On the Challenges of Automata Reconstruction in LTE Networks

Merlin Chlosta
merlin.chlosta@rub.de
Ruhr University Bochum
Germany

David Rupprecht
david.rupprecht@rub.de
Ruhr University Bochum
Germany

Thorsten Holz
thorsten.holz@rub.de
Ruhr University Bochum
Germany

ABSTRACT

Mobile networks are a crucial part of our digital lives and require adequate security measures. The 4G and 5G network standards are complex and challenging to implement, which led to several implementation issues being discovered over the last years. Consequently, we aim to strengthen automation in testing and increase test coverage to spot issues and potential security vulnerabilities.

In this paper, we explore *active automata learning* for 4G/LTE protocol state machines. We focus on LTE's Mobility Management Entity (MME), the main core network element that handles all user registration and authentication through the NAS protocol. Based on automata learning, we automatically reconstruct the NAS protocol state machine to study implementation-specific artifacts and their security implications. We design and implement a reliable UE-to-MME interface for testing the MME from the perspective of a user device. This method allows testing of fully functional core networks without modification. Based on a prototype implementation, we test two open-source projects, one commercial MME implementation, and one MME in an operator's test network replicating the live LTE network. We expose several bugs, including crashes in three of the four implementations, potentially leading to network outages.

CCS CONCEPTS

• **Networks** → **Mobile and wireless security**; **Protocol correctness**.

KEYWORDS

LTE Security, MME, Automated Testing, Automata Reconstruction

ACM Reference Format:
Merlin Chlosta, David Rupprecht, and Thorsten Holz. 2021. On the Challenges of Automata Reconstruction in LTE Networks. In *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28-July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3448300.3469133

1 INTRODUCTION

4G and 5G mobile networks connect billions of users to the Internet. With an increasing number of applications and the integration of critical services, mobile networks themselves have become critical infrastructure. As a result, network equipment should be subject to enhanced scrutiny and extended testing. Determining whether an implementation complies with the appropriate standard is a challenging problem. We study this problem with a focus on LTE's user management and authentication protocols, since these protocols represent a core function of mobile networks.

When smartphones register at the mobile network, they are at the beginning of a "maze": a complex protocol state machine that controls under which conditions a user can connect to the network and which security aspects need to be considered [10]. The protocols' complexity arises from the main use-case of mobile networks: user mobility, which introduces many shortcuts into the state machine. The LTE standard describes the protocols' state machines textually without detailed reference and the handling of unexpected messages is not specified thoroughly. That requires human interpretation, which can lead to ambiguity and misunderstanding, and thereby to security issues [3]. Prior work demonstrates that LTE equipment is indeed prone to vulnerabilities due to implementation flaws. Kim et al. [13] present the semi-automated LTEFuzz and find spoofing and Denial-of-Service attacks. Meier et al. perform fuzzing on message parsers in baseband implementations [15]. Current testing approaches start with an investigation of the standard, which potentially leads to similar misunderstandings. Therefore, we aim to change that in our approach and start by automatically analyzing the implementation, deriving a model that is later checked for basic security properties – without much knowledge from the standard.

We explore the method of *active automata learning*, which works by mutating the *order* of otherwise valid messages, in contrast to fuzzing, where message *contents* are mutated. The feedback – responses from the network – allows the construction of a protocol state machine that is later checked against security requirements from the standard. Our approach integrates two valuable features: *i*) it automatically generates test cases, aiming at comprehensive state coverage, and *ii*) the reconstruction of a state graph yields a powerful visualization where logical errors become apparent. We apply our approach to the core network's *gatekeeper*, the MME: it is responsible for authentication, granting network access to legitimate users, and denying others. Phones *directly* communicate with the MME via the NAS protocol, relayed by the base station. Attackers can send arbitrary NAS messages over-the-air by using customizable basebands such as srsUE [7]. Consequently, any faults in the MME put the mobile network at risk and may cause network outages or unauthorized access.

In summary, we provide three key research contributions:

- We propose automata reconstruction for LTE's NAS protocol to uncover security vulnerabilities in LTE deployments. We design and implement a reliable system for simulating user input to the MME. Our approach utilizes feedback from multiple protocol layers, building on components from User Equipment and Base Stations. We overcome several challenges that

WiSec '21, June 28-July 2, 2021, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-8349-3/21/06…$15.00
https://doi.org/10.1145/3448300.3469133

otherwise arise from wireless testing protocols. We extend state machine learning to handle non-deterministic system behavior to enable real-world testing deployments.
- We implement a prototype of the proposed concept and run tests with the commercial *Amarisoft* network and two open-source implementations *srsEPC* and *Open5GS*. Furthermore, we evaluated an operator's test network to demonstrate that our approach can test a complex, real-world LTE deployment.
- We automate the state graph analysis and develop a rule-based model checking to find logical errors and expose potential security issues. We cause crashes in three networks, and show irregularities in all of them.

The upcoming 5G remains very similar on the here-discussed NAS protocol layer, making our approach directly applicable for 5G.

Responsible Disclosure. Our analysis reveals non-compliant behavior, especially in open-source implementations. We contacted the authors and found that the issues were independently fixed. We re-tested later releases and confirmed the fix. One non-critical issue in commercial software was disclosed through the operator's support channels and the issue was found resolved.

2 PRELIMINARIES

We briefly introduce active state learning, and review relevant parts of the LTE network with focus on user management procedures.

2.1 Active Automata Learning

Learning By Doing is human intuition: children try everything and observe what happens until they understand it. Similarly, we can examine protocol implementations by providing input and monitoring the output. *Automata learning* performs this learning process in an automated way for finite state machines (FSM, also known as state machines/automata) and is a well-suited method for communication protocols [5, 6, 22]. Formally, Mealy machines are described by the tuple $(I, O, Q, q_0, \delta, \lambda)$ with input symbols I, output symbols O, and the set of states Q (with the initial state $q_0 \in Q$). The transitional function $\delta : Q \times I \rightarrow Q$ defines the transition to the next state when an input is given to the machine. $\lambda : Q \times I \rightarrow O$ is the output function, mapping the output to a state transition.

Active automata learning *infers* the state machine from sequences of inputs sent to the system and the corresponding responses. We use LearnLib [12], an abstract state learning implementation that requires a custom interface to the system under learning. LearnLib sends abstract message sequences. These are translated to messages that make sense to the system under learning and its targeted protocol. Likewise, the replies are translated back to an abstract form that LearnLib understands.

LearnLib requires the state machine to be a *finite* and *deterministic* Mealy machine. Precisely, for a given state and input, there is only a single transition to another state. During this transition, the state machine replies with one output. We observed that a non-deterministic behavior of the core network is a significant challenge for the learning process. A greater input alphabet increases the number of required queries, however, the TTT algorithm attempts to reduce the number of queries. Ultimately, the runtime depends on the complexity of the to-be-inferred state machine.

Hypotheses and Refinement. State learning successively learns the Mealy machine in a two-phase process, which is repeated until the learning process is complete. The *learning* phase explores the responses to sequences of input symbols, resulting in a state machine *hypothesis*. The *verification* phase compares the hypothesis to the actual system with supplementary queries. If the system responses conflict with the hypothesis, e. g., by responding to one input with a different output symbol, we call it a counterexample. The following learning round uses this counterexample to refine the hypothesis further. If no counterexample is found, the learning process is finished. We use the standard TTT algorithm [11] for the learning phase and a random oracle for *verification* phase.

In contrast to fuzzing, Automata Reconstruction results in a high-level view of the protocol state machine. It does not aim for *code* coverage and software bugs e.g. during message parsing, but for the logic of the implemented protocol.

2.2 LTE Network

In mobile networks, the User Equipment (UE) connects wireless to base stations (eNodeB), which forward all user data to and from the core network. The core network performs management tasks such as registration, mobility management, and authorization for Internet access.

User Equipment (UE). The UE is the user's access terminal, in most cases a smartphone. A baseband modem inside the smartphone processes all LTE-specific functionality. Additionally, the SIM card stores the user identity (IMSI) and a shared key, which is used to establishing mutual authentication and derive communication keys.

eNodeB. On the network side, eNodeBs span the wireless *cells* that users connect to. eNodeBs independently perform all radio-related management through the RRC protocol without much interaction to the core network. They provide the transport layer to the core network for user's IP traffic and NAS protocol.

Core Network: Mobility Management Entity (MME). The operator-run *core network* is a server landscape that performs all management aspects of mobile networks. The MME is the central component managing user access, mutual authentication, and keeping track of a user's location. Most of these functions involve many other network nodes; however, the MME orchestrates them. With this central role, the MME is the central element for the network's access security and, therefore, this paper's main scope.

UE and MME communicate through the NAS protocol with the base station as a relay, involving interaction between the NAS layer and the underlying transport protocols RRC (radio-side) and S1AP (network-side). For example, the NAS instructs the radio layers to create a connection for user data if the authentication succeeds. If, however, the authentication fails, it terminates the radio connection. We will discuss the interaction of these protocols and their implications for state machine learning in Section 3.

2.2.1 *User Mobility.*

User mobility is central to mobile networks and a significant challenge: calls and data connections should work without interruption even when users travel. At the same time, power saving is essential to make smartphones work. Both requirements together are the design rationale for LTE procedures. LTE

separates the management tasks: base stations handle radio connections, and the core network handles logical user management. This separation allows suspending the radio connection for power saving while the logical registration remains active. Similarly, the core network might ask to renew the registration while the radio connection is already active. This separated design adds complexity to the protocols; there is no simple "on" and "off" state. Instead, several kinds of transitions between sub-states of the radio and logical connections are possible.

2.2.2 Procedures and States in LTE's NAS Protocol. The MME is directly exposed to user input via the NAS protocol (hence accessible to adversaries) and plays a crucial role regarding the network's security. Consequently, we focus our analysis on the implementation of the NAS protocol at the MME. In the following, we introduce *common*, *specific*, and *connection management* procedures to understand the subsequent analysis. As an example, Figure 1 shows the *Attach* procedure, combining several subprocedures.

Timers. It is important to note that most procedures use timers for triggering fine-grained timeout reactions. For example, the network may take several actions upon a timeout: (i) message retransmission, (ii) procedure cancellation, or (iii) fallback to another procedure. For our study, we neglect timers, hence, infer a simplified state machine without timers. We detail our assumptions in Section 3.2.

Common Procedures. These procedures are the building blocks for the high-level *specific* procedures, and they usually consist of a simple request-response message pair:

- *Identification* procedure: The network sends an *Identity Request* to the UE to ask for the identity; the UE transmits the identity in an *Identity Response*.
- *Authentication* to mutually prove the identity; the network sends an authentication challenge, the UE responds.
- *Security Mode* procedure: The network enables the encryption and integrity protection with a *Security Mode Command*. The UE acknowledges the decision (*Security Mode Complete*) and protects subsequent messages.
- Lastly, *EMM Information* exchanges parameters that have not been included in other messages.

Some messages require encryption and integrity protection, but not all:, e. g., *Identification* and *Authentication* do not require integrity protection, as they may need to run before the *Security Mode* procedure.

Specific Procedures. These procedures combine the building blocks and transition between the high-level states, e.g., establishing the connection when switching the phone on.

- The *Attach* procedure shown in Figure 1 runs when users initially connect to the network. The UE-originating *Attach Request* usually specifies the user identity, the type of connection (e.g., data access or emergency call), and the user device's capabilities. If the supplied identity is invalid, e.g., an outdated temporary identifier, the MME requests the permanent identity. Subsequently, MME and UE perform mutual authentication and activate encryption and integrity protection in the *Security Mode* procedure.

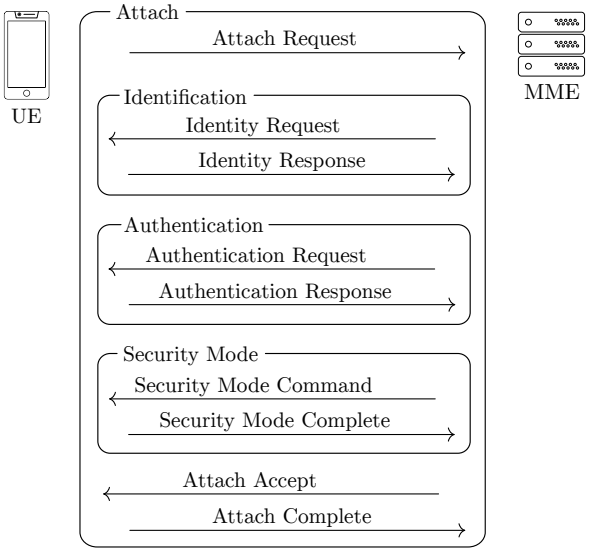

Figure 1: Attach Procedure in the NAS protocol with subprocedures. User device and MME communicate via a base station.

- The *Detach* procedure de-registers the user from the network, removing all active connections and contexts. Smartphones typically trigger this on switch-off or flight mode activation.

Connection Management Procedures. Most notable for LTE is that the UE may have an idle radio connection while being registered in the core network. To transit between the radio idle and active state, the following procedures are used:

- The network-initiated *paging* informs idle users to re-connect.
- The user-initiated *Service Request* is a fast short-cut alternative to the full *Attach*: if the *Service Request* passes the integrity check, the connection is immediately restored without re-authentication.

Handling of Unforeseen Messages. The LTE specification defines the procedures in a textual description. Hence there is room for ambiguity and misinterpretation. Further, performing these procedures must consider various timers, potential connection failures, and other side effects throughout the procedure's execution. Consequently, collisions between procedures can happen. For example, the UE may initiate the *Service Request* procedure right before receiving a (network-initiated) *Detach* request. Although the parts of specification try to clarify the handling of unknown and unforeseen protocol data, the actions are inadequately defined. The specification explicitly states that "If the network receives a message not compatible with the protocol state, the network actions are implementation dependent." [1, Section 7.4].

3 4G/LTE STATE LEARNING

Adapting the generic state-learning approach [22] for LTE brings some unique challenges. First, the MME has no direct interface for user input and requires a base station component (eNodeB) as an

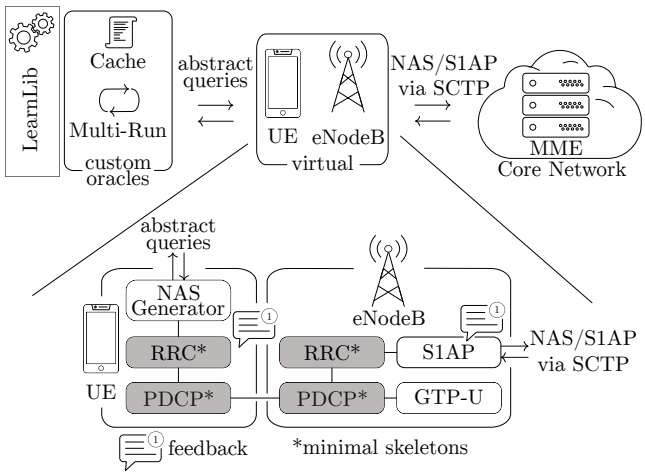

Figure 2: System architecture for MME automata learning.

intermediary node. Second, the additional components and associated protocols on different layers require caution due to possible side effects. We explain how we addressed these challenges and discuss potential caveats. More specifically, we introduce *i*) a reliable UE-to-MME interface with virtual UE and eNodeB component in Section 3.1 and *ii*) the LTE-specific considerations on state learning in Section 3.2. Figure 2 shows the designed system architecture, which we discuss in the following.

3.1 Simulating User Input to the MME

For enabling state machine learning on the NAS protocol level, we abstract the interaction between UE and MME. Therefore, we build an *instructable* UE-to-MME interface for generating and sending messages in arbitrary order. The lower half of Figure 2 shows the implementation. The UE-side NAS component receives instructions from LearnLib and returns the MME's answer message-by-message. eNodeB component transports the messages to the MME via the S1AP protocol. This approach is equivalent to sending the NAS message via the radio interface, even though we remove the radio layer for additional reliability, speed, and increased feedback. We used components of the srsRAN project [24] for our implementation.

Further, we add input actions for the other components and receive feedback from the lower protocol layers. All components share a common *response queue*: NAS messages, MME-issued S1AP commands, and error indicators are pushed to this event queue. The NAS generator class translates these events back to LearnLib output symbols or performs error handling. For generating output symbols, we prioritize NAS messages and include events from other layers (if available). If no response is available after a configurable timeout, we return a timeout symbol.

All protocols layers contain their own state machine and are inter-dependent on NAS. For example, the lower layers initialize on the first UE-sent message, and finish initialization with the first MME response. If that response never occurs (because the first message of that sequence did not make sense to the MME), this leaves the lower layers uninitialized. Our interface implementation abstracts these implications so that any input symbol can be sent at any time.

Attacker Model. We design our setup to test from a user's perspective. In general, all the input messages we send *to* the network can be sent over-the-air both for testing and for attacks. eNodeB-feedback would not be available, which hinders testing but not specific attacks. However, NAS-layer issues do not necessarily translate to exploitable vulnerabilities due to radio-layer security (see discussion).

3.2 Considerations for LTE State Learning

With the instructable UE-to-MME interface, we can send messages directly to the MME. However, in the complex environment of LTE core networks, automata reconstruction brings unique challenges.

3.2.1 Initial State and Reset Procedure. State learning requires starting from an initial, known state for each test case. More specifically, this applies to the MME as a whole and the UE's context maintained by the MME. The reset procedure attaches the UE, tests the Internet connection, and then detaches the UE. We consider this state as the initial state for all further tests, given that this is also the most realistic assumption in real core networks, where a UE context cannot be removed entirely. The reset procedure also acts as a health check to verify that the test setup is fully functional.

A major challenge is to perform a reliable state reset for the MME from *any* state the testing procedure may have caused. For example, we found open-source core networks frequently reach non-recoverable states (crashing, hangs, or non-responsive network components that the MME relies on). In those cases, we can not launch a simple detach procedure. Thus we monitor the reset and undertake additional actions if necessary. In some cases, we even need to restart the core network before further testing, including purging the UE's context in the packet gateway and other connected nodes. The core of our reset procedure works independent from any network, especially when the core network is stable.

3.2.2 Input and Output Alphabet. We support three types of input messages: *i*) NAS messages, *ii*) RRC/S1AP messages for emulating idle radio connections, and *iii*) a *Connection Check* symbol that triggers an ICMP ping through the LTE data bearers. We explicitly exclude timers and timeout behavior at all levels for simplicity and leave their analysis for future work in this area.

NAS message construction. The learning algorithm instructs the UE-to-MME interface to send and receive *abstract* symbols. These symbols must be translated to LTE-compatible messages. We support all messages that are part of the following NAS procedures: Attach, Detach, Authentication, Security Mode, Identification, ESM Information, and Service Request. The messages potentially lack the context and temporary values that usually come from the previous messages. Ensuring that the messages have a valid context, e. g., with valid security contexts, is the main challenge for the message construction. In particular, we must ensure that all messages are as meaningful as possible to trigger a state transition. If the current test run lacks a valid value, we reuse a value from the previous reset procedure. The reset procedure populates these temporary

values, ensuring that there is no inter-dependency between test cases. Mainly, we define the behavior for the following cases:

- *Authentication*: answers the last available challenge, re-playing the reset's challenge-response if necessary.
- *Encryption & Integrity Protection*: reuse keys from reset if current test case did not reach a key agreement.
- *Identification*: uses temporary identities in Service Request and prefers the permanent identity for Attach Requests.

Further, we parameterize the security protection of NAS messages to show if a message is accepted without a proper security check, and decouple the security header and payload protection:

- *Security Header*: the security header of a message indicates the following message protection (plain, integrity protected, integrity protected, and cipher). We allow the learner to *arbitrarily* set this header.
- *Payload Protection*: this parameter allows to skip the integrity protection, or both the integrity protection and encryption.

A NAS message input symbol is the combination of all parameters 'message type', 'security header', and 'payload security'. Similarly, NAS output symbols carry both the message type and the corresponding security header.

RRC/S1AP Multi-Layer Learning. Another challenge is that some MME state transitions come without a direct response or input symbol on the NAS layer. For example, the successful outcome of a Service Request is signaled by an RRC connection setup but not by a NAS message. Therefore, it requires using a multi-layer approach, including the NAS, RRC, S1AP, and IP layers, to enable state learning for LTE. We collect the following cross-layers feedback:

- *S1AP Bearer Setup (Output)*: when the MME instructs the eNodeB to set up a connection, e.g., due to a Service Request, this is only visible on S1AP.
- *S1AP Bearer Release (Output)*: similarly, the Bearer Release instructs the eNodeB to disconnect radio bearers to the UE, e.g., due to UE detach or unsuccessful authentication, but is not directly visible on NAS.
- *SCTP Connection Termination (Output)*: the transport-layer SCTP connection between eNodeB and MME was terminated; a strong indicator that the MME crashed.

Analog to the feedback, two functionalities are central to the NAS state machine but originate on other layers:

- *S1AP Bearer Release (Input)*: the eNodeB releases the UE and sends into the radio-idle mode due to inactivity. This command shall release the UE's data bearers.
- *Connection Check (Input)* a 'connection check' input symbol triggers sending an ICMP ping via the LTE data connection to test if the core network accepts and forwards user data. The ping response yields a 'connection successful' symbol.

3.3 Detecting & Handling Non-Determinism

LearnLib requires the to-be-learned state machine to be deterministic. However, complex systems like the LTE core network do not always act deterministically. Or rather: we cannot control all inputs to the system; thereby, the system may *seem* non-deterministic. We observed this aspect during error conditions, e. g., when input messages led to bad memory accesses that *sometimes* caused crashes. Re-enabling state learning for finite state machines with non-deterministic behavior prove as a major challenge. To solve this challenge, we modify LearnLib's learning process by adding a chain of *oracles* between LearnLib and the UE-to-MME interface. Generally speaking, we found that two kinds of unreliable behavior can appear, and we handle both cases differently: *i*) short-time transmission issues and *ii*) *real* non-deterministic behavior.

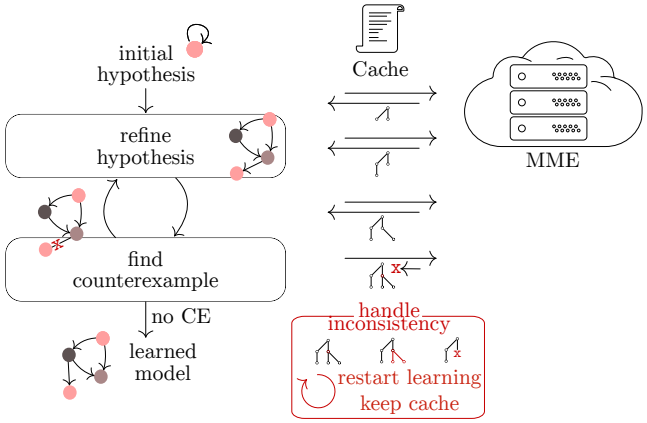

Figure 3: Inconsistencies are handled and replaced by a 'non-determinism' symbol in a multiple step process.

3.3.1 Transmission Issues. Re-transmissions, lost messages, and varying kinds of timeouts appear occasionally, without directly being caused by the NAS-level state. Thereby, these transmissions should not influence the learned state machine.

To handle transmission issues, we include a *probabilistic* oracle that executes every test case at least three times consecutively. If all results match, we assume it is free of transmissions errors. Otherwise, we repeat the execution until one answer occurs with a high probability. If no candidate appears reliably, a *non-determinism exception* is thrown that includes the two most likely answers.

3.3.2 Non-Determinism. Real non-determinism occurs when two identical input sequences result in contradicting responses. As the underlying learning algorithm assumes determinism, the learning process cannot infer a correct state machine. Nevertheless, we encountered this type of issue during the experiments, which we describe in the next section.

We might only realize that a message sequence yields unreliable output *after* one of the sequences already made it into the current hypothesis. However, LearnLib cannot *undo* observations. To address this shortcoming, we modified and extended the learning process to handle non-determinism, which is illustrated in Figure 3: We continuously record all executed test cases with an observation tree and check new responses against the observation tree for consistency. If any new observation disagrees with prior observation, we detect *Non-Determinism*. To introduce the *Non-Determinism* symbol into the learned state machine, we perform the following steps:

(1) Replacing states with *Non-Determinism* symbol. We remove all child nodes of the output symbols *after* the occurring non-determinism and replace them with a *Non-Determinism* symbol.

Table 1: Overview of the tested networks

Core Network	Section	Version	Inputs	States	Transitions
Amarisoft	5.3.1	2019-09-30	43	22	426
		2020-09-30	67	16	276
Open5Gs	5.3.2	2.0.3	30	33	697
		2.0.18	25	21	571
srsEPC	5.3.3	20.04	28	43	688
		20.10	67	9	179
commerical	5.3.4	-	17	10	170

(2) Learning the *Non-Determinism* symbol. In the previous step, we changed the considered state machine by introducing a new symbol. In consequence, the underlying learning is based on false premises. Therefore, we need to ensure that the underlying learning takes the newly introduced *Non-Determinism* symbol into account. We do this by restarting a clean learning algorithm and feed in the queries from the modified observation tree. This procedure ensures that we feed all knowledge into the learning algorithm, including the *Non-Determinism* symbol. An additional advantage of using the observation tree is the efficient re-loading after restarting the learning process.

We turn the *Non-Determinism* state into a sink: whenever the symbol occurs in a sequence, *any following input message automatically returns a Non-Determinism symbol* and does not reach the MME, as the answer would not be reliable anyway. This ensures the integrity and reliability of the underlying learning algorithm (TTT): towards LearnLib, the state machine is always deterministic as any non-deterministic behavior disappears behind the *Non-Determinism* state. The symbol still enables easy identification of potential error states: We can analyze the reasons for the non-determinism and perform manual retesting.

4 CHALLENGES OF THE EXPERIMENTS

We implemented an active automata learning system based on the preceding approach and conduct tests with four core networks: the open-source implementations *srsEPC* and *Open5GS*, the commercial *Amarisoft* EPC, and a live installation in an operator's test network. Table 1 lists versions and size of inferred state machines; we later perform a detailed analysis in Section 5. All core networks reflect realistic settings: their configuration requires encryption and integrity protection, and are otherwise unmodified. The networks also obey the standard interfaces; each network requires an integration phase and brings its own set of challenges for testing.

Some open-source core networks are hard to test. We often observed crashes in the HSS or gateways for those networks—leaving the network in an unstable state without any feedback, thereby impossible to handle for a reliable testing procedure. We decided to leave them untested and exclude them from further analysis. Likewise, the number of input symbols varies between core networks. The (very stable) Amarisoft was also tested with an additional *parameter* which effectively doubles the number of input symbols.

We found that both srsEPC and Open5GS were reliable enough for testing, but require a complete restart before each test case to ensure proper operation. The commercial networks required little to no adaption for our prototype implementation; these systems recovered from errors and prove reliable during our tests.

The execution time for test sequences varies. Naturally, the query length and the number of occurring timeouts influence the execution time. We set timeouts from 75 ms to 150 ms custom to each network, determined by the typical response time. However, the dominating factor is the time spent waiting for the core network to restart, as it was necessary for the tested open-source implementations. The runtime typically was below 4 hours. If crashes occur, this has a heavy impact on the runtime. This is why for some networks, we limit the number of tested input symbols: if we know that a certain *message* crashes the network, we do not test that message with all parameters.

Testing in an Operator's Test Network. We had the opportunity to perform several experiments in an operator's lab with a core network that replicates the public, live network. The deployed MME is a data center appliance, requires an operator-issued hardware SIM, and a slightly more complicated network setup to register an eNodeB. Besides that, it uses the same SCTP/S1AP/GTP-U interfaces and thereby accepts our eNodeB and UE components without issues.

However, the hardware SIM unexpectedly limited the generation of authentication responses to two per second. This posed a major bottleneck during our test, given the limited amount of time available. As a result, we only obtained a limited coverage and an incomplete state machine, at least compared to those implementations tested in our own lab. Further, we could not perform the 'connection check' due to the specific setup available to us and cannot determine if we had a working data connection in a given state. However, this does not influence the correctness of the rest of the state machine. The experiments showed that the approach is generally suitable to test commercial appliances, where source code or binary access is not an option, and the eNodeB interface is the only interface available for testing (see Section 5.3). Due to access restrictions caused by COVID-19 measures, we could not yet repeat and extend the tests in the operator's lab.

5 STATE MACHINE ANALYSIS

We perform our analysis based on the observed state machine. Implementation-internal states are not available for the analysis. Our analysis is solely based on the inferred state machine, and especially on its transitions (the request/response pairs).

As noted before, the standard lacks a well-defined state machine. Thus, assigning *names* to states and checking if the result matches the (underspecified) standard would contradict this analysis's purpose. Building a reference state machine puts us in the same position as a developer, and our interpretation would also be biased. Further, performing a manual analysis does not scale with the number of transitions. We choose *model checking* to check whether the given state machine meets specific security requirements.

Instead of comparing the inferred state machines to a reference, we distill the security-relevant properties and use model checking on that high-level abstraction. Figure 4 describes a two-step process that we apply on the inferred state machine. In the first step, we assign properties to the states themselves, based on the in- and outgoing transitions. Further, we inherit specific properties to following states if the connecting transitions meet all necessary

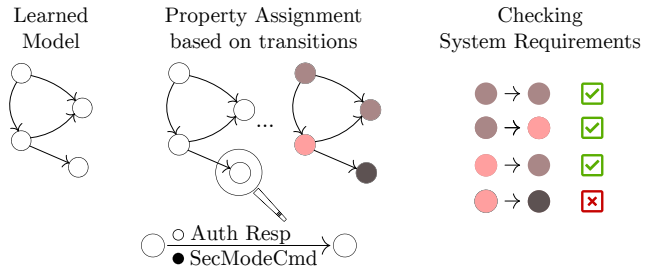

Figure 4: Model checking steps of the learned state machine.

criteria. This step does not conclude anything about security yet. In the second step, we check if the model meets a specific requirement. For this, we use the state graph's assigned state properties and direct transitions towards and from the state. As an example, Figure 5 shows part of an inferred state machine with properties already assigned.

5.1 Property Assignment

The property assignment is a building block for model checking. For example, it is central to our analysis whether a user has passed the Security Mode procedure, thereby we *mark* following states. We assign state properties based on the transitions. Some properties are propagated to the following states (directed) until a stop condition. In the following, we define the properties:

- **Authenticated State**: initially, all states that are reached through the Authentication/Security Mode procedure. We propagate the *authenticated* property through transitions where the incoming and outgoing NAS messages are integrity protected. It implies that the network successfully checks and decrypts the incoming message and acknowledges this with a protected message. We refer to those messages as *protected*. Essentially this means that all states after the *Security Mode* procedure are tagged as *authenticated* until a transition with missing protection occurs.
- **Unauthenticated State**: All states where no incoming transition indicates that a valid security context exists are considered *unauthenticated*, e. g., not reached through the Authentication/Security Mode procedure or exchanged a valid MAC in the header. By definition, we assume that radio-idle states are unauthenticated since we do not assume that authentication propagates through S1AP messages (leading to radio-idle).
- **Radio-idle State**: States where the state machine releases the radio connection, i.e., on S1AP Bearer Release.
- **Crashed State**: States in which the SCTP connection between eNodeB and MME is terminated, which indicates a crash of the MME.
- **Has-Internet State**: The *has-internet* property is based on a successful ICMP ping test.
- **Non-deterministic State**: States with an incoming *non-determinism* transition, which was created when the cache component detected such a condition.

5.2 Checking the Model: System requirements

We use the assigned state properties as blocks to formulate system requirements. We focus on six system requirements to analyze specific security-relevant aspects. Table 2 gives an overview of the inferred state machines. Property assignment and system requirements are implemented as Neo4J[16] queries. If some property is violated, the query returns the offending subgraph (see Figure 5 for example) that allows further manual investigation. We first introduce the system requirements and then present the results for the four core networks.

R1: Transitioning from Unauthenticated to Authenticated States. During attach, *only* the Authentication and following Security Mode procedure should cause a transition between unauthenticated to authenticated states. Further, the Service Request can cause such transitions because the radio-idle state is assumed to be unauthenticated.

R2: Acceptance of Unauthenticated Messages. The standard requires the MME to ignore *all* unprotected messages until the Security Mode procedure has finished – except for messages of the Attach and Identification procedure. Further, unprotected *reject* messages are allowed if the network cannot proceed with the UE registration.

R3: Network-Originating Unprotected Messages. After the security context is established through the Security Mode procedure, the MME must protect all outgoing messages. This means at least integrity protected or integrity protected and encrypted.

R4: Crashes. At no time crashes of the core network are allowed. Crashing an MME possibly renders the mobile network unavailable for all users in a given region.

R5: Non-Deterministic States. The state graph of the MME should not contain non-deterministic states. However, as noted in Section 3.3, we observed the non-deterministic behavior during our empirical experiments. Finding non-deterministic states is the starting point for manual investigation.

R6: Internet Access in Un-Authenticated States. No state should have both the *unauthenticated* and *has-internet* properties. Further, any path from *unauthenticated* states to *has-internet* states must either execute the Authentication and Security Mode procedure and pass only authenticated nodes from that point on, or use the Service Request procedure.

5.3 Core Networks

In the following, we describe the analysis results for each tested network. We first provide a general description of the results and then study the individual requirements. Table 2 summarizes the results.

5.3.1 Amarisoft. The Amarisoft core network is a commercially available system and is provided as executable binary [2]. The inferred state machine is derived from all available input symbols. We tested two versions *2019-09-30* and *2020-09-30*.

The inferred state machine for version *2019-09-30* has 22 states with a total of 43 distinct input symbols tested. Most notably, we found that Service Requests reliably crash the MME when sent

Table 2: Result Overview: ●yes, ○no, - missing data

Model Checking Queries	Amarisoft Sec. 5.3.1		Open5GS Sec. 5.3.2		srsEPC Sec. 5.3.3		Commercial Sec. 5.3.4
	v1	v2	v1	v2	v1	v2	
R1: Non-standard transitions *to* authenticated states	○	○	●	○	●	○	○
R2: Accepts unauthenticated messages after authentication	○	○	●	○	●	○	-
R3: Network-originating unprotected messages	○	○	○	○	○	○	●
R4: Network crashes on unexpected message	●	○	●	○	●	○	○
R5: Non-deterministic behavior	●	○	●	○	●	○	-
R6: Unauthenticated Internet access	○	○	●	○	○	○	-

without a valid MAC and we manually verified this behavior. This implies that an adversary can crash this MME, leading to a Denial-of-Service attack of the core network. Further, the Attach Request is always accepted and leads back to an *unauthenticated* state, while unprotected Detach Requests are not accepted. An attacker could thereby implicitly detach a target user by sending an Attach Request. Version *2020-09-30* fixes these issues: the overall state machine is more concise, without crashes or otherwise non-compliant behavior. In detail, we found the following behavior (applies to both versions if not stated otherwise):

- **R1:** Aside from the expected Authentication and Security Mode procedure, only Service Request leads from an unauthenticated to an authenticated state. The message must be properly integrity protected.
- **R2:** In authenticated states, the Amarisoft MME accepts only unprotected Attach Requests and the S1AP Bearer Release command: these lead back to unauthenticated states. Other messages are not accepted if unprotected.
- **R3:** Only Service Reject messages are sent by the network in plain, and only in states with no security context established.
- **R4:** Version *2019-09-30*: In radio-idle states, sending a Service Request with invalid message integrity should result in a Service Reject message. Instead, the Amarisoft MME reliably crashes. This enables an *unauthenticated crash* by sending an Service Reject message with invalid MAC from the initial state. Version *2020-09-30* does not crash during our tests.
- **R5:** Version *2019-09-30*: In some states, the Service Request does not *always* crashes the network, but produces a Service Reject. This ambiguity appears as non-deterministically in the state machine. Version *2020-09-30*: the network behaves deterministically and correct.
- **R6:** We do not see any unauthorized Internet access.

5.3.2 Open5GS. Open5GS (also called nextEPC) is an Open-Source 5G and EPC core network implementation written in C [17]. The learned model has 33 states and 991 transitions, which we learned with 30 input symbols. Those transitions often differ in small details leading to a new state allowing a fine-granular analysis. In the following, we analyze the state machine on the transition properties.

- **R1:** *Protected* Attach Requests cause the network to skip re-authentication and directly lead to authenticated states.
- **R2:** The network does accept unprotected messages, leading to the partly unauthenticated Internet access shown in Figure 5. In particular, Open5GS seems to skip the integrity check for Attach Complete and ESM Information Response.
- **R3:** All network-originating messages are protected, if possible, except for *Attach Reject* and Service Reject.
- **R4:** In many states, an unsolicited ESM Information Response and a subsequent random message causes a crash.
- **R5:** Our model does contain test cases that did not behave non-deterministically, however, no clear pattern emerges. The underlying issues may occur during the learning process and hypothesis creation, or within the UE-to-MME interface.
- **R6:** This criterion also catches the already-discussed unauthenticated Internet access in Figure 5.

We make two additional observations worth discussing as they indicate the core network's non-compliant behavior. First, the MME accepts a *Service Requests* even when a radio connection already exists. Usually, a Service Request shall reactivate the radio connection and only be sent by phone in the radio-idle state. Second, the MME accepts the Security Mode Complete with various security header types. Usually, the Security Mode Complete shall have the security header "integrity protected and ciphered with new EPS security context" [1, 5.4.3.3]. However, the MME accepts all security protected headers (expect plain) for the Security Mode Complete messages. Version 2.0.18 fixed all observed issues.

5.3.3 srsEPC. srsEPC is part of the srsRAN software suite that has the objective of providing an end-to-end LTE system [24]. The main focus of srsRAN is to develop a functional radio front-end rather than a fully functional core network. We tested versions *20.04* and *20.10*, showing fundamental changes in the state machine.

Version 20.04. We recognized two main patterns by the state machine of srsLTE in version *20.04*: *A*) Distinct states depend on the input *parameter* rather than only the message type. For example, a different security header type leads to a new state. Consequently, we find parallel paths during the initial attach procedure. *B*) The network *repeats* the previously sent message if it receives an unsolicited request instead of the expected answer. This behavior forces the state learning algorithm to derive new states for each unsolicited request and a response containing a repetition. Consequently, the state graph has more states than needed, reflected by the high numbers of 43 states and 1205 transitions with 28 input symbols. We filter repeating responses during the analysis. In the following, we analyze the state machine on the transition properties.

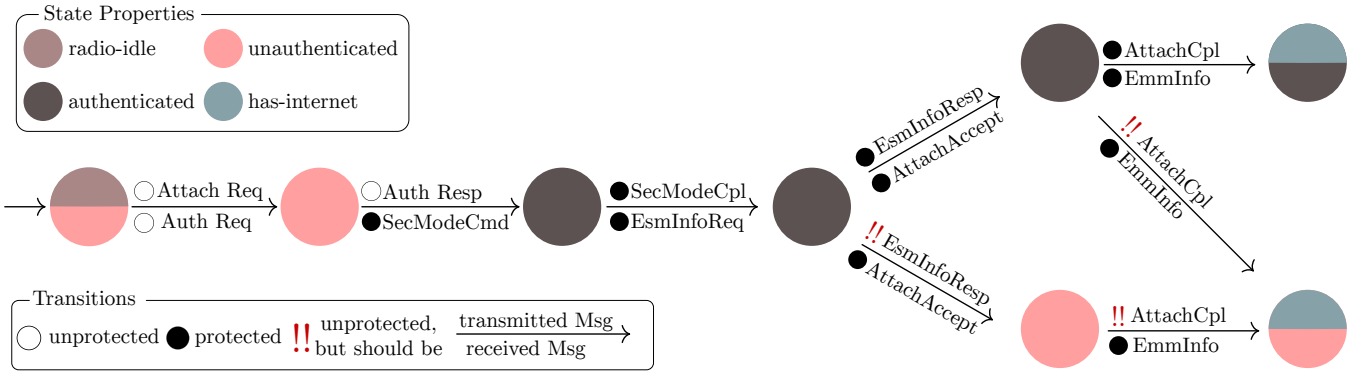

Figure 5: Path to Internet access for Open5GS. The upper path goes through authenticated states exclusively (correct), however, it accepts unauthenticated messages (lower path) after the Security Mode procedure.

- **R1:** Within our model of srsEPC, several messages lead to authenticated states which effectively provide a shortcut around re-authentication.
- **R2:** srsEPC accepts Authentication Response and Identity Response in most states, further patterns do not emerge due to the amount of states created.
- **R3:** Once a security context exists, all network-originating messages are protected, except for rejects.
- **R4** & **R5:** Crashes of srsEPC do occur upon handling unexpected Identity Responses. However, this does not always occur; sometimes, malformed messages are sent back instead without crashing. This is shown to us as non-deterministic behaviour within the state machine. After manual investigation of the code base, we found this to be an issue in addressing the internal UE context storage that leads to accessing uninitialized memory.
- **R6:** There is no unauthenticated state with Internet access.

Version 20.10. Version *20.10* drastically simplifies the network behavior, resulting in a much smaller state machine with only 9 states, despite more input symbols tested.

- **R1:** Only the Authentication and Service Request procedure lead from unauthenticated to authenticated states.
- **R2:** Unprotected messages lead to state transitions, but without network reply (timeout). These transitions have no further security implication.
- **R3:** Once a security context exists, all network-originating messages are protected, except for Rejects.
- **R4** & **R5:** We observed no crashes or non-deterministic behavior.
- **R6:** There is no unauthenticated state with Internet access.

5.3.4 Commercial Network. Testing the commercial network took place with an earlier software version that did not yet provide all opportunities used within this analysis section; essentially, input symbols like the connection test and feedback from the S1AP layers were still missing. As a result, the state learner did not yet work correctly in all cases and could not infer a complete model. Due to the current pandemic, we had to stop these tests and could unfortunately not perform additional tests in a commercial network.

Nevertheless, we gained important information from the partial state graphs and test sequences we obtained during our preliminary tests. Like all other networks, the commercial network provides a standards-compliant Attach procedure. We were able to successfully identify one issue, where the network returns an Attach Accept message *in plain text*. This result shows that our testing did hit an odd case within the protocol state machine. We conclude that our approach is capable of testing even a commercial deployment, with all additional complexity involved in such a system.

6 DISCUSSION

Selection of Core Networks. We tested two commercial and two open-source core networks. The open-source networks aim at providing a platform for test and experimentation, rather than being ready for a production-quality deployment. As such, we expected them to be less stable and prone to crashes. Still, they are suitable for developing the state learning approach and we identified several types of shortcomings in these implementations. Furthermore, we demonstrated the approach in commercial networks.

Transition from LTE to 5G. We demonstrated state machine learning for LTE networks. We expect that we can quickly adapt our approach to 5G by expanding the NAS-layer mapper to 5G-specific messages, and the S1AP transport to NGAP. However, the 5G protocols NAS and NGAP are similar to LTE, and we do not expect fundamental differences. We believe our approach can help to increase the security of future 5G networks by enabling a flexible method to recover the state machine implementation.

Over-The-Air Transmission. By implementing both the User Equipment and Base Station components in one software system, we can effectively skip the wireless transmission channel, thereby avoid additional complexity and utilize additional feedback. However, the radio transmission must be considered when interpreting the results. We deliberately trigger RRC behavior via S1AP messages, e.g., an S1AP message to simulate that the phone goes idle. Reproducing this behavior over radio connections requires that the user device actually remains idle until the base station detects inactivity. The radio link is also a crucial aspect regarding over-the-air attacks due to radio-layer encryption that may inhibit attacks.

Exploitability. The focus of our work is to verify the correct implementation of core networks and their corresponding state machines. It requires additional considerations and effort to determine if the discovered vulnerabilities are actually exploitable in practice. For example, unauthorized messages on NAS might not be exploitable due to additional security measures on the radio layer. We still informed all software authors about potential issues, which were found fixed independently.

Industry implications. In mobile networks, predefined tests are state of the art: the recently published GSMA Network Equipment Security Assurance Scheme (NESAS) [8] provides few selected test cases. These test cases must be defined in advance, and often test cases are added only as a reaction to previously discovered flaws. Further, specific test cases cannot cover the correct functionality of the complete state machine. In contrast, our approach allows comprehensive testing without defining particular test cases in advance. Consequently, our method is complementary to the industry's approach and suitable for an in-depth analysis.

7 RELATED WORK

State Machine Learning. State Machine learning originates as a formal method from the research area of software testing [22] and lately became a tool for analyzing various protocol implementations. Ruiter and Poll performed state machine learning for TLS implementations [5]. Fiterau et al. focused on the implementations of DTLS [6], which builds upon the TLS-Attacker framework[21].

In the area of wireless protocols, Stone et al. [23] use state machine learning to analyze the WiFi handshake of access points. The fundamental difference between WiFi and LTE is mobility management, leading to more complex access procedures and maintaining user contexts on multiple layers. We demonstrate instruction of multiple protocol layers and receive their feedback to enable state learning in LTE.

Mobile Network Security. The area of mobile network security research can be divided by their focus on *i*) specification and *ii*) implementation and deployment [18]. This work belongs to the latter, and we discuss only implementation and deployment-related work. The implementation of complex mobile protocols holds its own risk of implementation flaws. On the one side, implementations are prone to memory corruption bugs to a vulnerable parser. For example, Weinmann [25] found exploitable memory corruption by manual baseband analysis. Maier et al. extended the manual approach and developed a framework for fuzzing the parsers of Mediatek basebands [15]. Kim et al. analyzed 18 baseband firmware and found hundreds of mismatches with the specification [14].

In contrast to memory corruption, implementations can be also prone to logical flaws. The manual inspection of the state machine found various flaws in the phone or network, e. g., the impersonation of users towards a network [4, 19], or eavesdropping on phone calls [20]. Hong et al. [9] use meta information to find performance bottleneck in core networks. Kim et al. [13] performed semi-automated testing of protocol implementations in phones and networks with pre-generated test cases and vulnerabilities to severing multiple attacks aims, e. g., denial of service, or downgrade attacks. In contrast to their work, we start with the implementation and automatically *reconstruct* the MME's state machine, which allows us an in-depth analysis of the tested core network element without prior analysis of the standard.

8 CONCLUSION

Verifying the correct implementation of mobile core networks is challenging, considering the emerging requirements and complexity. In this paper, we propose state machine learning and model checking to increase the level of automation and coverage in testing mobile networks. We address the challenges of input message construction, non-determinism, and multi-layer feedback. We implemented a prototype based on LearnLib and srsRAN and learn partial NAS state machines of four MMEs. We discovered logic flaws that cause crashes and the processing of unauthenticated messages in open-source implementations during our evaluation. Finally, we use our approach with a commercial network. Overall, we believe that our approach can strengthen the security of the core network implementations.

9 ACKNOWLEDGEMENTS

This work was supported by the German Research Foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa – 39078197.

REFERENCES

[1] 3GPP. 2011. *Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3.* TS 24.301. 3rd Generation Partnership Project (3GPP). http://www.3gpp.org/ftp/Specs/html-info/24301.htm
[2] Amarisoft. [n.d.]. Amarisoft Core Network. https://www.amarisoft.com/. [Online; accessed June 2, 2021].
[3] Y. Chen, Y. Yao, X. Wang, D. Xu, C. Yue, X. Liu, K. Chen, H. Tang, and B. Liu. 2021. Bookworm Game: Automatic Discovery of LTE Vulnerabilities Through Documentation Analysis. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.
[4] Merlin Chlosta, David Rupprecht, Thorsten Holz, and Christina Pöpper. 2019. LTE Security Disabled — Misconfiguration in Commercial Networks. In *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. ACM.
[5] Joeri De Ruiter and Erik Poll. 2015. Protocol State Fuzzing of TLS Implementations. In *USENIX Security Symposium (SSYM)*. USENIX Association, 193–206.
[6] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. 2020. Analysis of DTLS Implementations Using Protocol State Fuzzing. In *USENIX Security Symposium (SSYM)*. USENIX Association.
[7] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. 2016. srsLTE: an open-source platform for LTE evolution and experimentation. In *ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)* (New York City, NY, USA). ACM, 25–32.
[8] GSM Association Security Group. [n.d.]. Network Equipment Security Assurance Scheme (NESAS). https://www.gsma.com/security/network-equipment-security-assurance-scheme/. [Online; accessed June 2, 2021].
[9] B. Hong, S. Park, H. Kim, D. Kim, H. Hong, H. Choi, J. P. Seifert, S. J. Lee, and Y. Kim. 2018. Peeking over the Cellular Walled Gardens - A Method for Closed Network Diagnosis. *IEEE Transactions on Mobile Computing* (2018).
[10] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Conference on Computer and Communications Security (CCS)*. ACM, 669–684.
[11] Malte Isberner, Falk Howar, and Bernhard Steffen. 2014. The TTT algorithm: a redundancy-free approach to active automata learning. In *International Conference on Runtime Verification*. Springer, 307–322.
[12] Malte Isberner, Falk Howar, and Bernhard Steffen. 2015. The Open-Source LearnLib. In *Computer Aided Verification*, Daniel Kroening and Corina S. Păsăreanu (Eds.). Springer International Publishing, Cham, 487–495.
[13] Hongil Kim, Jiho Lee, Eunkyu Lee, and Yongdae Kim. 2019. Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane. In *IEEE Symposium on Security and Privacy (SP)*. IEEE.

[14] Kim, Eunsoo and Kim, Dongkwan and Park, CheolJun and Yun, Insu and Kim, Yongdae. 2021. BASESPEC: Comparative Analysis of Baseband Software and Cellular Specifications for L3 Protocols. In *Symposium on Network and Distributed System Security (NDSS)* (San Diego, CA, USA). ISOC.

[15] Dominik Maier, Lukas Seidel, and Shinjo Park. 2020. BaseSAFE: Baseband SAnitized Fuzzing through Emulation. In *Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. ACM.

[16] Neo4J. [n.d.]. Neo4J. https://neo4j.com/. [Online; accessed June 2, 2021].

[17] Open5GS. [n.d.]. Open5GS. https://github.com/open5gs/open5gs. [Online; accessed June 2, 2021].

[18] David Rupprecht, Adrian Dabrowski, Thorsten Holz, Edgar Weippl, and Christina Pöpper. 2018. On Security Research towards Future Mobile Network Generations. *IEEE Communications Surveys & Tutorials* (2018).

[19] David Rupprecht, Kai Jansen, and Christina Pöpper. 2016. Putting LTE Security Functions to the Test: A Framework to Evaluate Implementation Correctness. In *Workshop on Offensive Technologies (WOOT)* (Austin, TX, USA). USENIX Association.

[20] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. 2020. Call Me Maybe: Eavesdropping Encrypted LTE Calls With ReVoLTE. In *USENIX Security Symposium (SSYM)*. USENIX Association.

[21] Juraj Somorovsky. 2016. Systematic fuzzing and testing of TLS libraries. In *Conference on Computer and Communications Security (CCS)*. ACM, 1492–1504.

[22] Bernhard Steffen, Falk Howar, and Maik Merten. 2011. *Introduction to Active Automata Learning from a Practical Perspective*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[23] Chris McMahon Stone, Tom Chothia, and Joeri de Ruiter. 2018. Extending automated protocol state learning for the 802.11 4-way handshake. In *European Symposium on Research in Computer Security*. Springer, 325–345.

[24] Software Radio Systems. [n.d.]. srsRAN. https://github.com/srsran/srsRAN. [Online; accessed June 2, 2021].

[25] Ralf-Philipp Weinmann. 2012. Baseband Attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks. In *Workshop on Offensive Technologies (WOOT)* (Bellevue, WA, USA). USENIX Association.