything other than regReq.
A7	After the security context has been established and the AMF has sent a regAcpt message if it receives a deregReq request message, it should respond with a deregAcpt message and not respond to anything other than regReq.
A8	After registration is complete, if it receives a deregReq request message, it should respond with a deregAcpt message and not respond to anything other than regReq.
A9	After registration is complete, if it receives a regReq(GUTI) request message, it should respond with a regAcpt message and not respond to anything other than regCompl.
S1	When no PDU session is present, if the SMF receives an PDUEstReq message, it should send an PDUEstAcpt message.
S2	When the PDU session is established, if the SMF receives an PDUModReq message, it should send an PDUModCmd message and not respond to anything other than PDUModCompl.
S3	When the PDU session is established, if the SMF receives an PDURelReq message, it should send an PDURelCmd message and not respond to anything other than PDURelCompl.

Table 8: List of properties used to generate property-driven equivalence queries