Emulation-Instrumented Fuzz Testing of 4G/LTE Android Mobile Devices Guided by Reinforcement Learning

Kaiming Fang and Guanhua Yan(✉)

Department of Computer Science, Binghamton University,
State University of New York, Binghamton, USA
{kfang2,ghyan}@binghamton.edu

Abstract. The proliferation of 4G/LTE (Long Term Evolution)-capable mobile devices calls for new techniques and tools for assessing their vulnerabilities effectively and efficiently. Existing methods require significant human efforts, such as manual examination of LTE protocol specifications or manual analysis of LTE network traffic, to identify potential vulnerabilities. In this work, we investigate the possibility of automating vulnerability assessment of 4G/LTE mobile devices based on AI (Artificial Intelligence) techniques. Towards this end, we develop LEFT (LTE-Oriented Emulation-Instrumented Fuzzing Testbed), which perturbs the behavior of LTE network modules to elicit vulnerable internal states of mobile devices under test. To balance exploration and exploitation, LEFT uses reinforcement learning to guide behavior perturbation in an instrumented LTE network emulator. We have implemented LEFT in a laboratory environment to fuzz two key LTE protocols and used it to assess the vulnerabilities of four COTS (Commercial Off-The-Shelf) Android mobile phones. The experimental results have shown that LEFT can evaluate the security of 4G/LTE-capable mobile devices automatically and effectively.

1 Introduction

The last decade has witnessed rapid advancement of mobile technologies, evolving from the 2G/GSM systems to the 3G/UMTS systems and then to the current 4G/LTE systems. The GSMA Intelligence estimates that the number of 4G/LTE connections will increase from 500 million at the end of 2014 to 2.8 billion by 2020 worldwide [12]. Currently, the majority of LTE-capable mobile devices run on Android, which has dominated the mobile OS market with a market share of 87.7% in the second quarter of 2017 [5]. The security risks posed by vulnerable mobile devices have been revealed in a number of reports, including SMS flooding attacks [18], man-in-the-middle attacks [26], and botnet attacks [13,14].

Existing research on LTE network security has mainly focused on manual analysis of potential attacks against 4G/LTE networks [28,29] or manually constructing abstract models from 3GPP standards to validate 4G/LTE protocol

J. Lopez et al. (Eds.): ESORICS 2018, LNCS 11099, pp. 20–40, 2018.
https://doi.org/10.1007/978-3-319-98989-1_2

implementations with model checkers [20,31]. Although shedding lights on the vulnerabilities of a small number of 4G/LTE mobile devices, the level of human efforts required in these works makes it difficult, if not impossible, to perform device-specific vulnerability assessment at a large scale. As there have been more than 24,000 distinct Android devices from over 1000 brands [3], there is necessity for new techniques that can automate vulnerability assessment of various 4G/LTE Android mobile devices.

Against this backdrop, we develop a testbed called LEFT (*LTE-Oriented Emulation-Instrumented Fuzzing Testbed*) for fuzz testing COTS (Commercial-Off-The-Shelf) 4G/LTE Android mobile devices. LEFT uses high-fidelity LTE network emulation to create an immersive environment for testing these devices. By perturbing the behavior of the emulated LTE network according to a user-provided threat model, LEFT elicits unexpected sequences of messages from the LTE network that, hopefully, can expose vulnerable internal states of the mobile device under test. Moreover, different from most existing fuzzers which are based upon evolutionary algorithms, LEFT uses reinforcement learning (RL) to train a fuzzer agent that not only balances its vulnerability discovery efforts between exploration and exploitation but also avoids the undesirable crashes of the LTE network emulator due to inconsistent process states after behavior perturbation.

In a nutshell, our key contributions are summarized as follows:

- We have developed three novel fuzzing methodologies, emulation-instrumented fuzzing, threat-model-aware fuzzing, and RL-guided fuzzing in LEFT to assess the vulnerabilities in LTE-capable mobile devices;
- We have implemented LEFT in a laboratory environment based on an open-source LTE network emulator and commodity SDR (Software-Defined Radio)-based devices. To achieve full automation, we adopt both a systematic approach to recover from process failures by catching OS (Operating System)-level signals and an algorithmic approach that can train the fuzzer agent to avoid actions likely to cause emulator crashes.
- Using LEFT, we have fuzzed the behavior of two key LTE protocols, one responsible for managing the connections between end devices and base stations and the other providing mobility services to end devices by the core network. The experimental results on four COTS Android mobile phones have demonstrated that LEFT is capable of discovering LTE protocol vulnerabilities automatically and effectively.

The remainder of the paper is organized as follows. Section 2 summarizes the related work. Section 3 discusses the background and methodology of this work. Section 4 presents the design of LEFT and Sect. 5 its implementation. We show the evaluation results of LEFT in Sect. 6 and draw concluding remarks in Sect. 7.

2 Related Work

LTE Network Security. Due to the rising popularity of 4G/LTE services, a number of previous works have been dedicated to analyzing and improving their

security. A comprehensive survey was given in [16] on the vulnerabilities in LTE networks at different levels; it also offered suggestions on how to address these vulnerabilities. The work in [22] surveyed possible attacks against the availability of LTE networks, including DoS (Denial of Service) attacks, DDoS (Distributed Denial of Service) attacks, and insider attacks. In [31], six types of vulnerabilities in cellular networks were reported due to problematic protocol interactions. Another recent survey summarizes the security challenges in different generations of mobile communication networks based on a variety of root causes, including specification issues, implementation issues, protocol context discrepancy, and wireless channel [27]. A theoretical security architecture was proposed in [23] to thwart smart jamming attacks in LTE networks. The work in [25] investigated the vulnerabilities of 4G/LTE modem implementations by several manufacturers. The work [32] addresses the issue of open transmission of IMSIs (International Mobile Subscriber Identities) with Pseudo Mobile Subscriber Identifiers. Our work has been motivated by the work in [29], which demonstrated practical attacks against privacy and availability in 4G/LTE networks. The work in [20] proposed LTEInspector, which combines symbolic model checking and cryptographic protocol verifier to identify vulnerabilities in LTE network protocols. A semi-automated framework was proposed in [28] to evaluate the implementation compliance of the security procedures of LTE devices.

Our contributions in this work differ from the existing works on 4G/LTE network security. *First*, it relies on fuzzing to reveal vulnerabilities automatically, which differs from the verification-based methods used in [20,31,32]. Protocol security verification requires significant efforts in developing accurate abstraction models of the protocols based on their specifications and cannot reveal implementation-specific vulnerabilities; verification based on model checking is also time consuming and thus may not be applicable to analysis of complex security protocols. *Second*, although the works in [20,25,28,29] also use an emulated LTE network environment for security evaluation, they used emulation to validate the vulnerabilities found instead of using it to fully automate the process of vulnerability discovery. Our work, instead, instruments an LTE network emulator to support perturbation of network protocol behavior. *Finally but not least*, our work explores a new direction in applying AI-based methods to search for vulnerabilities in LTE networks automatically.

Network Protocol Fuzzing. Fuzzing is a technique widely used to evaluate security protocols. Fuzzing was used to assess the security of IKE (Internet Key Exchange) implementations [30] and the security of TLS (Transport Layer Security) implementations [15,17]. The work in [21] developed T-Fuzz, a model-based fuzzing framework to assess the robustness of telecommunication protocols, and used it to fuzz a simplified state machine of a key protocol in LTE. T-Fuzz is useful for testing a protocol at its design phase but cannot be used directly to test the security of a COTS system. Our work has explored a new direction in network protocol fuzzing: it leverages the protocol implementation of an existing network emulator and instruments it to support fuzzing capabilities. As such implementation code provides more low-level details than abstract models

derived from protocol specifications, it can be used to reveal fine-grained protocol implementation vulnerabilities in COTS systems.

3 Background and Methodology

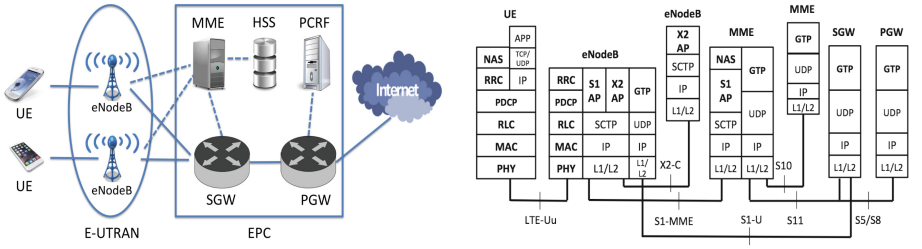
The architecture of a typical 4G/LTE network, which is shown in Fig. 1(1), consists of three key components:

- **UE (User Equipment):** A UE is a mobile device (e.g., a cell phone) used by an end user to communicate through the LTE network.
- **E-UTRAN (Evolved Universal Terrestrial Radio Access Network):** The E-UTRAN consists of base stations called eNobeBs in LTE parlance.
- **EPC (Evolved Packet Core):** The EPC is an LTE system's packet-only core network. Its S-GW (Serving Gateway) is responsible for handoffs among different eNodeBs and data transfer of packets across the u-plane of LTE. The P-GW (PDN Gateway) acts as a middleman between the LTE network and other packet data networks such as the Internet. The MME (Mobility Management Entity) is the key control node of an LTE system, performing functionalities such as initiating paging and authentication of mobile devices, location tracking and radio resource management. The HSS (Home Subscriber Server) is a database that stores users' subscription information. Finally, the PCRF (Policy and Charging Rules Function) module supports flow-based charging and enforces policy rules on users' data flows.

The LTE protocol specification defined by the 3GPP organization is complex, covering thousands of pages [1], and as illustrated in Fig. 1(2), it describes the rules and conventions at multiple protocol layers in each LTE module. Clearly, it is a daunting task to verify whether a COTS 4G/LTE mobile device has a security vulnerability due to its non-conformance with the 3GPP standards. Moreover, vendors usually do not reveal the details on how their Modem firmware implement the LTE protocols, making it difficult for white-hat security researchers to find vendor-specific security bugs. Finally but not least, as evidenced by previous security analysis of LTE protocols [20,31], the LTE protocol specification itself may have design flaws that can pose serious security risks to the mobile users.

In the design of LEFT – a testbed dedicated to assessing the vulnerabilities of COTS 4G/LTE-capable Android mobile devices – we adopt three novel fuzzing methodologies to overcome the aforementioned challenges:

(1) **Emulation-instrumented blackbox fuzzing:** Due to the difficulty of knowing the LTE implementation specifics in a mobile device under test, we treat it as a blackbox when assessing its vulnerabilities. To this end, we create an immersive environment in which the test device can interact with an emulated LTE network. By instrumenting the emulation code, we perturb the behavior of the emulated LTE network, hoping that the reply packets from the network are unexpected by the LTE protocol implementation of the test device and thus elicit its vulnerable internal states.



(1) 4G/LTE network architecture (2) 4G/LTE protocol stacks

Fig. 1. The architecture and protocol stacks of an LTE network. Its key protocols include: NAS (Non-Access Stratum), RRC (Radio Resource Control), PDCP (Packet Data Convergence Protocol), and RLC (Radio Link Control).

(2) **Threat-model-aware fuzzing:** In a network environment, there can be a wide spectrum of threat vectors, who may have different capabilities and attack goals. By differentiating threat models, we are able to design customized fuzzing strategies aimed at specific attack scenarios to improve fuzzing efficiency. For instance, an attack scenario aimed at compromising confidentiality may differ significantly from that targeting availability.

(3) **RL-guided fuzzing:** Inspired by the success of RL in solving real-world problems (e.g., playing against professional go players), we use RL-guided fuzzers to balance efforts on exploring new regions in the fuzzing space and exploiting previous fuzzing experiences. One hurdle we face in perturbing the behavior of the emulated LTE network is that it constantly causes the LTE emulator to crash due to inconsistent process states after behavior perturbation. When the LTE emulator crashes, we have to restart the LTE emulator from a clean state, which incurs significant computational overhead. Using reinforcement learning, we train the fuzzer to avoid actions that are likely to cause emulator crashes.

4 Design

The high-level architecture of LEFT is illustrated in Fig. 2. It requires a user-provided threat model, which defines the security goal of the attacker (e.g., information leakage) as well as his prior knowledge. Based on the threat model, a configuration file is generated to set up the testbed. The hardware needed by the testbed includes the 4G/LTE Android device under test, a combination of USRP B210 boards and LTE antennae that enable SDR-based communications between the UE and the emulated LTE network, and workstations that are used to emulate the LTE network and perform fuzzing. The LEFT software adopts a client-server structure which includes:

– **Client:** The client interacts with the 4G/LTE Android device under test through ADB (Android Debug Bridge). The client can execute ADB commands to elicit responses from the device, and has a sensor that parses these

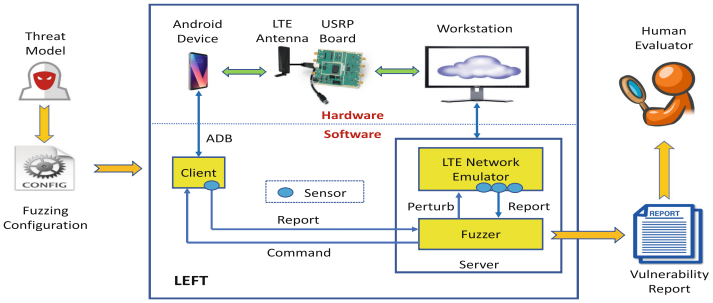


Fig. 2. The architecture of LEFT

responses to infer its internal states. The client reports these inferred states to the server and also accepts commands from it.
- **Server:** The controller of the server is a fuzzer that perturbs the behavior of an emulated LTE network. The LTE network emulator is instrumented to take a perturbation action requested by the fuzzer for each incoming message from the test device. It also includes sensors that can assess the vulnerabilities based on the messages received.

The output of a fuzzing campaign is a list of vulnerabilities in the test device that are exploitable under the threat model assumed. The vulnerability report is sent to a human evaluator for further examination.

4.1 Threat Model

Due to the integrated emulation functionality within LEFT, it can be used to construct a variety of attack scenarios. For each attack scenario, the LEFT user needs to specify the following:

- **Knowledge:** Does the attacker know the victim UE's authentication credential, such as the secret key stored in its SIM card?
- **Security goal:** What security goal does the attacker want to compromise (e.g., confidentiality, integrity, or availability)?
- **Capability:** Can the attacker emulate a malicious LTE network or UE?
- **Protocol:** What is the specific LTE protocol, as illustrated in Fig. 1(2), that the attacker would like to focus on?
- **Action:** What kind of actions can the attacker take? There are two options for fuzzing protocol messages, *bit-level* and *order-level*. Most network protocol fuzzers implement bit-level message fuzzing, which randomly flips the bits in a protocol message. In contrast, order-level message fuzzing perturbs the order of protocol messages to find logic flaws in the protocol implementation.

Focus of This Work. As the attack surface in an LTE network is so large that it cannot be tackled all at once within a single work, this study focuses on a practical threat model where the attacker is able to emulate a malicious LTE

network, including its core EPC network and eNodeBs, but knows nothing about the victim's secrets such as the keys stored in the SIM card. The malicious eNodeB can force a victim UE to make a connection to itself with high signal power, as a typical UE constantly measures the received signal strengths from neighboring eNodeBs and chooses the one with the strongest power during a hand-over procedure [24]. Moreover, the threat model assumed in this work considers the attack scenarios where the attacker is interested in compromising either confidentiality or availability, although the work can be easily extended to evaluate the integrity of the test UE. Finally, this work assumes the actions of the attacker as perturbing the order of protocol messages, so its focus is to find logic-level vulnerabilities in the LTE protocol implementation of the test UE instead of its low-level software vulnerabilities such as buffer overflow or use-after-free bugs.

4.2 Sensors

To evaluate the security status of the victim UE, we deploy two types of sensors:

- **Client-side sensors:** A client-side sensor parses the UE's responses to ADB commands and infers its status. For example, the client can issue an ADB shell ping to test the UE's network connectivity. Its responses can be used to assess the availability of the network to the UE.
- **Server-side sensors:** On the server side, we instrument the code of an LTE network emulator and place a sensor at each protocol layer to infer leakage of sensitive data without proper encryption. As a protocol layer only parses a certain portion of an incoming message, its corresponding sensor detects if there exists any sensitive data unencrypted at that layer. If it is true *and* no other sensor at any lower layer on the protocol stack detects that the payload has been decrypted by the protocol on that layer, an information leakage compromising confidentiality occurs.

It is noted that to compromise confidentiality, the attacker can sniff LTE packets in the air and parse them with tools such as Wireshark [11]. Although it is possible to use passive packet sniffers along with sensors to detect information leakage, their deployment requires additional computational resources and incurs unnecessary communication latency between the sensors associated with the sniffers and the fuzzer. As the LTE network emulator needs to parse the messages anyway, we can place the sensors *inside* the network emulator in a non-intrusive manner to infer the possible leakage of sensitive information.

4.3 RL-Guided Fuzzing

The key component of LEFT is its fuzzer, which decides how to perturb the behavior of the LTE network emulator based on the reports collected from the various sensors deployed. Due to the No-Free-Lunch-Theorem [33], there is no optimal fuzzing strategy suitable for all situations. As LEFT is aimed at discovering vulnerabilities in a variety of 4G/LTE Android mobile devices, we design

the fuzzer inside the server to be an intelligent agent that can balance exploring test scenarios with unsure results and exploiting those close to the expectation based on its previous experiences. Such a tradeoff between exploration and exploitation is well studied within the RL framework, which is built upon the Markov decision process (MDP) model (S, A, P, R, γ):

- **S**: a finite set of states, including both environment states and agent states;
- **A**: a finite set of actions that can be taken by the agent;
- **P**: $P_a(s, s')$ gives the transition probability from state s to s' after the agent takes action a;
- **R**: $R_a(s, s')$ is the immediate reward received by the agent after taking action a, which leads to state transition from s to s';
- γ: $\gamma \in [0, 1]$ is a per-step discount of the reward.

An important concept in RL is its value functions, including state values $V(s)$ and state-action values $Q(s, a)$, which reflect the agent's long-term returns of entering state s and taking action a at state s, respectively. To apply RL for perturbing the behavior of the LTE network emulator, we need to extract a MDP model from its emulation code. Network protocols are usually developed from FSM (Finite State Machine) specifications, and as a result, they are often implemented in an event-driven programming style. Hence, when an incoming message is received by a protocol layer, the event is processed by the emulator based on the message type and sometimes a few internal state variables. Accordingly, in the MDP model, we use the collection of message types as its state set S. The action set of the agent A includes all possible cases dealing with the different types of incoming messages. In some cases, the emulator's action depends upon an internal state variable. For example, after receiving the same type of messages, the emulator does a if an internal state c is true, or does a' otherwise. Then, both actions a and a' are added to the agent's action set A.

In the RL framework, the fuzzer agent works by demanding the LTE emulator to choose a certain action $a \in A$ when a message of type $s \in S$ is received. This action a may not be the proper one according to the LTE protocol specification. Our hope is that, by choosing nonconforming actions on the LTE network side, it returns unexpected messages to the victim UE that can reveal its vulnerabilities.

The key challenge in applying RL-guided fuzzing is the *delayed reward issue*, which manifests itself for both confidentiality and availability:

- **Confidentiality:** For state-action pair (s, a), action a in state s may induce a message M, which is sent to the victim UE by the LTE network emulator; after the UE receives this packet, it sends back another one M' to the LTE network emulator according to its LTE implementation. The reward of state-action pair (s, a) is calculated by a sensor inside the LTE network emulator after it receives message M'. Hence, there is a round-trip delay before the agent can collect its reward.
- **Availability:** Consider the case where the security goal is to assess whether the victim UE is able to connect to a benign LTE network (which is emulated separately from the malicious one being perturbed) after the attack. The

agent, however, cannot assess the UE's loss of network connectivity directly; instead, it sends a command to the client (see Fig. 2), which further sends an ADB command to the UE to test its network connectivity with a benign LTE network. The sensor located inside the client reports the test result to the agent in the server for calculating its reward. Different from the case of confidentiality, the fuzzer agent cannot calculate its reward after each state-action pair (s, a) because this requires the victim UE to test its network connectivity with a benign LTE network after every message it has received.

With the delayed reward issue in mind, we develop the RL-guided fuzzing algorithm as illustrated in Algorithm 1. In parlance of RL, an *episode* consists of a sequence of states and actions: $(s_0, a_0, s_1, a_1, ..., a_{n-1}, s_n)$ where s_n is a terminal state. In LEFT, a terminal state is defined as one of the following three cases:

- $S_{timeout}$: After taking action a_{n-1}, the LTE network emulator has not heard from the UE after τ seconds. This indicates either the end of a normal protocol procedure or a preemptive termination action by the test UE.
- S_{limit}: To prevent an episode from running indefinitely without revealing any vulnerability, we set an upper bound δ on the number of steps in each episode. If the length of an episode exceeds the limit, it is forced to terminate.
- S_{crash}: Perturbing the behavior of the LTE network emulator may crash the process due to inconsistent internal states of the emulator process. When the emulator crashes, the current episode terminates.

Table 1. Notations used in Algorithm 1 and their meanings

Notation	Meaning	Default value
goal	Security goal (`CONFIDENTIALITY` or `AVAILABILITY`)	User-provided
N	Order of the UE being tested	0
Q_c^{avg}	Average Q-table for evaluating confidentiality	**0**
Q_a^{avg}	Average Q-table for evaluating availability	**0**
Q_c	Q-table for evaluating the current UE's confidentiality	Q_c^{avg}
Q_a	Q-table for evaluating the current UE's availability	Q_a^{avg}
δ	Maximum number of steps in an episode	50
τ	Timeout for waiting the UE's messages	10 s
α	Learning rate	0.1
γ	Reward discount per step	0.9
ϵ_{min}	Minimum probability of exploration per step	0.1
β^{crash}	Agent's reward if the emulator crashes	-100
β_c^{leak}	Agent's reward if there is an information leakage	100
β_a^{loss}	Agent's reward if the UE loses network connectivity	100
episodes	Number of episodes performed for the current UE	0
steps	Number of steps performed within the current episode	0
L	List of state-action pairs in the current episode	$\emptyset$

Algorithm 1. RL_GUIDED_FUZZING($goal$, Q_c^{avg}, Q_a^{avg}, N)

$\alpha \leftarrow 0.1$, $\epsilon_{min} \leftarrow 0.1$, $\delta \leftarrow 50$, $\gamma \leftarrow 0.9$, $\tau \leftarrow 10$
$\beta^{crash} \leftarrow -100$, $\beta_c^{leak} \leftarrow 100$, $\beta_a^{loss} \leftarrow 100$
initialize Q_c and Q_a to be Q_c^{avg} and Q_a^{avg}, respectively
$episodes \leftarrow 0$
for each episode **do**
    $episodes \leftarrow episodes + 1$, $(s, a) \leftarrow (null, null)$, $steps \leftarrow 0$, $L \leftarrow []$
    restart the LTE network emulator, command the client to restart the UE
    for each new message M received from the victim UE **do**
        if M leads to information leakage **then**
            $r_c \leftarrow \beta_c^{leak}$
        else
            $r_c \leftarrow 0$
        end if
        append state-action pair (s, a) to list L
        $s' \leftarrow$ the type of message M
        if $goal =$ `CONFIDENTIALITY` **then**
            $Q = Q_c$
        else if $goal =$ `AVAILABILITY` **then**
            $Q = Q_a$
        else
            error
        end if
        $a' \leftarrow \epsilon$-greedy(s', Q, $\max\{1/episodes, \epsilon_{min}\}$)
        $Q_c(s, a) \leftarrow Q_c(s, a) + \alpha(r_c + \gamma \max_{a''}(Q_c(s', a'') - Q_c(s, a)))$
        $steps \leftarrow steps + 1$
        if $steps > \delta$ **then** break **end if**
        let the LTE network emulator take action a'
        if the emulator crashes **then**
            $Q_c(s', a') \leftarrow \beta^{crash}$, $Q_a(s', a') \leftarrow \beta^{crash}$
            append state-action pair (s', a') to list L
            break
        end if
        $(s, a) \leftarrow (s', a')$
        schedule a timer which fires after τ seconds
        wait for a new incoming message or the firing of the timer
        if the timer fires **then** break **else** cancel the timer **end if**
    end for
    send a command to client to test the UE's network connectivity
    wait for the report from the client
    if the client reports that the UE has good network connectivity **then**
        $r_a \leftarrow 0$
    else
        $r_a \leftarrow \beta_a^{loss}$
    end if
    for the i-th state-action pair (s_i, a_i) on list L where $1 \leq i \leq |L|$ **do**
        $Q_a(s_{i-1}, a_{i-1}) \leftarrow Q_a(s_{i-1}, a_{i-1}) + \alpha(\frac{1/2^{(|L|-i+1)}}{1-1/2^{|L|}} \cdot r_a + \gamma \max_{a''}(Q_a(s_i, a'') - Q_a(s_{i-1}, a_{i-1})))$
    end for
end for
$N \leftarrow N + 1$, $Q_c^{avg} \leftarrow Q_c^{avg} + \frac{1}{N}(Q_c - Q_c^{avg})$, $Q_a^{avg} \leftarrow Q_a^{avg} + \frac{1}{N}(Q_a - Q_a^{avg})$

Table 1 summarizes the notations used in Algorithm 1 and their meanings. In Algorithm 1, function ϵ-greedy(s, Q, ϵ) returns a random action with probability ϵ (i.e., exploration) and $\text{argmax}_a Q(s, a)$ with probability $1-\epsilon$ (i.e., exploitation).

Algorithm 1 is called when a new UE needs to be tested for a user-provided security goal, either confidentiality or availability. Assuming that this UE is the N-th one tested, the average Q-tables for both confidentiality and availability calculated over the previous $N-1$ UEs are used to initialize the Q-tables for evaluating the current UE. In RL, these Q-tables are crucial to the agent's decision-making as each entry (s, a) stores the agent's estimated long-term value of taking action a at state s. By initializing the current UE's Q-tables as the Q-tables averaged over the previous ones tested, it is assumed at the very beginning that there is nothing special about this new UE, which should have similar vulnerabilities as the ones tested.

It is, however, possible that the new UE may be different from the ones previously tested (e.g., a new model just released to the market). We thus update the Q-tables associated with this UE dynamically based on its test results, hoping that they can lead to an optimal policy in generating test cases (by perturbing the behavior of the LTE network emulator) for revealing its vulnerabilities. After the evaluation of this UE, its Q-tables are used to update the average Q-tables in an incremental manner (see Line 49 in Algorithm 1).

As no explicit MDP model is needed in Algorithm 1, it adopts a model-free approach to reinforcement learning. This is desirable because in practice it is usually difficult to derive an accurate yet efficient environment model such as its state transition probabilities. Model-free control samples the environment by repeating test episodes (see Line 5 in Algorithm 1). Algorithm 1 uses Q-learning, a popular model-free off-policy RL technique, to update its Q-tables (Lines 24 and 46). For confidentiality, the agent knows the reward of its previous action at each step of an episode (when it receives a new message from the UE). Hence, the agent can use its reward received to update its Q_c-table immediately without waiting for the episode to terminate (see Line 24 of Algorithm 1). For availability (i.e., network connectivity), the agent needs to test if the UE is able to connect to the benign LTE network after each episode finishes; hence, the corresponding Q_a-table is updated after an episode finishes (see Line 46 of Algorithm 1).

There is a subtle issue when assessing the UE's vulnerabilities related to network connectivity. Consider a complete episode $(s_0, a_0, ..., s_{n-1}, a_{n-1}, s_n)$, which ends due to either a timeout ($s_n = S_{timeout}$), the limit on the episode length ($s_n = S_{limit}$), or a crash of the LTE network emulator ($s_n = S_{crash}$). Suppose that after the episode, through the client, it is found that the UE cannot connect to a benign LTE network any more. The overall reward for this episode is β_a^{loss}, but we need to assign the credits among the state-action pairs in it, which is a classical problem in reinforcement learning. In Algorithm 1, we use an exponential credit assignment scheme based on the assumption that later state-actions in the episode should be given more credits than the previous ones: the credit assigned to each state-pair (s_i, a_i) is half of that of (s_{i+1}, a_{i+1}), which leads to the equation on Line 46 in the algorithm.

Different from a standard RL algorithm, Algorithm 1 updates two Q-tables, Q_c for confidentiality and Q_a for availability. As seen among Lines 16–23, the security goal (i.e., the *goal* parameter of Algorithm 1) decides which Q-table should be used to generate the next action for perturbing the emulator's behavior (*on-goal learning*), while the other one is updated in an offline fashion (*off-goal learning*). It is noted that on/off-goal learning should *not* be confused with *on/off-policy learning* in RL parlance because the Q-learning technique, which belongs to the category of off-policy learning, is used for updating both Q_c and Q_a (Lines 24 and 46). The purpose of combining both on-goal and off-goal learning is to improve efficiency: although the LEFT user specifies one security goal in her threat model, the fuzzing experiments performed may reveal a different type of vulnerabilities in the UE. By incorporating these findings into the corresponding average Q-table, they can benefit later experiments aimed at finding the same type of vulnerabilities.

Algorithm 1 can be easily extended for finding multiple fine-grained types of UE vulnerabilities. For example, sensitive information of a UE includes its IMSI, its exact location, and the user's voice data. At the cost of maintaining a separate Q-table for each vulnerability type, it may be useful to learn a separate policy for perturbing the LTE emulator's behavior for revealing each fine-grained type of information leakage. With the aforementioned off-goal learning approach, we can use the same set of experiments to update the multiple Q-tables maintained for different types of vulnerabilities.

5 Implementation

The software component of LEFT adopts a client/server architecture, as seen in Fig. 2. The client is written in around 280 lines of Python code, and on the server side, the RL-guided fuzzing algorithm is written in around 350 lines of C code embedded within the OpenAirInterface LTE network emulator [8]. The fuzzer agent uses the same process as the network emulator, but replaces its control flow with Algorithm 1.

5.1 Emulator Instrumentation

There are multiple open-source LTE network emulators, such as LENA [7], OpenAirInterface [8], openLTE [9], and srsLTE [19]. After surveying these different emulators, we decide to use OpenAirInterface for the LTE network emulator in LEFT because it offers the most comprehensive functionalities with emulation code for all the key LTE network elements. The drawback of this choice, however, is that the codebase of OpenAirInterface is complex, making it a daunting task to understand and revise its source code. Another challenge in incorporating the OpenAirInterface emulator into LEFT is that some of its documentation has become outdated due to its active development. As illustrated in Fig. 1(1), an LTE network consists of a variety of network elements.

The OpenAirInterface LTE network emulator separates them into two different modules, `openair-cn` and `openairinterface5G`. The `openair-cn` module emulates the LTE core network (i.e., its EPC), including its MME, HSS, S-GW, and P-GW, while `openairinterface5G` focusing on emulation of UEs and eNodeBs. The current implementation of LEFT builds upon the development branch of `openairinterface5G` downloaded on November 14, 2017 and the `nas_pdn_type_ipv6_handle` branch of `openair-cn` downloaded on July 28, 2017 from their respective github pages.

To reveal the protocol-level vulnerabilities of a test UE, we need to instrument the emulation code of the corresponding protocol in the OpenAirInterface emulator by supporting the perturbation functionality. There are various protocols involved in LTE communications, as seen from Fig. 1(2). In this work, we focus two of them, *RRC* and *EMM* (EPS (Evolved Packet System) Mobility Management), both of which play a key role in LTE network security.

RRC: The RRC protocol is responsible for controlling the air interface between the UE and the E-UTRAN, and its functions include broadcast of system information for both the non-access and the access strata, establishment, maintenance and release of RRC connections between the UE and the E-UTRAN and radio bearers, RRC connection mobility functions, paging, control of ciphering, RRC message integrity protection, UE measurement reporting, and so on [2]. The RRC protocol is implemented by the `openairinterface5G` module. In file `openairinterface5g/openair2/RRC/LITE/rrc_eNB.c`, function `rrc_enb_task(void* args_p)` contains a message processing loop, which uses the message type of an incoming message `msg_p` (i.e., `ITTI_MSG_ID(msg_p)`) to decide how to process the message. The loop contains 16 switch cases, including the default one, and each of these cases defines a respective action. Hence, we have 16 states and 16 actions defined for the purpose of RL.

EMM: EMM is a NAS-stratum protocol dealing with mobility over an E-UTRAN access in the control plane between the UE and the MME module in the core network. The 3GPP organization specifies three types of EMM procedures: *common procedures* include GUTI (Global Unique Temporary ID) reallocation, authentication, security mode control, identification, and EMM information, *specific procedures* define mechanisms for attaching to the EPC and detaching from it, and *connection management procedures* manage UEs' connections with the core network such as service request, paging procedure, and transport of NAS messages. The EMM protocol is implemented within the `openair-cn` module as it deals with functionalities of the LTE core network.

In `openair-cn`, message processing by the EMM protocol is done in two functions in file `openair-cn/SRC/NAS/EMM/SAP/emm_as.c`: `_emm_as_recv()`, which decodes and processes normal EMM messages, and `_emm_as_establish_req()`, which processes connection establishment requests for EMM access stratum SAP (Service Access Point). In function `_emm_as_recv()`, there are 12 switch cases for the message type (`emm_msg->header.message_type`), each representing a unique state in RL. In nine of these 12 cases, the emulator's action further depends on a Boolean variable, `emm_cause`; hence, we define 21 separate actions.

In function `_emm_as_establish_req()`, there are five switch cases for the message type (`emm_msg->header.message_type`), but two of them are identical as in function `_emm_as_recv()`. As processing one of these five cases does not depend on variable `emm_cause`, function `_emm_as_establish_req()` introduces five new actions. In total, there are 15 states and 26 actions defined for the purpose of RL.

5.2 Testbed Automation

An important goal of LEFT is to minimize human efforts in assessing vulnerabilities of 4G/LTE Android mobile devices. A key challenge in testbed automation is to prevent emulator crashes. As we perturb the behavior of the LTE network emulator, it is possible that the action taken by the emulator results in a faulty internal state and thus causes it to crash or freeze. In Algorithm 1, we use a negative reward β^{crash} to discourage the fuzzer agent from taking an action that causes the emulator to crash at a certain state. However, the network emulator may still crash or freeze due to an inappropriate action taken by the agent working in an exploration mode.

The fuzzer agent's internal states, such as its Q-tables and reward tables, are declared as global variables of the network emulator. To prevent loss of these states after the emulator crashes, we modify the emulator's handler of its logging timer, which fires every 30 s, and periodically save these states onto persistent storage. Moreover, LEFT applies two techniques deployed at different places to detect if the network emulator crashes or freezes:

- On the server side, we add sensors inside the LTE network emulator to catch certain Linux signals that indicate system failures. These signals include SIGTERM (process termination), SIGSEGV (invalid memory reference), SIGABRT (abort), and SIGFPE (floating point exception). When a sensor catches such a signal, it saves the fuzzer's internal states, as well as the last state-action pair retrieved from the crashed network emulator, onto persistent storage. When the fuzzer agent recovers, it collects a reward for the last state-action pair as β^{crash}.
- For the rare cases where some faulty situations are not caught by the Linux signal handlers, LEFT uses heartbeat messages communicated between the client and the server every two seconds. If the client has not received the heartbeat message from the server after 20 s, the client restarts the server. Since the client does not know the exact state-action pair that causes the emulator to crash, the fuzzer agent, after it is restarted, *cannot* collect a negative reward for its last action (we do not save every state-action pair onto persistent storage due to its high overhead). In this case, the RL algorithm cannot prevent the agent from taking the same action that has caused the network emulator to crash or freeze in the future.

At the end of each episode, the client clears the test UE's internal state by toggling its airplane mode with appropriate ADB shell commands.

6 Evaluation

We perform experiments to evaluate the effectiveness of LEFT in assessing both confidentiality and availability. We configure LEFT to emulate two LTE networks, one benign and the other malicious:

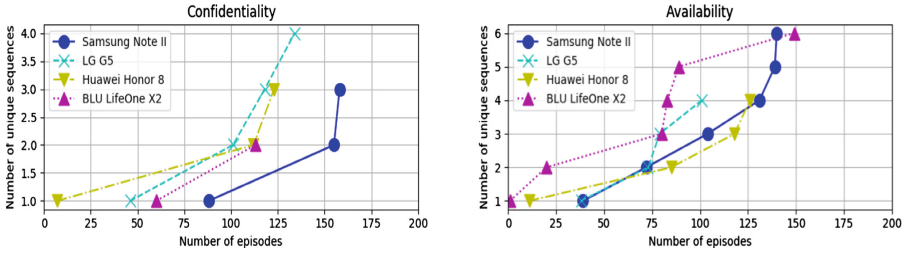
- **Benign LTE network:** Its EPC is emulated with a Lenovo ThinkCenter mini-workstation (CPU: Intel i7 6700T, RAM: 16 GB, and Ubuntu 16.04 with kernel 4.13) and its eNodeB is emulated with a Dell Inspiron 5000 laptop (CPU: Intel i7 6500U, RAM: 8G, and Ubuntu 16.04 with low-latency kernel 4.13). The communications between the test UE and the benign eNodeB are supported by a set of USRP B210 board and LTE antenna. The benign LTE network is assumed to know each test UE's authentication credentials.
- **Malicious LTE network:** A single commodity desktop PC (CPU: Intel i7 4790, RAM 32G, and Archlinux with kernel 4.15) is used to emulate both the malicious LTE network's EPC and eNodeB. The EPC is emulated within a VirtualBox VM (Virtual Machine) configured to have 4G RAM and run Ubuntu 16.04 with kernel 4.10, and the eNodeB within another VirtualBox VM configured similarly, except that it uses a low-latency kernel. A different set of USRP B210 board and LTE antenna is used for the communications between the UE and the malicious eNodeB. The malicious LTE network can be configured based on the user-provided threat model (see Sect. 4.1).

The parameters of Algorithm 1 in LEFT are configured with their default values, which have been shown in Table 1. We use LEFT to assess the vulnerabilities of four COTS Android mobile phones, `Samsung Note II` (unknown LTE Cat 4 Modem), `LG G5` (Snapdragon X12 LTE Modem), `Huawei Honor 8` (Balong 720 chipset), and `BLU LifeOne X2` (Snapdragon X6 LTE Modem). Each of these phones uses a sysmoUSIM-SJS1 SIM card [6] recommended by the OpenAirInterface developers.

We consider two sets of experiments: fuzzing the RRC protocol to compromise confidentiality and fuzzing the EMM protocol in NAS to compromise availability (see Fig. 1(2)). As it is trivial to catch the IMSIs in the air [32], such vulnerabilities are thus ignored from our experiments. For checking network connectivity, we let the test UE ping any machine on the Internet through the benign LTE network, and if the ping test fails after 30 s, the UE is deemed to lose the LTE network connectivity.

6.1 Perturbation Sequences with Vulnerabilities Discovered

We first assess the performance of LEFT in discovering the vulnerabilities in the four test UEs individually without off-goal learning. In each experiment, we evaluate one test UE for 200 episodes, with either confidentiality or availability as the goal. All experiments are performed independently. After each experiment, we collect the unique *perturbation sequences* that have led to the compromise of the pre-defined security goal, where a perturbation sequence is defined as the sequence of state-action pairs observed in an episode.



(1) Confidentiality, RRC fuzzing (2) Availability, EMM fuzzing

Fig. 3. Number of unique perturbation sequences with vulnerabilities discovered

Figure 3 shows the number of unique perturbation sequences with vulnerabilities discovered throughout an evaluation experiment for each phone. We observe that for each phone tested, there are two to four unique perturbation sequences that have revealed a confidentiality vulnerability, and four to six unique perturbation sequences that lead to loss of network connectivity.

To verify whether each vulnerability discovered is correct, we present in Table 3 the list of state-action pairs in each unique perturbation sequence that has revealed a confidentiality vulnerability in LG G5. Common to each perturbation sequence is a measurement report received from the test UE in a `Connection Reconfiguration Complete` message. According to 3GPP standard, the measurement report from a UE may include detailed location information (e.g., GNSS (Global Navigation Satellite System) location information), physical cell identity of the logged cell, and cell identities and carrier frequencies of neighboring cells [4], which presents a confidentiality risk. This vulnerability has been reported in the work in [29] but our work discovers it automatically through fuzzing.

In Table 3, we show the six perturbation sequences that cause loss of network connectivity for Samsung Note II. Common to these sequences are responses with `TAU reject` or `attach reject`. As neither `TAU reject` nor `attach reject` messages are integrity-protected, for certain rejection causes, the UE may not only lose the connectivity with the malicious eNodeB but also with the benign one [10,29]. Hence, the vulnerabilities automatically found by LEFT are indeed exploitable by an adversary under the assumed threat model.

On average, for each phone it takes 253 min to finish a confidentiality test of 200 episodes, and 277 min an availability test of 200 episodes – *without any human supervision*. It may seem unintuitive that an availability test needs only slightly more time than a confidentiality test, because it requires testing the network connectivity with the benign LTE network and if the network connectivity is lost the UE has to wait for 30 s of ping tests. Actually the fraction of episodes with lost network connectivity is small, which explains why the execution times for a confidentiality test and an availability test are comparable.

Table 2. Perturbation sequences leading to location leakage (each tuple shows the type of message received and the type of message by which it is processed)

No	Perturbation sequence
1	(RRC Connection Request, RRC Connection Setup), (RRC Connection Setup Complete, RRC Connection Reconfiguration), (RRC Connection Reconfiguration Complete + Measurement Report, RRC Connection Release)
2	(RRC Connection Request, RRC Connection Setup), (RRC Connection Setup Complete, RRC Connection Reconfiguration), (RRC Connection Reconfiguration Complete + Measurement Report, Security Mode Command), (Security Mode Reject, RRC Connection Release)
3	(RRC Connection Request, RRC Connection Setup), (RRC Connection Setup Complete, RRC Connection Reconfiguration), (RRC Connection Reconfiguration Complete + Measurement Report, RRC Connection Release)
4	(RRC Connection Request, RRC Connection Setup), (RRC Connection Setup Complete, RRC Connection Reconfiguration), (RRC Connection Reconfiguration Complete + Measurement Report, RRC Connection Reconfiguration)

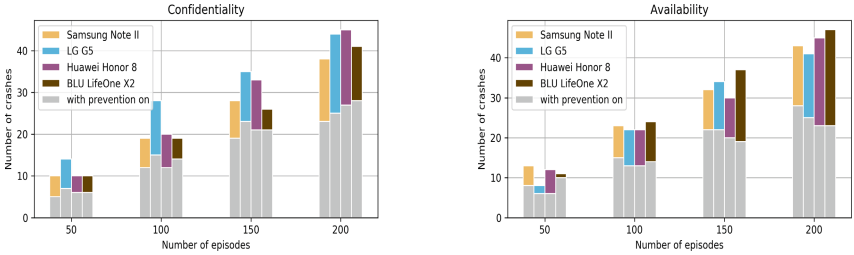
Table 3. Perturbation sequences leading to loss of network connectivity (each tuple has the same meaning as in Table 2)

No	Perturbation sequence
1	(TAU request, TAU reject)
2	(TAU request, detach request), (detach accept, paging), (attach request, attach reject (with random cause))
3	(TAU request, TAU reject), (attach request, identity request), (identity response, detach request (reattach required)), (detach accept, paging), (attach request, attach reject (with random cause))
4	(TAU request, TAU reject), (attach request, attach reject (with random cause))
5	(TAU request, TAU reject), (attach request, authentication request), (authentication response, authentication reject)
6	(TAU request, TAU reject), (attach request, authentication request), (authentication failure, identity request), (identity response, detach request), (detach accept, paging), (attach request, attach reject (with random cause))

Note: TAU stands for Track Area Update.

6.2 Prevention of Emulator Crashes

The RL-guided fuzzing algorithm shown in Algorithm 1 assigns a negative reward to discourage the fuzzer agent from taking actions that may lead to emulator crashes. To show its effect, we compare the numbers of emulator crashes with and without using a negative reward when the emulator crashes. The results are illustrated in Fig. 4. In the case of confidentiality with RRC fuzzing, using a negative reward, after 200 episodes the number of emulator crashes has been reduced by 39.47%, 43.18%, 40.00%, and 31.70% for Samsung Note II, LG G5, Huawei Honor 8, and BLU LifeOne X2, respectively, and the average reduction rate over the four phones is 38.59%. Similarly in the case of availability with EMM fuzzing, using a negative reward reduces the number of emulator crashes by 34.88%, 39.02%, 48.89%, and 51.06%, respectively, for these same phones after 200 episodes, leading to an average reduction rate of 43.47%. Clearly, the fuzzer agent benefits significantly from exploiting its previous crash experiences to avoid crash-causing actions.



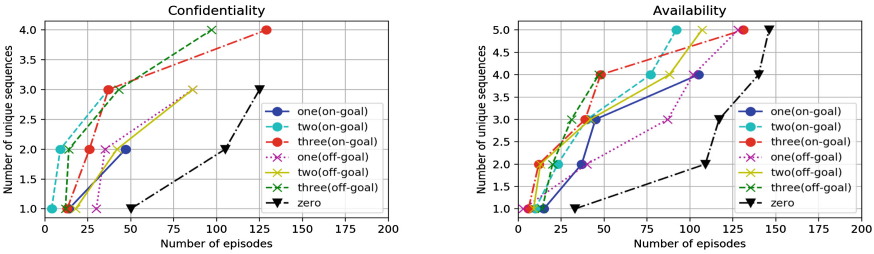
(1) Confidentiality, RRC fuzzing (2) Availability, EMM fuzzing

Fig. 4. Number of emulator crashes

6.3 Cross-Device Learning

In addition to using a negative reward to avoid actions likely to crash the emulator process, the fuzzer agent can also exploit the experiences with previous test UEs. Recall that in Algorithm 1, for each of confidentiality and availability, an average Q-table is maintained over all the UEs tested. We next perform a set of experiments to evaluate the effects of using such an average Q-table to bootstrap the Q-table when evaluating a new UE. In each experiment, we use `Huawei Honor 8` as the target phone (always evaluated last). To calculate the average Q-table, we choose zero to three phones from `Samsung Note II`, `LG G5`, and `BLU LifeOne X2` in the same order and evaluate them sequentially. When a security goal is specified for the target phone, the remaining phones can be tested either on goal (using the same security goal) or off goal (using a different security goal).

Figure 5 confirms the advantages of cross-device learning. When the target phone is evaluated first without any previous experiences for guiding vulnerability discovery, the fuzzer agent needs more episodes to find the same number of



(1) Confidentiality, RRC fuzzing (2) Availability, EMM fuzzing

Fig. 5. Effects of cross-device learning

unique perturbation sequences with vulnerabilities discovered, regardless of the security goal. Moreover, the differences between on-goal and off-goal learning are small, suggesting that we can improve the efficiency of security evaluation by updating the Q-tables maintained for different security goals with the same set of episodes.

7 Conclusions

This work explores a new direction of using AI-based techniques to assess vulnerabilities of LTE-capable Android mobile phones. We have developed LEFT, a testbed adopting three novel fuzzing methodologies, emulation-instrumented blackbox fuzzing, threat-model-aware fuzzing, and RL-guided fuzzing to assess vulnerabilities of 4G/LTE Android mobile phones. We have demonstrated that LEFT can be used to discover automatically the vulnerabilities in four COTS Android mobile phones, which may be exploited to compromise confidentiality or availability. In our future work, we plan to extend the capabilities of LEFT to discover new types of vulnerabilities in LTE networks automatically.

Acknowledgments. We acknowledge the Critical Infrastructure Resilience Institute, a Department of Homeland Security Center of Excellence, for supporting this work. We also thank anonymous reviewers for their valuable comments.

References

1. 3GPP. http://www.3gpp.org/
2. 3GPP TS 25.331 version 12.4.0 Release 12. http://www.etsi.org/deliver/etsi_ts/125300_125399/125331/12.04.00_60/ts_125331v120400p.pdf
3. Android Fragmentation, August 2015. http://opensignal.com/reports/2015/08/android-fragmentation/
4. ETSI TS 137 320 V11.2.0, February 2013. http://www.etsi.org/deliver/etsi_ts/137300_137399/137320/11.02.00_60/ts_137320v110200p.pdf

5. Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017. https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/
6. http://sysmocom.de
7. LENA: LTE-EPC Network simulAtor. http://networks.cttc.es/mobile-networks/software-tools/lena/
8. OpenAirInterface. http://www.openairinterface.org/
9. openLTE. https://github.com/osh/openlte
10. http://www.sharetechnote.com/html/Handbook_LTE_AttachReject.html
11. Wireshark. https://www.wireshark.org
12. GSMA - mobile spectrum: data demand explained. Technical report (2015). http://www.gsma.com/spectrum/wp-content/uploads/2015/06/GSMA-Data-Demand-Explained-June-2015.pdf
13. Hacked cameras, DVRs powered today's massive internet outage. Technical report (2016). https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/
14. Antonakakis, M., April, T., Bailey, M., et al.: Understanding the mirai botnet. In: Proceedings of USENIX Security Symposium (2017)
15. Beurdouche, B., et al.: A messy state of the union: taming the composite state machines of TLS. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 535–552. IEEE (2015)
16. Cao, J., Ma, M., Li, H., Zhang, Y., Luo, Z.: A survey on security aspects for LTE and LTE-A networks. IEEE Commun. Surv. Tutor. **16**(1), 283–302 (2014)
17. De Ruiter, J., Poll, E.: Protocol state fuzzing of TLS implementations. In: USENIX Security Symposium, pp. 193–206 (2015)
18. Enck, W., Traynor, P., McDaniel, P., La Porta, T.: Exploiting open functionality in SMS-capable cellular networks. In: Proceedings of the 12th ACM Conference on Computer and Communications Security (2005)
19. Gomez-Miguelez, I., Garcia-Saavedra, A., Sutton, P.D., Serrano, P., Cano, C., Leith, D.J.: srsLTE: an open-source platform for LTE evolution and experimentation. arXiv preprint arXiv:1602.04629 (2016)
20. Hussain, S.R., Chowdhury, O., Mehnaz, S., Bertino, E.: LTEInspector: a systematic approach for adversarial testing of 4G LTE. In: Proceedings of the Network and Distributed System Security Symposium (2018)
21. Johansson, W., Svensson, M., Larson, U.E., Almgren, M., Gulisano, V.: T-fuzz: model-based fuzzing for robustness testing of telecommunication protocols. In: Proceedings of the Seventh IEEE International Conference on Software Testing, Verification and Validation, pp. 323–332. IEEE (2014)
22. Jover, R.P.: Security attacks against the availability of LTE mobility networks: overview and research directions. In: Proceedings of the 16th International Symposium on Wireless Personal Multimedia Communications. IEEE (2013)
23. Jover, R.P., Lackey, J., Raghavan, A.: Enhancing the security of LTE networks against jamming attacks. EURASIP J. Inf. Secur. **2014**(7) (2014). https://jis-eurasipjournals.springeropen.com/track/pdf/10.1186/1687-417X-2014-7
24. Karandikar, A., Akhtar, N., Mehta, M.: Mobility Management in LTE Networks. In: Karandikar, A., Akhtar, N., Mehta, M. (eds.) Mobility Management in LTE Heterogeneous Networks, pp. 13–32. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-4355-0_2
25. Michau, B., Devine, C.: How to not break LTE crypto. In: Proceedings of the ANSSI Symposium sur la sécurité des Technologies de l'information et des Communications (2016)

26. Onwuzurike, L., De Cristofaro, E.: Danger is my middle name: experimenting with SSL vulnerabilities in android apps. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (2015)
27. Rupprecht, D., Dabrowski, A., Holz, T., Weippl, E., Pöpper, C.: On security research towards future mobile network generations. IEEE Commun. Surv. Tutor. (2018). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8329226
28. Rupprecht, D., Jansen, K., Pöpper, C.: Putting LTE security functions to the test: a framework to evaluate implementation correctness. In: Proceedings of USENIX Workshop on Offensive Technologies (2016)
29. Shaik, A., Borgaonkar, R., Asokan, N., Niemi, V., Seifert, J.-P.: Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In: Proceedings of the Network and Distributed System Security Symposium (2015)
30. Tsankov, P., Dashti, M.T., Basin, D.: SECFUZZ: fuzz-testing security protocols. In: Proceedings of the 7th International Workshop on Automation of Software Test, pp. 1–7. IEEE (2012)
31. Tu, G.-H., Li, Y., Peng, C., Li, C.-Y., Wang, H., Lu, S.: Control-plane protocol interactions in cellular networks. ACM SIGCOMM Comput. Commun. Rev. **44**(4), 223–234 (2015)
32. van den Broek, F., Verdult, R., de Ruiter, J.: Defeating IMSI catchers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 340–351. ACM (2015)
33. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)