Gotta Detect 'Em All: Fake Base Station and Multi-Step Attack Detection in Cellular Networks

Kazi Samin Mubasshir*, Imtiaz Karim*, Elisa Bertino
Purdue University

Abstract

Fake base stations (FBSes) pose a significant security threat by impersonating legitimate base stations (BSes). Though efforts have been made to defeat this threat, up to this day, the presence of FBSes and the multi-step attacks (MSAs) stemming from them can lead to unauthorized surveillance, interception of sensitive information, and disruption of network services. Therefore, detecting these malicious entities is crucial to ensure the security and reliability of cellular networks. In this paper, we develop FBSDetector–an effective and efficient detection solution that can reliably detect FBSes and MSAs from layer-3 network traces using machine learning (ML) at the user equipment (UE) side. To develop FBSDetector, we create FBSAD and MSAD, the *first-ever* high-quality and large-scale datasets incorporating instances of FBSes and 21 MSAs. These datasets capture the network traces in different real-world cellular network scenarios (including mobility and different attacker capabilities) incorporating legitimate BSes and FBSes. Our novel ML framework, specifically designed to detect FBSes in a multi-level approach for packet classification using stateful LSTM with attention and trace level classification and MSAs using graph learning, can effectively detect FBSes with an accuracy of 96% and a false positive rate of 2.96%, and recognize MSAs with an accuracy of 86% and a false positive rate of 3.28%. We deploy FBSDetector as a real-world solution to protect end-users through a mobile app and extensively validate it in real-world environments. Compared to the existing heuristic-based solutions that fail to detect FBSes, FBSDetector can detect FBSes in the wild in real time.

1 Introduction

The widespread adoption of cellular networks has brought about unprecedented improvements in data rates, latency, and device connectivity, resulting in a surge in the number of connected devices worldwide. In 2024, the number of mobile devices is assessed at 17.72 billion, having an estimated 4.88 billion global users, marking a 4.9% annual increase [1], and the number of mobile devices is expected to reach 18.22 billion by 2025 having 5.28 billion users [2]. Given the extensive usage and dependence on cellular networks, they become desirable targets for malicious entities. These entities try to disrupt cellular network services by exploiting different types of vulnerabilities in cellular network protocols. Of all the attacks and threat models targeting cellular networks, Fake Base Stations (FBSes), a.k.a. false base stations and rouge base stations, are among the most widespread and represent a significant threat. Due to the lack of authentication of initial broadcast messages as well as the unprotected connection setup in the bootstrapping phase [3–5], adversaries can install FBSes that can lure unsuspecting devices to connect to them and then launch sophisticated Multi-Step Attacks (MSAs) [6–15].

Motivation. The threat posed by FBSes is not new and has been around for a while, but they are still extensively used by attackers worldwide [16, 17]. The key motivations of our work are: (1) *A persistent problem.* Despite considerable efforts, FBSes remain a persistent challenge in cellular network security. They can still be deployed for attacks in 5G cellular networks, even with the use of the encrypted permanent ID (SUCI), in contrast to the unencrypted IMSI used in 4G, which the UE transmits to the base station for authentication [18]. Recent efforts have introduced certificate-based solutions and digital signatures [6, 19, 20] to address such problems. Nonetheless, these proposals are still in their infancy and would impose substantial overhead and require changes to the specifications to be applicable in a real-world setting. Moreover, the proposed certificate-based solutions make the design of roaming difficult for cellular network providers. When a user travels from one place to another place or from one country to another country, the different network providers will have to share their secret keys for authentication, which is very challenging considering the security aspects of the sharing process [21]. (2) *Billions of unprotected devices.* The 3GPP is planning to incorporate several defense mechanisms in the protocol to defend against FBSes [6] in future

* Equal contribution. The student author's name is given first.

generations. However, these defense mechanisms will still take several years to roll out in the specification and then to the implementation. Currently, billions of devices worldwide are vulnerable to FBS-based attacks, and billions of new devices released in the coming years will be vulnerable to attacks until the new protocol is implemented. These devices need to be secured through in-device solutions because replacing them is impractical and would incur huge costs. (3) *Impracticality and high cost of existing detection mechanisms.* Although efforts [3, 14, 22–37] have been undertaken to detect FBSes, they suffer from at least one of the following limitations: (A) Heuristic [32–34] and signature/rule [3, 14, 24] based solutions fail to adapt to the detection of ever-evolving attacks. (B) Some solutions depend on crowd-sourced data [31], which is impractical to scale up to a system that has to protect billions of devices. (C) Some detection solutions [25, 26, 35, 36] require installing expensive hardware; in most cases, they are proprietary. For example, CellDAM [37] requires a separate companion node near the UEs to capture the signaling messages. This is not a practical solution for protecting billions of devices. (D) Lower layer based solutions [22, 23, 27–30] cannot detect sophisticated FBSes, and are not inherently able to detect MSAs.

Recently, both Google and Apple have adopted new approaches to defeat FBSes [38]. These approaches are very promising within their scope; however, a knowledgeable and well-equipped adversary beyond their scope can still continue to operate. Therefore, it is essential to design effective and efficient solutions to detect FBSes and MSAs. In this paper, we aim to address this by developing a low-overhead, no-cost, and in-device solution that can effectively detect FBSes and MSAs that use FBSes in their threat model, from the network traces in the UEs.

A practical solution. Machine learning (ML) can be a practical in-device solution to address all the existing problems in detecting and defending against the FBSes and MSAs. On a high level, an ML-based solution has the following benefits: ❶ It can protect all the existing end-user devices vulnerable to the attacks. ❷ No change is required in the protocol. Using the network traces, especially the higher layer (layer-3) traces, the ML algorithms can determine whether there are any FBSes in the network and recognize MSAs. ❸ No additional hardware is required. ❹ ML algorithms add little overhead regarding memory and power consumption. ❺ As the attack patterns are similar worldwide, ML algorithms can detect FBSes and MSAs anywhere in the world and thus can support roaming of the device they are deployed in.

Challenges. The design of an ML-based system for the detection of FBSes, however, requires addressing several major challenges: (1) Acquiring a comprehensive and high-quality dataset encompassing a wide range of real-world cellular network scenarios. (2) Incorporating the surrounding context into consideration. (3) Capturing, representing, and learning the unique characteristics that define the MSAs. (4) Combining the predictions of layer-3 protocols. Layer-3, also known as the network layer, has two key protocols: (i) NAS (Non-Access Stratum) [39] and (ii) RRC (Radio Resource Control) [40]. These protocols operate within the control plane, each serving distinct purposes in facilitating communication between the User Equipment (UE) and the network infrastructure. To have a unified detection and improve robustness, the predictions made separately for the two protocols need to be combined. (5) Enabling real-time detection of the attacks. The framework needs to be an in-device solution that captures, processes and analyzes incoming packets promptly to detect the presence of the attacks effectively.

Our approach. To detect FBSes effectively, in this paper, we present the design, implementation, and deployment of FBSDetector, an ML-based framework for FBS detection and MSA recognition. As it is illegal to create FBSes in public areas and there are no publicly available datasets of FBS and MSA traces, we create FBSAD and MSAD, the *first comprehensive* and *high-quality* FBS and MSA datasets, respectively. To achieve this, we utilize different facilities at POWDER [41]. POWDER is a city-scale and end-to-end software-defined platform to support mobile and wireless research. The dataset created using POWDER is analogous to real-world datasets. This is ensured by including over-the-air actual packets transmitted within its spectrum between actual devices instead of simulated wire transmissions [42–46]. To tackle the second challenge, incorporating the surrounding context into detection, we design a two-step detection framework: a packet-level classification followed by a trace-level classification, ensuring both granular and contextual analysis. For MSA detection, we use graph learning–derived from our intuition that all the MSAs follow a specific pattern , and that to recognize MSAs successfully, it is necessary to capture these patterns. To obtain a unified prediction we combine predictions made on NAS and RRC packets using a weighted confidence-based fusion method. For the deployment of FBSDetector, we create a mobile app that analyzes the packet traces by running the pre-trained models in the device to detect FBSes and MSAs effectively in real time.

Experimental results. The unprocessed FBSAD and MSAD datasets have a combined size of 9.2 GB. Trained on this combined dataset, our experiments show that our FBS detection framework can detect FBS with 96% accuracy and a false positive rate (FPR) of 2.96%. Similarly, our graph learning model can detect 21 MSAs with 86% accuracy and an FPR of 3.28%. Our experiments also show that combining NAS and RRC predictions improves the performance by $1 \sim 2\%$. Furthermore, FBSDetector detects unseen Overshadow attacks with 86% accuracy. To validate FBSDetector's fidelity and evaluate its performance, memory and power consumption in real-world scenarios, we instantiate a mobile app for 4G UEs and set up FBSes and MSAs in a controlled lab environment. Using our lab setup, we spawn FBSes and run experiments with different FBS detection and MSA recognition scenarios.

Lastly, we run longer evaluations with the FBSDetector app in multiple countries and areas with varying population densities with diverse use cases. The experimental results show that compared to previous signature/heuristic-based approaches FBSDetector can detect FBSes and MSAs effectively using an average 835 KB of memory and less than 2 mW of power. Furthermore, we discuss how FBSDetector can be deployed and combined with network side defenses to create a robust ecosystem to prevent attacks in cellular networks in Section 8.

Contributions. This paper makes the following contributions:

- We develop a new framework–FBSDetector to detect FBSes and MSAs from network traces using ML. For this, we create FBSAD and MSAD, the *first-ever* large-scale, high-quality, real-world datasets containing FBS and MSA traces in different scenarios.
- We design a two-step detection framework: a packet-level classification followed by a trace-level classification, ensuring granular and contextual analysis. For the packet level classification, we design a stateful LSTM with attention utilizing stateful training and attention in parallel layers, which improves the detection accuracy and reduces the false positive rates. For MSAs, we innovate by converting the attack signatures to graphs and using a graph-based learning approach to detect the attacks. Graph learning models perform better than any other state-of-the-art model in recognizing MSAs. Moreover, even when MSAs evolve, unseen and reshaped MSAs can still be detected by this approach by using maximum overlapping sub-graphs.
- We deploy the solution in a mobile app and validate its performance in real-world setups. Compared to the available end-user FBS detection solutions, including signature-based solutions, our approach significantly improves the performance of FBS and MSA detection.

2 Background

In this section, we introduce relevant background about 4G, FBSes, MSAs and POWDER–the platform we use for dataset generation.

2.1 4G Cellular Networks

In a 4G network, cellular devices are called User Equipments (**UE**). The core network is called the Evolved Packet Core (**EPC**). Geographic locations are partitioned into hexagonal cell areas, each of which is serviced by a designated BS (**eNodeB**), which enables connectivity of UEs in that cell to the EPC. The Mobility Management Entity (**MME**) manages the connectivity and mobility of UEs in a particular tracking area (a set of cell areas).

Non-Access Stratum (NAS). In 4G, the Non-Access Stratum (NAS) [39] protocol is a layer-3 (Network Layer) protocol specified by 3GPP that serves as a functional layer between the core network and the UEs. Its primary role is to manage the communication sessions and seamlessly maintain the connections with the UE, even when the UE roams.

Radio Resource Control (RRC). The Radio Resource Control (RRC) [40] protocol is another layer-3 protocol used between the UEs and the BS. The major functions of the RRC protocol include connection establishment and release functions, broadcast of system information, radio bearer establishment, reconfiguration and release, and paging notification and release.

2.2 Fake Base Station (FBS)

An FBS is an unauthorized device an attacker uses to impersonate a legitimate BS within a cellular network. FBSes typically consist of a radio transceiver capable of broadcasting signals at legitimate BSes' frequencies. By emitting these signals, FBS creates a cell or coverage area, attracting nearby mobile devices to connect to it. With the deployed FBS, attackers carry out Multi Step Attacks (MSAs), resulting in DoS, location tracking, bidding down attacks, and traffic monitoring. Detecting FBSes can essentially stop these MSAs, because FBSes are the key stepping stones for these attacks. However, MSA detection provides fine-grained information about the attack and attacker, which is essential for forensics and defense design. In the following, we discuss an MSA done with an FBS–Tracking Area Update Reject (TAU) Reject attack [11].

TAU Reject attack. To deploy an FBS and to interrupt the existing connections between nearby user devices and legitimate BSes, the attacker would adjust the signal strength of the FBS to guarantee that it offers a much higher signal strength than the legitimate BS. Furthermore, the FBS broadcasts MCC and MNC numbers identical to the network operator of targeted subscribers to impersonate the real network operator. Once the attacker has properly configured the FBS, the attacker usually does the following steps for a FBS and TAU Reject attack: ① The FBS broadcasts its *SystemInformation* using the configured radio frequency. To overcome different UE functionalities, the FBS exploits a feature named *absolute priority-based cell reselection*. The principle of priority-based reselection is that UEs in the IDLE state should periodically monitor and try to connect to BSes operated with high-priority frequencies. Hence, even if the UE is close to a real eNodeB, operating the FBS on a frequency with the highest reselection priority would force UEs to attach to it. These priorities are defined in SIB Type number 4, 5, 6, and 7 messages broadcast by the real BSes. Using a passive attack setup, the attacker can sniff these priorities and configure the FBS accordingly. ② When a UE receives the system information of the FBS, it detects it as a new BS with a higher signal strength, and generally, when UE detects a new TA, it initiates a *TrackingAreaUpdateRequest* to the FBS. In order to trigger such a request, the FBS operates on a TAC that is different from the real BS. ③ For the TAU Reject attack Upon receiving the *TrackingAreaUpdateRequest* the FBS sends a *TrackingAreaUpdateReject* message. This attacker can utilize different EMM causes to either deny the LTE network (downgrade) or deny all network services (shown in Figure 1).

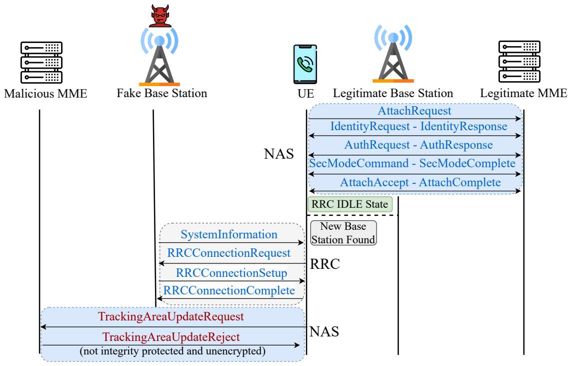


Figure 1: Communication process of an FBS with an MSA

⑤ When the FBS completes sending messages, it cuts off the cellular connections with its UEs by lowering its signal strength or shutting down the signal. After that, the affected UEs may or may not automatically re-connect to a legitimate BS (some UE's require manual restarting to connect back).

2.3 Multi-Step Attacks (MSAs)

Location tracking. IMSI catchers, StingRays, or Cell-site simulators have been widely used to collect user IMSIs and track them [25]. An attacker can also use Measurement Reports to track user location [11]. For this attack, an attacker forces a subscriber who is initially attached to a legitimate BS to connect to an FBS with a similar approach to the TAU Reject attack discussed in the previous sub-section. The subscriber UE completes the RRC connection and initiates a TAU procedure with the FBS. Next, UE enters into CONNECTED state. At this moment, the attacker turns off the first FBS and starts a second FBS. Meanwhile the UE detects it has lost synch and starts Radio Link Failure (RLF) timer. When the RLF timer expires, UE creates an RLF report and goes into IDLE mode. In this mode, UE starts cell selection procedure to attach to second FBS. The attacker sends an unprotected *UEInformationRequest* message through the second FBS and gets the *UEInformationResponse* message with the RLF report with the failure events and signal strength of neighboring BSes.

Activity monitoring. To monitor users' activity patterns, an attacker can create an FBS, connect to the devices and gather the UE capabilities. This is possible because the UE transfers its capabilities without performing authentication. An attacker can acquire the UE's core network capabilities but not the radio capabilities because they are exchanges after the RRC security setup [12]. Therefore, an FBS setup is needed to monitor all the capabilities. Other attacks such as authentication relay attack [8] allows an attacker's malicious UE to impersonate a legitimate UE and poison location history or profile network usage.

Bidding down attacks. These attacks allow an FBS to force a UE to use an older version of cellular protocols [47]. Such attacks can be carried out in different ways. During the TAU procedure the FBS can send a *TAUReject* message to force the UE to start searching for 2G and 3G networks in the area [11]. Furthermore, a recent study has found that most new devices are vulnerable to bidding down attacks, which can be divided into inter-generation and intra-generation bidding down attacks manipulating different messages and message parameters [47]. The most common way is to utilize NAS Reject messages. These messages include a specific cause that informs the UE about how to behave when rejected by the network. Since the UE is allowed to accept unprotected reject messages if it receives them before the establishment of the security context an attacker with an FBS can use the reject messages to completely disable support for the current network generation and do a downgrade attack. Another option is to use BS redirections. The BS uses *RRCRelease* to release the radio connection with a UE, e.g., if the UE switches into IDLE mode. As the release procedure can be initiated before the radio connection is secured, an FBS can cause RRC redirection from 4G to 3G. Lastly, 3G to 2G redirection is possible since the specification does not prevent a pre-authenticated RRC redirection. The recent measures by Google and Apple to disable 2G connectivity and prevent using a "null-cipher" still do not resolve the downgrade attacks to 3G and are limited to newer devices and operating systems (Android-14, iOS-17).

DoS attacks. There are numerous MSAs that an attacker can utilize to cause both short and long-term DoS. The easiest option is to use reject messages such as *AuthReject*. *RRCReject*, and *NASReject* [6, 8, 13]. This forces the UE to disconnect from the network. Another set of DoS attacks can be caused by paging channel hijacking [8]. For hijacking the paging channel, the FBS operates the same frequency band as the legitimate BS and broadcasts fake empty paging messages in the shared paging channel. One pre-requisite for the attack is to know the victims paging cycle. The FBS broadcasts paging messages with higher signal power, ensuring attack success. The victim is unable to receive legitimate paging messages from the core network.

Energy depletion and power drain. These attacks aims to make the victim UE perform expensive cryptographic operations [8, 9]. One way to achieve this is to force the UE to keep carrying out the expensive attach procedure repeatedly by sending a paging message with IMSI between two successive attach procedures. Other ways are to force victim device to release existing connection and spend energy on further cryptographic operations [9].

2.4 POWDER

POWDER (**P**latform for **O**pen **W**ireless **D**ata-driven **E**xperimental **R**esearch) [41] is a cityscale, remotely accessible, end-to-end software-defined platform funded by NSF to support mobile and wireless research and provides advances in scale, realism, diversity, flexibility, and access.

Fidelity of POWDER to the real-world. POWDER's fidelity extends beyond conventional simulation due to the following reasons: (1) POWDER provides a dedicated frequency band and real devices [41]. This access to real devices while maintaining scalability and mobility makes POWDER equivalent to real-world testbeds. (2) POWDER's fidelity is demonstrated through rigorous testing and evaluation, establishing that solutions developed using POWDER, which mirrors the complexity and challenges found in real network environments, perform effectively well in real-world scenarios [42–46]. Because of incorporating over-the-air actual packets in a dedicated spectrum instead of simulated wire transmissions, datasets created using POWDER are analogous to real-world datasets. See the detailed version of the paper [48] for a detailed overview of POWDER.

3 Overview of FBSDetector

In this section, we discuss the threat model, deployment scope, challenges, and requirements of FBSDetector.

3.1 Threat Model

For our FBS attacker threat model, we consider the adversary can impersonate the legitimate BS and thus force a victim UE to initiate a reselection with a higher signal strength than the legitimate BS. We assume the adversary can learn and mimic the legitimate values of the original BS by eavesdropping the public channels. We also assume that the adversary cannot break the cryptographic assumptions and cannot tamper with SIM cards, BSes or core network components. For instance, an attacker can only create plain-text packets but is unable to create integrity-protected or encrypted packets other than just replaying them. Furthermore, the attacker may employ various techniques to evade detection. This includes rapidly changing the parameters of the FBSes, adjusting transmission power, or adopting sophisticated obfuscation methods to mask its activities.

3.2 Deployment Scope

The current deployment scope of FBSDetector is detecting FBSes and MSAs in the context of 4G cellular networks. There are two significant reasons for this. *First,* because of the extensive infrastructure already in place, 4G networks are widely accessible to a larger portion of the population. As of 2024, 4G adoption stands at 59% among 8.6 billion SIM connections [49–52]. 4G adoption is predicted to stay above 50% until 2027, and in 2029 5G is expected to overtake 4G [49, 53]. Therefore, until 2027, many end-user devices using 4G are at risk of FBS attacks. *Second,* the platform we use for real-world dataset generation for FBS and MSAs (i.e., POWDER) supports 4G functionalities and real-world experiments can be run in POWDER for different scenarios only in 4G. POWDER currently does not support all the 5G functionalities (for instance, handover). In a related discussion, recently, the research community has uncovered a new kind of attack called *signal overshadowing attacks* [54–56] that can be an alternative for attacks without requiring an FBS. However, conducting physical signal overshadowing attacks at a large scale in the POWDER testbed presents significant challenges due to the need for sophisticated and fine-grained control over network devices and precise physical placement. Researchers aiming to explore such attacks still need specialized, controlled lab environments with complete control over the physical and radio environment to achieve the necessary precision to experiment with these attacks. Thus, it is out of scope to include overshadowing attacks in POWDER.

Despite not being trained on an overshadowing attack dataset, since being trained on layer-3 data, the FBSDetector model performs relatively well against our controlled lab environment Overshadowing [54] attack dataset. We conduct a zero-shot evaluation with this dataset and discuss the results in Section 6.3.

3.3 Challenges

The challenges in designing our ML-based framework for detecting FBSes and MSAs are:

C1. Dataset availability and quality. Acquiring a comprehensive and high-quality dataset that encompasses a wide range of real-world cellular network scenarios is a significant challenge. The dataset must include instances of legitimate BSes, FBSes, and execution traces of different MSAs. Currently, there is no such dataset publicly available. There are several reasons behind this: (1) The law prohibits the deployment of FBSes in public areas. If someone wants to deploy a FBS for research and experimentation, they must do so in a controlled RF environment. (2) Incorporating different real-world cellular network scenarios, such as handovers and mobility, is a difficult task and would require a lot of specialized hardware and other equipment and facilities.

C2. Detecting FBSes from packet traces. Unlike traditional methods where packets themselves are inherently malicious, in the case of detecting FBSes, the true nature of a packet hinges on *when* and *in which context* the packet was sent. For instance, a legitimate BS or an FBS can send the same packet with the same contents. One solution is to perform only a single trace-level classification, where the ML model takes the entire trace as input and outputs whether an FBS is present. This approach for detecting FBSes would be at a very high granularity level and thus miss a lot of context at the packet level. A well-equipped adversary can bypass the detection by keeping the trace the same as benign but changing the packet configurations and thus executing different attacks. Also, detecting unseen and reshaped attacks would not be possible at trace-level classification. In order to accurately detect FBSes and MSAs, we need to design an approach with two different granularity levels: at the single packet level and at the packet sequence or trace level. We need to classify each packet individually as suspicious or benign, serving as a preliminary filter to identify potential FBS activity. Subsequently, sequences of packets, or traces, containing packets flagged as suspicious need to undergo another contextual analysis at the trace level, which examines the order of packets and sequence patterns to discern characteristics indicative of FBS transmissions.

Sl	Attack	Attack Category	Impact
1	Authentication relay attack [8]	Activity monitoring; DoS	Complete or selective DoS; Location history poisoning; Network profiling
2	Bidding down with *AttachReject* [11]	DoS	Selective DoS
3	Paging channel hijacking attack [8]	DoS	Complete DoS
4	Location tracking via measurement reports [11]	Location tracking	Leak fine-grained location
5	Capability Hijacking [12]	DoS, Downgrading	Selective DoS and downgrading
6	Incarceration with *rrcReestablishReject* [9]	DoS	Complete DoS
7	Lullaby attack using *rrcReestablishRequest* [9]	Battery drain	Force state change, battery draining
8	Bidding down with *ServiceReject* [11]	DoS	Selective DoS
9	Mobile Network Mapping (MNmap) [12]	Device Identification	Identify devices on a mobile network
10	Energy Depletion attack [8]	Battery drain	Battery draining
11	Lullaby attack with *rrcResume* [9]	Battery drain	Force state change, battery draining
12	Stealthy Kickoff Attack [8]	DoS	Complete DoS
13	Incarceration with *rrcReject* and *rrcRelease* [9]	DoS	Complete DoS
14	IMSI catching [25]	Information Leak	Leaking sensitive information
15	NAS counter Desynch attack [9]	DoS	Complete, prolonged DoS
16	X2 signalling flood [13]	Resource waste	Waste resources for the network
17	Handover hijacking [13]	DoS, Energy Depletion	Complete DoS and battery draining
18	RRC replay attack [6]	DoS	Complete DoS
19	Lullaby attack with *rrcReconfiguration* [9]	Battery drain	Force state change, battery draining
20	Bidding down with *TAUReject* [11]	DoS	Selective DoS
21	Panic Attack [8]	Misinformation	Artificial emergency

Table 1: MSAs detected by FBSDetector

C3. Detecting MSAs. Recognizing MSAs from packet traces is even more challenging than FBS detection. These attacks have unique characteristics that define them. Moreover, MSAs often exhibit complex, evolving patterns that require careful observation to distinguish them from legitimate traffic. An adversary can improve the attacks adaptability by constantly changing to evade detection. Consequently, our approach must evolve alongside these threats. We must capture these attack characteristics and represent them effectively in a structured data format.

C4. Combining NAS and RRC predictions. Training our models separately on NAS and RRC layer packets is a necessary step due to their distinct features and characteristics. However, a challenge arises when we must consolidate these separate model predictions into a unified model.

C5. Real-time detection. Enabling real-time detection of FBSes and MSAs is a challenge due to the large volume and velocity of network traffic. The framework needs to be an in-device solution that captures, processes, and analyzes incoming packet traces promptly to detect the presence of FBSes and MSAs effectively.

3.4 Proposed Solution

In this section, we present and analyze our proposed solutions to the discussed challenges.

S1. We use different facilities available at POWDER [41] to create FBSAD and MSAD, real-world datasets to detect FBSes and MSAs. We design different networking scenarios, incorporating legitimate BSes and FBSes, and collect data from these scenarios. We also collect network traces of different MSAs that use FBSes in their threat model. Different adversaries are incorporated for both FBSes and MSAs, from less sophisticated to more sophisticated, with the ability to clone legitimate BSes and change signatures. The dataset is then processed to make it appropriate for training different ML models. This includes protocol filtering, feature extraction and feature alignment.

S2. To address the challenge of granular classification of packets and incorporating the context for detection, we propose a two-level ML-based detection framework. The approach integrates packet-level classification with trace-level classification, leveraging the strengths of machine learning at both granular and sequential analysis levels. The high-level overview of both the detection models is discussed below: (i) *Packet-Level Classification.* We perform a packet-level classification by classifying each packet individually as suspicious or benign. We leverage a stateful LSTM model with attention for this packet-level classification to model long-term dependencies that span the fixed-size sequences of the packets. We utilize the attention mechanism to learn each training sequence optimally by focusing on the parts of each sequence that affect the classification outcome the most. An essential objective this design helps us achieve is incorporating the surrounding context into consideration while giving attention to only relevant information while classifying packets. (ii) *Trace-Level Classification.* Subsequently, traces containing packets flagged as malicious or benign undergo another contextual analysis. One possible solution is to flag the trace as malicious in case one of the packets in the packet-level classification is inferred to be malicious. However, such simple heuristics lack contextual sensitivity, fail to adapt to evolving attack strategies and are not flexible. Therefore, in this stage, we employ a simple classification model to examine and classify the trace. The model examines the order of packets and sequence patterns to discern characteristics indicative of FBS transmissions.

S3. Each MSA shows a unique pattern if MSAs are represented as a directed graph. Based on this observation, we devise an innovative solution centered around graph data structures and learning techniques. We transform the packet traces into directed graphs. This graph representation captures the relationships inherent in the trace, which is ideal for detecting MSAs with complex and evolving patterns. We train a graph learning model specialized in learning complex patterns within the graph and learn the patterns of the MSAs. With the trained model, by using a maximum overlapping sub-graphs approach, we can recognize MSAs even when they are unseen or reshaped from known attacks.

S4. To combine the predictions for RRC and NAS traces, we design a weighted confidence-based fusion method, a widely used technique in the multi-sensor information fusion field [57, 58]. The weights are assigned to the best trace-level classification model of each layer. This method offers a robust means to blend the predictions for NAS and RRC layer.

S5. The development of a mobile app emerges as a pragmatic solution for deploying FBSDetector for real-time detection. Such a dedicated app would serve as an in-device solution, capable of swiftly capturing, processing, and analyzing incoming packet traces with on-device ML models, specifically tailored for detecting FBSes and MSAs. Regular model updates would ensure adaptability to new and unseen attacks.

4 Detailed Design

In this section, we discuss the detailed design of FBSDetector (see Figure 2 for an overview). On a high-level, the design of FBSDetector is divided into three components: (1) Dataset Construction, (2) ML Framework, and (3) Deployment.

4.1 Dataset Construction

Our dataset construction process is described below.

4.1.1 Dataset Generation

We create cellular networks in POWDER incorporating legitimate BSes, FBSes, and MSAs and capture the packets from all the cellular network components to generate the dataset.

Mobility. One real-world phenomenon in cellular networks is the mobility of the UEs. Signal strengths of BSes received at UEs moving from one place to another vary, causing the UEs to be handed over from one BS to another. Incorporating this scenario in the dataset is important; otherwise, a benign handover due to mobility might be interpreted as malicious. With the help of mobile endpoints available at POWDER, we incorporate mobility scenarios into our dataset.

Attacker ability. The attackers we consider have a diverse set of abilities. Based on their abilities, we rank them in five levels, level 0 being the least sophisticated and level 4 being the most sophisticated.

- **(Level 0)** Attackers only set up FBSes naively with a high signal strength.
- **(Level 1)** Attackers set up FBSes with an optimal signal strength sufficient to trigger a handover in the UEs.
- **(Level 2)** Attackers can clone all the parameters of a legitimate BS and impersonate the legitimate BS. Parameters such as the Cell ID, Mobile Network Code (MNC), Mobile Country Code (MCC), Tracking Area Code (TAC), and Physical Cell ID (PCI) can be cloned to impersonate a legitimate base station. They can also replicate radio frequency parameters like carrier frequency, bandwidth, and transmission power. Protocol-specific information such as the System Frame Number (SFN), Timing Advance (TA), Synchronization Signal Block (SSB), and Random Access Configuration may also be copied. Additionally, network-specific details like the Public Land Mobile Network (PLMN) ID and neighbor cell information can be cloned.
- **(Level 3)** Attackers use level 2 FBS and can carry out MSAs with the usual signatures.
- **(Level 4)** At the most sophisticated level, the attacker is aware of typical defenses and actively reshapes attacks to evade them. Adaptive adversaries employ two primary strategies: ① changing fields of malicious messages. In the first strategy, attackers manipulate non-critical fields—those that do not impact the success of the attack—within malicious messages to evade signature-based detection [7–9, 11, 59, 60]. For instance, an attacker can modify the cause field in reject messages or adjust optional fields in attach responses without affecting the attack's functionality [11]. Similarly, they can use different reserved values for security headers in messages to cause the same impact [39, 40].② The second strategy involves altering the temporal sequence of malicious messages to evade detection systems that rely on identifying standard message patterns. For instance, an attacker might inject multiple *IdentityRequest*, *AuthenticationRequest* messages or other extraneous protocol messages before executing an attack, creating many variant sequences that avoid detection. To find the fields and the messages that alter the temporal attack sequence but do not affect the attack success, we manually go through the specifications [59] and follow the prior works [7–9, 11, 60]. Furthermore, before deploying these adaptive reshaped attacks to POWDER, we run some of the attacks in our lab setup and manually validate the attack's effectiveness.

MSA data generation. To create MSAD, we chose several attacks that are executed in multiple steps and use FBS in their threat model. As shown in Table 1, we have selected 21 attacks, covering a wide range of practical threats, including DoS, privacy leakage, and downgrade. We implement and execute these attacks in the cellular networks in POWDER and incorporate instances of each attack in MSAD.

4.1.2 Dataset Preprocessing

In order to make FBSAD and MSAD suitable for training ML models, we pre-process them in several steps.

Protocol filtering and field extraction. Each packet in the network trace contains multiple protocol information. We focus only on NAS and RRC data and use protocol filtering to isolate the relevant packets precisely. We further extract the values of fields associated with these packets.

Dataset features for training the ML Models. After the protocol filtering, we find 119 fields in NAS layer packets and 183 fields in RRC layer packets. We use these fields as features to train our ML models (see the detailed version of the paper [48] for a detailed list of these fields).

4.1.3 Dataset Labelling

The datasets are labeled according to their reason for generation. Formally, we define FBSAD $:=< X_{\mathsf{FBSAD}}, Y_{\mathsf{FBSAD}} >$, where X_{FBSAD} are the packet features, Y_{FBSAD} are the packet labels. Then for each packet $\mathcal{P}$ we label $\overrightarrow{y}^{(i)}_{\mathsf{FBSAD}}$ as:

$$\overrightarrow{y}_{\mathsf{FBSAD}}{}^{(i)} = \begin{cases} 0 & \text{if } \overrightarrow{x}_{\mathsf{FBSAD}}{}^{(i)} \text{ is a benign packet} \\ 1 & \text{if } \overrightarrow{x}_{\mathsf{FBSAD}}{}^{(i)} \text{ is generated from the FBS} \end{cases}$$

For MSAD, we define the set of MSAs detected by

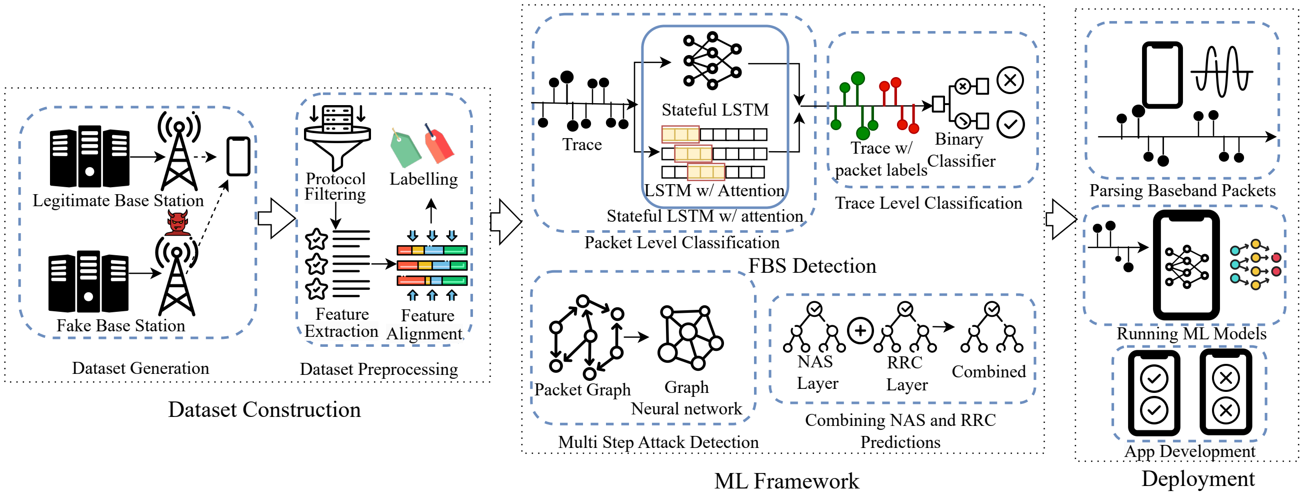


Figure 2: Overview of FBS-Detector

FBSDetector as $\mathcal{A} := \{attack_1, attack_2, \cdots, attack_k\}$ where k is the number of MSAs that can be detected and $\mathcal{L}_{\mathcal{A}} := \{0, 1, 2, \cdots, k\}$ as the set of labels for the MSAs. MSAD $:= < X_{\text{MSAD}}, Y_{\text{MSAD}} >$, where X_{MSAD} are the packet features, Y_{MSAD} are the packet labels. Then for each packet $\mathcal{P}$ we label $\vec{y}^{(i)}_{\text{MSAD}}$ as:

$$\vec{y}_{\text{MSAD}}^{\,(i)} = \begin{cases} 0 & \text{if } \vec{x}_{\text{MSAD}}^{\,(i)} \text{ is a benign packet} \\ \mathcal{L}_{\mathcal{A}}[attack_j] & \text{if } \vec{x}_{\text{MSAD}}^{\,(i)} \text{ is generated} \\ & \text{by } attack_j \in \mathcal{A} \end{cases}$$

4.2 Machine Learning Framework

In what follows, we detail the ML framework used in FBSDetector.

4.2.1 FBS Detection

We design a two-step framework for FBS detection.

Packet-level classification. In the first step, we perform a packet-level classification using a stateful LSTM model with attention. This model utilizes stateful training and attention in parallel layers, merged and fed forward for the final combined output for the packet class prediction. The statefulness models long-term dependencies that span across sequences and the attention mechanism focuses on the parts of each sequence that affect the classification outcome the most. The algorithm is shown in Algorithm 1. The algorithm begins by taking the dataset and len_{seq} hyperparameter as input. It then defines procedures for the Stateful LSTM and LSTM with Attention. The Stateful LSTM procedure initializes LSTM parameters, sets the stateful property to true to maintain state across batches, and then iterates over each timestep, feeding input x_t and previous hidden state h_{t-1} and cell state c_{t-1} into the LSTM. It then returns the final hidden state h_t, which enables state continuity across batches, ensuring that temporal dependencies are maintained even when sequences span multiple batches. The LSTM with Attention procedure initializes LSTM parameters, sets the return sequences property to true to output sequences instead of just the last timestep, processes the input sequence x_t through the LSTM, computes a context vector from an attention mechanism over the LSTM outputs, and calculates the attended output h'_t. After defining the procedures, in lines $19-22$, the main algorithm sets the input sequence x_t using len_{seq} as the sequence length, computes the output of the Stateful LSTM and LSTM with Attention modules, concatenates their outputs, and applies a dense layer to produce the final output y_t. The modules' outputs are concatenated to provide a richer input to the dense layer, facilitating more informed and potentially more accurate predictions. The Stateful LSTM preserves temporal continuity, while the Attention module highlights relevant sequence parts. The model is then trained using the calculated loss between the predicted output y and the ground truth $\hat{y}$, and the gradients are propagated back through the network for parameter updates.

Trace-level classification. In the trace-level classification phase, traces comprising packets identified as benign or malicious are subjected to additional contextual analysis. This step applies a simple binary classification model, which analyzes the temporal sequence of packets to discern distinctive characteristics of FBS transmissions. By analyzing the packet traces, this model differentiates between FBS-related activity and benign network behavior, and gives the final prediction about the presence of an FBS in the traces.

4.2.2 MSA Recognition

MSAs can be uniquely represented as directed graphs and detected using graph learning. We describe the steps to create a directed graph from the traffic dataset and graph learning in Algorithm 2. The algorithm constructs a directed graph from the packets - each node denoting a packet, each packet having a directed edge towards the next packet. Edges are labeled with their reason for generation, the same as the packet label, generated as part of a benign flow or in the path of an MSA. Then, a graph learning model learns and generalizes

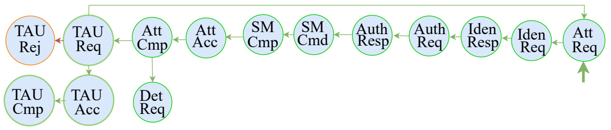


Figure 3: Graph of TAU Reject Attack

this knowledge from the graphs, and the model is finally returned. Recognizing MSAs using graph learning approaches can be formally represented as follows: let $G = (V, E)$ be the graph constructed from the packets, where each node V denotes a packet. E represents the directed edges between nodes in the flow graph. An MSA is recognized by identifying a specific path $P(G)$ corresponding to $\mathcal{A}$ within the graph G. These paths represent the packets' sequences that follow an MSA pattern. If there is an evolving and unseen attack, the attack would deviate from path $P(G)$ and follow a different path $P'(G)$. Due to the nature of the vulnerabilities exploited by the attackers, $P'(G)$ and $P(G)$ will not be completely edge-disjoint and will have overlaps. From these overlaps, we can detect evolving, unseen and reshaped attacks.

MSA detection graph example. To illustrate the process of MSA detection, we circle back to the *TAUReject* attack discussed in Section 2.2. For this attack the FBS connects the legitimate BS and injects a Reject message. The graph generated from the NAS attack traces is shown in Figure 3. The communication starts when a UE sends *AttachRequest* to the legitimate BS, which is the starting node of the graph. An incoming edge is added to the node that represents the subsequent messages. The graph enters into an attack sequence when a *TAUReject* is sent after a *TAURequest*.

4.2.3 Combining NAS and RRC Predictions

To combine the predictions from the NAS and RRC layer trace-level classification models, we leverage the Dempster–Shafer theory (DST) [61] to facilitate the fusion of predictions. We employ a weighted confidence-based fusion method, where weights are assigned to the best model of each layer based on the model performance. Let W_{NAS} and W_{RRC} represent the weights assigned to the best trace-level classification models of NAS and RRC layer, $P = \{P_{NAS}, P_{RRC}\}$ represent the predictions made by those models, and P_W is the final prediction. We assign weights W_{NAS} and W_{RRC} proportional to their support scores on the inference. The more confident model among the two models receives a higher weight, indicating a more significant influence on the combined prediction. Mathematically, this is expressed as:

$$W_{NAS} \propto \text{Support\_Score}(P_{NAS})$$
$$W_{RRC} \propto \text{Support\_Score}(P_{RRC})$$

When there's a disagreement between the models, the Dempster–Shafer theory is applied, and the prediction with higher confidence gets priority. This process is formalized as:

$$P_W = \begin{cases} P_{NAS} & \text{if } P_{NAS} = P_{RRC} \\ P_{NAS} & \text{if } P_{NAS} \neq P_{RRC} \text{ and } W_{NAS} > W_{RRC} \\ P_{RRC} & \text{if } P_{NAS} \neq P_{RRC} \text{ and } W_{NAS} < W_{RRC} \end{cases}$$

Combining the predictions in this way yields a better detection accuracy than the individual predictions.

5 Implementation

Now, we discuss the implementation details of each component of FBSDetector. We implement two distinct models: one to detect FBSes and another specifically tailored to recognize MSAs.

5.1 Dataset Construction

Incorporation of FBSes. To incorporate FBSes, we create (i) a legitimate core network and BS pair, and (ii) a fake core network and BS pair in POWDER. The legitimate BS serves UEs that the FBS can attack by spawning near it with higher signal strength. The core networks use Open5GS [62] to support handover, and the UE uses srsRAN [63] and OpenAirInterface(OAI) [64]–the two available open-source implementations. We equally use srsRAN and OAI for the dataset generation to reduce bias on a specific implementation. We introduce mobility in the dataset by using UEs with the mobile endpoints in POWDER that are mounted on the campus shuttles. Benign handovers are initiated when a shuttle with a mobile endpoint moves from the vicinity of one BS to another, and the signal quality received at the UE changes.

Implementation of handover capability in the srsUE. The UEs usually initiate handovers by sending a *TrackingAreaUpdateRequest* message. This message is not implemented in the current release of srsRAN and OAI. We implement the *TrackingAreaUpdateRequest* message in UEs and make it able to initiate a handover request by sending a *TrackingAreaUpdateRequest* message.

Implementation and execution of MSAs. We implement all the attacks listed in Table 1, including all the attacker levels in srsRAN and OAI and execute them in the experimental setup we created in POWDER. All the traces from different components are captured for both NAS and RRC layer.

Data processing. We decode the packets using tshark [65] and use a Python script for protocol filtering, packet field extraction and packet field alignment Lastly, we use *scikit-learn*'s *LabelEncoder* to encode the categorical fields into numerical representations.

Data labeling. We assign each packet a label according to the reason for its generation. The NAS layer packets (fewer in number) are labeled manually by checking each packet and assigning a label according to the reason for its generation. Labeling all the NAS layer packets takes approximately 2 hours of one-time manual effort. An automated script is used to label the RRC layer packets (which are much more numerous) in batches of intervals by detecting attack intervals in the NAS layer traces.

5.2 FBS Detection

Train-test split. We use a custom script to split our dataset for training and testing, which does roughly $80-20$ split, preserving the sequence of packets while ensuring the standard split; also, no experimental trace is cut in between.

Stateful LSTM with attention model. We utilize the TensorFlow functional API and use stateful LSTM and LSTM with attention in parallel layers. Their outputs are merged and fed forward to a common dense layer. Each network side is trained according to its architecture in this model. The Stateful LSTM model is architecturally identical to the vanilla LSTM; however, the learning algorithm has been altered to maintain the states. Both return sequences and maintain state parameters are set to true. For the LSTM with Attention model, the time-distributed dense layer is replaced by an attention layer. Return sequences are set to true, enabling the complete hidden layer sequences to be sent forward to the attention layer, where they are processed similarly to the encoder/decoder and vanilla LSTM models.

5.3 MSA Recognition

For NAS layer packets, we create a node for every unique value of the nas_eps_nas_msg_emm_type_value field, the packet name for NAS layer packets. Similarly, for the RRC layer packets, we create a node for every unique value of the lte-rrc_c1_showname field, which is the packet name for the RRC layer packets. Every packet maps to a node in the graph corresponding to its packet name. We add an edge from the node representing one packet to the node representing the next packet in the sequence and label that edge with the same label we labeled the next packet. This denotes if the transition was benign or due to an attack.

5.4 Deployment and Integration

To deploy FBSDetector, we use Mobileinsight [66] to parse the baseband traces in the mobile phones, *TensorflowLite* [67] to run the ML models, and *Flutter* [68] to build the app.

6 Evaluation

We evaluate the effectiveness of FBSDetector based on the following research questions: **RQ1.** What is the performance of each step in the FBS detection framework, namely the packet classification and trace classification? What is the performance improvement of using stateful LSTM with attention in packet-level classification? How does combining predictions of NAS and RRC trace classification further improve performance? How does graph learning improve MSA recognition performance? Why does the simple heuristic-based detection not work? **RQ2.** What is the memory and power consumption of the detection framework? How much time does the inference take? **RQ3.** Was FBSDetector deployed and tested in a real-world setup? How does it contrast with existing FBS detection solutions? **RQ4.** Is FBSDetector robust and generalizable against reshaping and unseen attacks? Also can it detect Overshadowing attacks?

Model	NAS Layer Packets				RRC Layer Packets			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
Random Forest	0.85	0.87	0.82	0.84	0.68	0.81	0.16	0.69
SVM	0.81	0.21	0.70	0.59	0.66	0.79	0.00	0.66
Decision Tree	0.86	0.89	0.81	0.82	0.75	0.84	0.53	0.76
XGBoost	0.89	0.89	0.84	0.84	0.83	0.87	0.79	0.84
k-NN	0.86	0.81	0.80	0.79	0.86	0.89	0.82	0.83
Naïve Bayes	0.53	0.87	0.35	0.58	0.80	0.08	0.52	0.37
Logistic Regression	0.50	0.68	0.69	0.53	0.74	0.85	0.50	0.71
CNN	0.86	0.39	0.57	0.52	0.84	0.67	0.78	0.66
FNN	0.79	0.85	0.68	0.73	0.80	0.88	0.84	0.78
LSTM	0.86	0.85	0.82	0.89	0.89	0.86	0.81	0.89
Stateful-LSTM w/ attention	**0.91**	**0.97**	**0.86**	**0.95**	**0.94**	**0.97**	**0.95**	**0.92**

Table 2: Performance of packet level classification for FBS detection

Experimental Setup. For training the models, we utilized a Lenovo ThinkPad T480 equipped with 32GB of memory and an Intel Core i7-8650U CPU @ 1.90GHz × 8. The operating system was Ubuntu 22.04.1 LTS (64-bit), running GNOME Version 42.2. We provide detailed information about the model hyperparameters, in Appendix Section B.

In what follows, we delve into the details and answers to those research questions.

6.1 RQ1. Packet and trace level classification

Packet and trace level classification. Our stateful-LSTM model with attention performs substantially better compared to the other packet-level models (shown in Table 2). Meanwhile, for trace-level classification, all the classical ML models perform similarly (shown in Table 3). This is expected because the heavy lifting of FBSDetector is done in the packet-level classification phase.

Performance improvement using Stateful-LSTM with attention. Stateful-LSTM with attention improves the performance of the vanilla LSTM model by 6% in NAS layer packet classification and 3% in RRC layer packet classification (shown in Table 2). Substantial enhancements are also observed in precision, recall and f1-score. Improving recall and accuracy in malicious traffic classification means the model is better at capturing a larger proportion of actual threats, reducing the chances of missing malicious activity. This is crucial for minimizing false negatives and enhancing overall detection effectiveness. Figures 6a and 6b show the distribution of the length of the FBS generated packet sequences and Figures 6c and 6d show the impact of sequence length on the detection performance. The LSTM model performs better when the input sequence length is between $9-15$ for NAS layer packets and $80-120$ for RRC layer packets. The reason is that in an FBS session, based on the communication process of the FBSes, packets exchanged between the FBSes and the UEs in the NAS and RRC layer follow a specific distribution. These distributions are captured better when the sequence lengths are set in the range that can accommodate the distribution completely, and models trained on these segments of packets that contain these patterns can sufficiently learn better about the attack.

Model	NAS Layer Trace				RRC Layer Trace			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
Logistic Regression	0.95	0.94	0.94	0.94	0.92	0.91	0.91	0.91
K-Nearest Neighbors	0.95	0.94	0.94	0.94	0.92	0.91	0.91	0.91
Decision Tree	0.95	0.94	0.94	0.94	0.92	0.91	0.91	0.91
Random Forest	0.95	0.94	0.94	0.94	0.92	0.91	0.91	0.91
Gradient Boosting	0.95	0.94	0.94	0.94	0.92	0.91	0.91	0.91
Support Vector Machine	**0.96**	**0.95**	**0.95**	**0.95**	**0.93**	**0.92**	**0.92**	**0.92**

Table 3: Performance of trace level classification

Model	Precision	Recall	F1-Score	Accuracy
Trace Level FBS Classifier (NAS)	0.96	0.95	0.95	0.95
Trace Level FBS Classifier (RRC)	0.93	0.92	0.92	0.92
Combined Trace Level Prediction	**0.96**	**0.95**	**0.96**	**0.96**

Table 4: Performance of combined trace level classification for FBS detection

Graph learning. We use GraphSAGE as our MSA recognition model since, among the graph learning models, it performs better than any other models (shown in Table 5). Though the improvement seen in accuracy might seem small (1%), a significant improvement is seen in precision (4-8%), recall (10-12%) and f1-score (8-10%).

Predictions combining. The performance after combining the predictions of NAS and RRC trace classification using the weighted confidence-based fusion method for FBS detection and MSA recognition is shown in Table 4 and Table 6, respectively. By leveraging knowledge from both layers, this method improves on all the performance metrics from the best trace-level classification model among the two and makes the system robust, especially by improving recall.

Necessity of Trace Level Classification. To evaluate the efficacy of the trace-level classification model we conduct a performance comparison between packet-level classification only and the combined approach with trace-level classification. For the packet-level classification-only approach, we assume that if at least one packet is flagged as malicious, we consider the whole trace as malicious. In Table 7, we see by enriching the analysis with aggregated data, trace-level classification enhances detection accuracy and ensures a more comprehensive approach to identifying threats.

6.2 RQ2. Overhead Analysis

Overhead Analysis of ML Models. We evaluate the overhead of the ML models used in FBSDetector based on several criteria. Figure 4a shows the time required to predict packets, which increases linearly with the number of packets. The slope of the increase is minimal, which means that the solution can scale to high throughput applications. Figure 4c shows the memory consumption and Figure 4b shows the power consumption of FBSDetector, consumed in packet processing and running the ML model on the processed packets to generate inferences. The trend of power consumption also linearly increases, with a small slope and the trend of memory consumption decreases, which can result from multiple hardware-level optimizations by the operating system. Compared to a recent approach [3], which uses an average of 4 mW power, FBSDetector uses less than 2 mW of power to detect a FBS. These results show that FBSDetector is a promising solution for deployment in real-world systems, having negligible overhead. The load testing (CPU and memory usage under different loads) for the mobile app is shown in Figure 4d.

Model	NAS				RRC			
	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
Random Forest	0.617	0.628	0.454	0.76	0.343	0.278	0.254	0.42
SVM	0.088	0.200	0.122	0.44	0.076	0.200	0.110	0.38
Decision Tree	0.342	0.402	0.356	0.66	0.076	0.200	0.110	0.38
XGBoost	0.794	0.573	0.786	0.84	0.586	0.530	0.548	0.47
k-NN	0.734	0.702	0.706	0.81	0.484	0.454	0.392	0.47
Naive Bayes	0.440	0.384	0.274	0.29	0.440	0.258	0.178	0.11
Logistic Regression	0.284	0.420	0.142	0.46	0.412	0.318	0.300	0.49
CNN	0.132	0.274	0.114	0.22	0.072	0.259	0.106	0.36
FNN	0.128	0.282	0.222	0.24	0.274	0.220	0.144	0.40
LSTM	0.150	0.320	0.200	0.49	0.128	0.164	0.130	0.30
Graph Attention Network	0.868	0.672	0.865	0.84	0.316	0.338	0.298	0.36
Graph Attention Network v2	0.842	0.864	0.857	0.83	0.504	0.411	0.421	0.42
Graph Convolutional Network	0.832	0.754	0.818	0.80	0.306	0.464	0.352	0.40
Graph Transformer	0.836	0.882	0.841	0.81	0.436	0.495	0.430	0.41
GraphSAGE	**0.872**	**0.924**	**0.883**	**0.85**	**0.633**	**0.561**	**0.557**	**0.59**

Table 5: Performance of MSA recognition from NAS and RRC layer packets

Model	Precision	Recall	F1-Score	Accuracy
GraphSAGE (NAS)	0.867	0.879	0.856	0.850
GraphSAGE (RRC)	0.611	0.480	0.52	0.590
Combined Prediction	**0.875**	**0.901**	**0.874**	**0.865**

Table 6: Performance of MSA recognition after combining predictions

Task	Prec	Rec	F1	Acc
Packet Level Classification Only	0.91	0.90	0.91	0.90
Combined w/ Trace Level Classification	**0.96**	**0.95**	**0.96**	**0.96**
Overshadow attack [54] detection (Zero Shot)	0.84	0.82	0.83	0.86

Table 7: Performance of combined packet and trace level classification for FBS detection and a zero shot detection evaluation of overshadowing attack

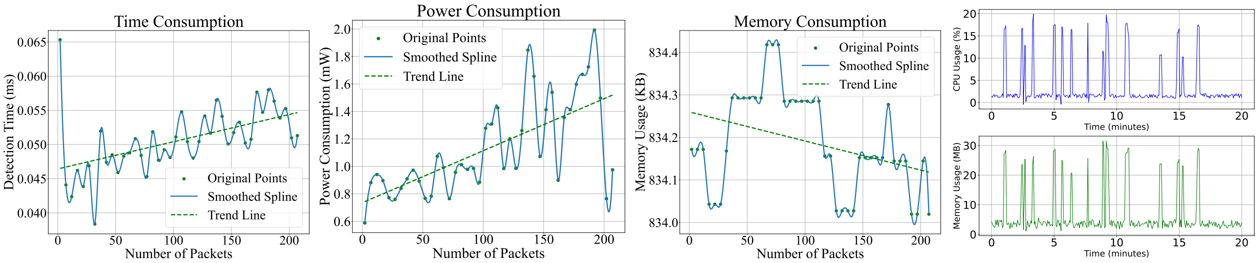
6.3 RQ3. Validation and Comparison

Real-world App validation. To validate FBSDetector app's performance against threats in the wild in real environments, we perform tests in our controlled lab environment. This is because it is illegal to deploy FBS in public places.

Lab environment: For the controlled lab environment, we create a testbed using (1) two USRP B210 [69], (2) two engineering laptops and (3) a smartphone with a Google Fi sim card with the FBSDetector app installed. We used Open5GS [62] for the core network and srsRAN [63] and OAI [64] for the BS. Following the standard approaches, we create and spawn an FBS using the laptop as the core network and the USRP B210 SDR as the BS. To test the FBS and MSA detection in different setups, we create the following scenarios: (1) lab 4G network with our own SIM card as legitimate; (2) commercial network with a Google Fi SIM card; (3) varying distance between FBS and device; (4) limited mobility in the lab. For

Solution	Supports 4G	No Change in Protocol	No Additional Hardware Required	Source Available	Detects FBS
Crocodile Hunter [26]	✓	✓	✗	✗	-
Darshak [32]	✗	-	-	-	-
Baron [19]	✓	✗	-	-	-
Phoenix [3]	✓	✓	✓	✗	-
Android IMSI Catcher [33]	✓	✓	✓	✓	✗
SnoopSnitch [34]	✓	✓	✓	✓	✗
FBSDetector	✓	✓	✓	✓	✓

Table 8: Comparison of FBSDetector with existing solutions.



(a) Time consumption (b) Power consumption (c) Memory consumption (d) Load testing of app

Figure 4: Performance overhead of FBSDetector

a thorough evaluation, we run the app for 24 hours with the Google Fi SIM card, create an FBS and run all 21 MSAs each for 5 times. For general FBS, all five times are detected. For the MSA, in a total of 105 attacks, 88 are detected as True Positive, 5 are misclassified as benign and 17 MSAs are classified into other attacks. For the whole 24-hour period, we got 6 false positives, where no attack was conducted, but the app still showed a notification. For a detailed breakdown of the results, see the detailed version of the paper [48].

Long Term Evaluation. For long-term evaluation with the FBSDetector app we ran the app for total of 7 days with different use cases such as web browsing, video streaming, calling, idle time, maps and navigation. Within this time, we ran the app in different areas with varying population densities, such as metropolitan cities and high-population events. Lastly, we ran the app in 2 different countries with local 4G connectivity providers. For stress testing, we conducted all 21 attacks 5 times within 24 hours. For the extensive longer tests, we just ran the app for 7 days with commercial SIM cards. From the packets that we saved in that period, for the stress test, it was 105,561 (NAS and RRC combined) packets within 24 hours, whereas for the longer tests, it was 326,385 (NAS and RRC combined) packets in 7 days. On the whole we found 2 alerts during the longer tests. Since we do not have any ground truth data we can not certainly discuss false positives and false negatives. However, even if we consider the 2 alerts as false-positives, compared to previous stress testing, the performance is significantly better. Therefore, we can argue in real-world usage FBSDetector would perform better in terms of False Positives.

Existing solutions comparison. We compare FBSDetector with different real-world solutions. The criteria for the comparison are: (1) The solution must not require any changes in the protocol. (2) The source of the solution must be available and maintained. (3) No additional hardware is required for operation. The comparison summary in Table 8 shows that only Android IMSI Catcher (AIMSICD) [33] and SnoopSnitch [34] satisfy all comparison criteria. Therefore, we test them in our testbed by creating an FBS in our controlled lab environment and spawning it near a mobile device with AIMSICD and SnoopSnitch installed and running. We ran the experiment several times, but neither of the solutions could detect the FBS, whereas FBSDetector detected the FBS every time. Note that, the comparison with AIMSICD and SnoopSnitch was conducted by downloading and testing them in same controlled setup used for testing our app. These apps are closest compared to our app, are developed by open-source communities, are not well maintained, and we do not have many details about their inner workings. We have contacted the developers but, at the time of the write-up, have not heard back.

Comparison with Phoenix. For a more comprehensive comparison of MSA detection with existing solutions, we find Phoenix [3] the most appropriate according to Table 8. To compare FBSDetector with Phoenix, we first implement a simple implementation of Phoenix in Python as the implementation is not publicly available. Phoenix uses three different signature representations, (1) Deterministic Finite Automata (DFA); (2) Mealy machine (MM) [70]; (3) propositional, past linear temporal (PLTL) [71]. We create and run Python scripts for all three signature representations on our attack traces (level 0-4) that Phoenix can detect. From the results shown in Table 10, we can see that FBSDetector performs significantly better than Phoenix for all the attacks.

Why heuristic/signature-based approaches fail. For further evaluating signature-based detection approaches with FBSDetector we evaluate with traces from an adaptive adversary. The adaptive adversary reshapes attack to evade detection and active employs the two techniques discussed in Section 4.1.1) level 4 attacker ability. We conduct this evaluation with Phoenix [3] as well. We chose Phoenix for several reasons: (i) similar to FBSDetector it also deploys a device-centric attack detection mechanism; (ii) it deploys sophisticated signature based schemes for MSA detection. From Table 9, we see that signature-based detection techniques used in Phoenix [3] with PLTL (the best performing signature) struggle to detect reshaped attacks like Attach Reject, IMSI Catching, and Service Reject due to their reliance on rigid, predefined rules and patterns. Attackers can reshape attacks by subtly modifying the attack behavior to avoid violating these rules, leading to misclassification or missed detections. For instance, for Attach Reject, Phoenix's signature is to detect an attack as soon as it receives an *AttachReject* message. Therefore, sending an out-of-sequence *AttachReject* with a dif-

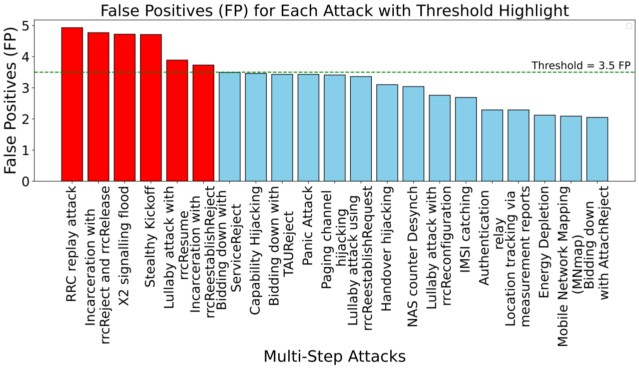


Figure 5: MSA detection FP breakdown

ferent cause field misclassifies all the attacks to this category. Similar results are for IMSI catching, Service Reject, and TAU Reject, where the signature detects attacks based on only the specific message type. Similarly, signatures where temporal orderings are included are broken by changing the sequence. For instance, Phoenix detects a Numb Attack when *AuthenticationReject* message is received without the previous NAS message being *AuthenticationResponse*. This is broken by sending an *AuthenticationRequest* message before sending the reject message. In contrast, our machine learning-based approach excels by learning complex patterns and relationships in data, enabling it to generalize across diverse attack variations.

Error Analysis. For FBS detection, level 0 and level 1 attacks both have low False Positive (FP) and False Negative (FN) rates (see the detailed version of the paper [48]). However, the FP and FN increase in levels 2-4, where an attacker clones all the parameters of the legitimate BS, and it becomes harder to detect the FBS. This is mostly due to the lack of features when an attacker clones all the parameters, and the behavior completely resembles a legitimate BS. For MSA, the analysis in Table 11 highlights that the RRC replay attack, Incarceration with *rrcReject* and *rrcRelease*, X2 signaling flood, and Stealthy Kickoff Attack are the top contributors to FP, accounting for 4.93%, 4.77%, 4.72%, and 4.71% of the total FP, respectively. Approximately 20% of the attacks are responsible for majority of the FP, which means that the overall false positives of the system will be lower on average (see Figure 5). For all these attacks, from the UE perspective, the attack behavior is precisely the same as the benign behavior of the network. For instance, for the Stealth Kick-off Attack, the attacker clones the paging message and includes the UEs IMSI. From the Layer-3 packets and features, it is difficult for the ML model to detect it and cause FP. The same goes for FN behavior, where the attack behavior closely resembles legitimate network activities, making detecting challenging due to overlapping features or insufficient distinction between benign and malicious patterns. The attacks contributing most to FN in the system include Paging Channel Hijacking and Lullaby Attack using *rrcReestablishRequest*, among others.

Zero shot detection of Overshadow attack. To generate overshadowing attack data in our controlled lab environment, we start by configuring the legitimate eNodeB with specific LTE parameters and connecting a UE to establish a baseline connection. A second SDR is then set up as a malicious transmitter. This transmitter synchronizes with the legitimate LTE network to ensure proper timing and frequency alignment. The malicious transmitter generates high-power LTE signals with the same Cell ID to overshadow legitimate signals. The success of the attack is validated by confirming that the UE receives the malicious signal instead of the legitimate one. Table 7 shows the zero shot detection capability of FBSDetector for Overshadowing attack. The attack is detected with 86% accuracy.

6.4 RQ4. Unseen and Reshaped Evaluation

To establish a benchmark and prove the robustness and generalizability of FBSDetector, we evaluate its capability to detect unseen and reshaped attacks.

Unseen attacks. FBSDetector can detect unseen attacks by leveraging anomalies in behavior that deviate from benign patterns. In cases where it encounters an attack it has not seen before, there will be a misclassification into an existing class. However, the attack would not go undetected, as the deviation from established benign behavior will still trigger an alert, ensuring that all anomalies are identified and addressed. We validate this using k-fold cross-validation. We keep one attack aside while training and then test on it. Experiments show that all the unseen attacks are classified as another type of attack (see the detailed version of the paper [48] for detailed results), proving that attacks will not go undetected. This shows the capability and robustness of FBSDetector to detect unseen attacks and to generalize.

Reshaped attacks. In case of attack reshaping, especially for attackers with more sophisticated capabilities, even if the attacker is aware of the presence of FBSDetector and reshapes the attack pattern completely to evade detection, FBSDetector can still detect it. This reshaped behavior deviates from benign behavior and overlaps with the original attack that was reshaped, and FBSDetector can detect it from this deviation and overlap. To test FBSDetector's capability to detect reshaped attacks properly, we create additional reshaped data following level 4 of attacker capability (section 4.1.1) and evaluate FBSDetector's performance. Experiments show that all the reshaped attack packets are classified as original attacks (see the detailed version of the paper [48] for details).

7 Related Work

Several approaches have been proposed to address the challenge of detecting FBSes. Recent efforts have introduced certificate-based solutions and digital signatures [4, 6, 19, 20, 22, 72] for BS authentication. However, these techniques require modifications to specifications, require huge infrastructure changes, add overhead, and are not able to defend the billions of devices currently in the market. This makes FBSDetector highly suitable for defending a wide variety of attacks. The other approaches are based on several simple heuristics and signatures [3, 14, 32–34]. As shown by the extensive experiments in this paper, heuristic based approaches are not well suited to defend against an adaptive adversary. The approaches proposed in [35, 36] require installing expensive hardware and in most cases they are proprietary. The techniques in [31] depends on crowd-sourced data, which is not a practical solution for scaling up. ML based efforts are effective in detecting attacks and anomalies from traces [30]. However, they work on small datasets generated in simulated environments that lack diversity; also because they work only in the data plane, they cannot detect MSAs. Recently, researchers have used simulation models to analyze fake base station attacks on a large scale [73], but they fail to capture different real-world scenarios. Previous works have shown that information related to the connection between a UE and a BS can be used to reason about the authenticity of the BS [23–25, 27–29, 74].

8 Discussion

Applicability to 5G. To the best of our knowledge, no open-source protocol stack for the standalone 5G core network supports handover, which is a prerequisite for creating a real-world FBS and MSA dataset. Therefore, we leave the detection of FBSes and MSAs in 5G cellular networks with FBSDetector as a future work. However, we believe that, based on 4G, our approach is equally applicable to 5G because most of the layer 3 procedures are unchanged from 4G. Thus, the ML model designs will remain the same when porting FBSDetector to 5G.

Deployment. If FBSDetector were deployed in a real-world setting, we envision the model will be periodically retrained to incorporate new attacks, with updates pushed promptly to user devices via app updates. Since we collect NAS and RRC traces, which can contain sensitive information, data collection is enabled via user consent, ensuring transparency and privacy compliance. The FBSDetector app is built on top of MobileInsight [66]. Therefore, the requirements for running MobileInsight apply to our solution as well, which includes rooting the phone for most smartphone models (for more details, we refer to the MobileInsight website). Apart from this deployment scenario recently, we have been in discussion with a commercial connectivity vendor about applying FBSDetector on top of the baseband directly, without requiring the phone to be rooted.

Defense against the detected FBSes. For defense in our mobile app, the user is notified immediately for an FBS upon detection. Additionally the user has the capability to switch to another cell, and add the current cell to a temporary block list. For instance, if the FBSDetector app detects an IMSI-catcher after receiving an *IdentityRequest* in the sequence, it turns the radio off to stop leaking sensitive information (with permission from the user). Another advanced solution can be to design and integrate a learning-based (e.g., reinforcement learning) decision-making agent directly into the baseband that can not only detect but also recover from the attack in an automated way.

UE vs network side defense. Defense against FBS can be deployed at the UE or network sides. These network side solutions are designed for traditional cellular architecture [24, 30] and for the emerging O-RAN architecture [14, 22]. There are two critical limitations with network-based deployments: (i) a network-level solution might be able to detect that a cell is affected by an FBS but would be unable to take any necessary action to protect user privacy and prevent attacks. For instance, FBSDetector prevents sensitive information leakage upon detecting attacks by turning off the radio; (ii) certain MSAs necessarily cannot be observed by the network operators, which is observable only from the device vantage point. For instance, after an FBS has been connected to the device, it is not possible for the network operator to uncover the type of attack. Because of these reasons FBSDetector opts for an UE-centric solution. However, device-centric solutions also have limitations, such as requiring root access or sensitive permissions, especially on Apple devices. In the future, this can be resolved by deploying the UE-centric solutions on top of the baseband directly without requiring the phone to be rooted. Overall, we conclude that both network and UE side defense and detection mechanisms would ultimately be needed to defend against FBS attacks and create a robust ecosystem to prevent attacks altogether.

Implication of FP and FN. FBSDetector detects FBSes with 96% accuracy. However, FBS detection is a hard problem not because of the difficulty of detecting attacks but because it is hard to prevent false positives, given that attacks are rare. Therefore, a system with high FP might not be suitable for general use. FBSDetector has a 2.96% FP rate, and in the FBSDetector app stress testing, we found 6 FP instances out of 110 instances; in longer tests, we found 2 alerts. The ambition of FBSDetector is to bring FBS detection to the masses; however, currently, it is more suitable for safety-critical UEs and communication for people with sensitive information, where security is prioritized. In the future, we plan to improve the FP rate further by incorporating lower-layer features and pushing direct app updates.

9 Conclusion And Future Work

In this paper, we present FBSDetector, an ML-based FBS and MSA detection system for cellular networks, which leverages network traces at Layer 3. To train the ML models, we have created FBSAD and MSAD, the *first-ever* large-scale real-world datasets. We deploy FBSDetector on a mobile app that effectively detects FBSes and MSAs in all the tested real-world scenarios.

Future work. In the future, we will port FBSDetector to 5G, support overshadow attacks and focus on detecting FBSes within the emerging Open Radio Access Network (ORAN) environment, employing advanced machine learning algorithms through xApps. We will also investigate different defense mechanisms to effectively stop an attack once detected by FBSDetector.

10 Acknowledgements

This work is supported by NSF Grant No. 2112471. We thank the POWDER team for their support in utilizing the platform and Mir Imtiaz Mostafiz for his feedback on the paper.

Ethics considerations

All the experiments in this paper followed the ethics consideration policies. The experiments were done in an isolated lab setup where all the victim UE's belong to us. Furthermore, we use a shielding box to prevent our USRPs from interfering with the commercial networks' licensed spectrum, following ethical guidelines. We also use logical precautions to make sure only our UE IMSIs are connected to the attack setup and we do not affect any other UEs. These ethical steps and precautions are in line with all the previous research on cellular network security.

Compliance with the open science policy

In compliance with the open science policy, we release the datasets, the models and the Android App for general users to use and foster further research[1].

References

[1] *HOW MANY PEOPLE HAVE SMARTPHONES IN 2024?*. https://www.oberlo.com/statistics/how-many-people-have-smartphones.

[2] *Forecast number of mobile devices worldwide from 2020 to 2025 (in billions)**. https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/.

[3] Mitziu Echeverria, Zeeshan Ahmed, Bincheng Wang, M. Fareed Arif, Syed Rafiul Hussain, and Omar Chowdhury. PHOENIX: device-centric cellular network protocol monitoring using runtime verification. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[4] Syed Rafiul Hussain, Mitziu Echeverria, Ankush Singla, Omar Chowdhury, and Elisa Bertino. Insecure connection bootstrapping in cellular networks: The root of all evil. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Yomna Nasser. Pre-Authentication messages as a common root cause of cell network attacks. San Francisco, CA, January 2020. USENIX Association.

[6] *3GPP TS 33.809 Study on 5G security enhancements against False Base Stations (FBS): Certificate based solution for Protecting System Information Messages with Digital Signature in an NPN*. https://www.3gpp.org/ftp/TSG_SA/WG3_Security/TSGS3_100Bis-e/Docs/S3-202717.zip.

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

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

[9] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5greasoner: A property-directed security and privacy analysis framework for 5g cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 669–684, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy attacks to the 4g and 5g cellular paging protocols using side channel information. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[11] Altaf Shaik, Jean-Pierre Seifert, Ravishankar Borgaonkar, N. Asokan, and Valtteri Niemi. Practical attacks against privacy and availability in 4g/lte mobile communication systems. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[12] Altaf Shaik, Ravishankar Borgaonkar, Shinjo Park, and Jean-Pierre Seifert. New vulnerabilities in 4g and 5g cellular access network protocols: exposing device capabilities. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 221–231, 2019.

[13] Altaf Shaik, Ravishankar Borgaonkar, Shinjo Park, and Jean-Pierre Seifert. On the impact of rogue base stations in 4g/lte self organizing networks. In *Proceedings of*

[1] Datasets, codebase and models for FBSDetector are publicly available at https://zenodo.org/records/14720824

the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks, WiSec '18, page 75–86, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Haohuang Wen, Phillip Porras, Vinod Yegneswaran, Ashish Gehani, and Zhiqiang Lin. 5g-spector: An o-ran compliant layer-3 cellular attack detection service. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS'24)*, San Diego, CA, February 2024.

[15] Martin Kotuliak, Simon Erni, Patrick Leu, Marc Röschlin, and Srdjan Čapkun. {LTrack}: Stealthy tracking of mobile phones in {LTE}. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1291–1306, 2022.

[16] *DHS confirms it has detected evidence of mobile snooping devices around DC.* https://www.cnn.com/2018/04/03/politics/dhs-stingrays-washington-dc/index.html.

[17] *Gang Of Drivers Caught Using Stingrays To Send Fake Links And Steal Cash.* https://thainewsroom.com/2023/05/25/gang-of-drivers-caught-using-stingrays.

[18] Merlin Chlosta, David Rupprecht, Christina Pöpper, and Thorsten Holz. 5g suci-catchers: Still catching them all? In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 359–364, 2021.

[19] Alessandro Lotto, Vaibhav Singh, Bhaskar Ramasubramanian, Alessandro Brighente, Mauro Conti, and Radha Poovendran. Baron: Base-station authentication through core network for mobility management in 5g networks. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '23, page 133–144, New York, NY, USA, 2023. Association for Computing Machinery.

[20] Ankush Singla, Rouzbeh Behnia, Syed Rafiul Hussain, Attila Yavuz, and Elisa Bertino. Look before you leap: Secure connection bootstrapping for 5g networks to defend against fake base-stations. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 501–515, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Gabriel K. Gegenhuber, Wilfried Mayer, Edgar R. Weippl, and Adrian Dabrowski. Mobileatlas: Geographically decoupled measurements in cellular networks for security and privacy research. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[22] Guillem Reus-Muns, Dheryta Jaisinghani, Kunal Sankhe, and Kaushik R. Chowdhury. Trust in 5g open rans through machine learning: Rf fingerprinting on the powder pawr platform. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[23] Leyli Karaçay, Zeki Bilgin, Ayşe Bilge Gündüz, Pinar Çomak, Emrah Tomur, Elif Ustundag Soykan, Utku Gülen, and Ferhat Karakoç. A network-based positioning method to locate false base stations. *IEEE Access*, 9:111368–111382, 2021.

[24] Prajwol Kumar Nakarmi, Mehmet Akif Ersoy, Elif Ustundag Soykan, and Karl Norrman. Murat: Multi-rat false base station detector. *CoRR*, abs/2102.08780, 2021.

[25] Adrian Dabrowski, Nicola Pianta, Thomas Klepp, Martin Mulazzani, and Edgar Weippl. Imsi-catch me if you can: Imsi-catcher-catchers. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, page 246–255, New York, NY, USA, 2014. Association for Computing Machinery.

[26] Cooper Quintin. Detecting fake 4g LTE base stations in real time. USENIX Association, February 2021.

[27] Arslan Ali and Georg Fischer. Enabling fake base station detection through sample-based higher order noise statistics. In *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pages 695–700, 2019.

[28] Hamad Alrashede and Riaz Ahmed Shaikh. Imsi catcher detection method for cellular networks. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–6, 2019.

[29] Ke-Wen Huang and Huiming Wang. Identifying the fake base station: A location based approach. *IEEE Communications Letters*, 22:1604–1607, 2018.

[30] Prajwol Kumar Nakarmi, Jakob Sternby, and Ikram Ullah. Applying machine learning on rsrp-based features for false base station detection. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[31] Zhenhua Li, Weiwei Wang, Christo Wilson, Jian Chen, Chen Qian, Taeho Jung, Lan Zhang, Kebin Liu, Xiangyang Li, and Yunhao Liu. Fbs-radar: Uncovering fake base stations at scale in the wild. *Proceedings 2017 Network and Distributed System Security Symposium*, 2017.

[32] *Darshak.* https://github.com/darshakframework/darshak.

[33] *The Android-IMSI-Catcher-Detector (short: AIMSICD).* http://www.tea-after-twelve.com/about-us/our-authors/aimsicd.

[34] *SnoopSnitch.* https://play.google.com/store/apps/details?id=de.srlabs.snoopsnitch&hl=en_US&gl=US.

[35] *OVERWATCH IMSI CATCHER DETECTION SERVICES.* https://comsecllc.com/comsec-llc-adds-overwatch.

[36] *SeaGlass: City-Wide IMSI-Catcher Detection.* .https://news.ycombinator.com/item?id=27173717.

[37] Zhaowei Tan, Jinghao Zhao, Boyan Ding, and Songwu Lu. CellDAM: User-Space, rootless detection and mitigation for 5g data plane. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1601–1619, Boston, MA, April 2023. USENIX Association.

[38] *Apple and Google Are Introducing New Ways to Defeat Cell Site Simulators, But Is it Enough?.* https://www.eff.org/deeplinks/2023/09/apple-and-google-are-introducing-new-ways-defeat-cell-site-simulators-it-enough.

[39] *Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3.* https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072.

[40] *Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification.* https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2440.

[41] Joe Breen, Andrew Buffmire, Jonathon Duerig, Kevin Dutt, Eric Eide, Mike Hibler, David Johnson, Sneha Kumar Kasera, Earl Lewis, Dustin Maas, Alex Orange, Neal Patwari, Daniel Reading, Robert Ricci, David Schurig, Leigh B. Stoller, Jacobus Van der Merwe, Kirk Webb, and Gary Wong. Powder: Platform for open wireless data-driven experimental research. In *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, WiNTECH'20, page 17–24, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Sergei Chuprov, Leon Reznik, Antoun Obied, and Srujan Shetty. How degrading network conditions influence machine learning end systems performance? In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2022.

[43] David Johnson, Dustin Maas, and Jacobus Van Der Merwe. Nexran: Closed-loop ran slicing in powder -a top-to-bottom open-source open-ran use case. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization*, WiNTECH '21, page 17–23, New York, NY, USA, 2021. Association for Computing Machinery.

[44] Jose Monterroso, Jacobus Van der Merwe, Kirk Webb, and Gary Wong. Towards using the powder platform for rf propagation validation. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021.

[45] Kirk Webb, Sneha Kumar Kasera, Neal Patwari, and Jacobus Van der Merwe. Wimatch: Wireless resource matchmaking. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2021.

[46] Zhenghao Zhang. Zcnet: Achieving high capacity in low power wide area networks. In *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 702–710, 2020.

[47] Bedran Karakoc, Nils Fürste, David Rupprecht, and Katharina Kohls. Never let me down again: Bidding-down attacks and mitigations in 5g and 4g. 2023.

[48] Kazi Samin Mubasshir, Imtiaz Karim, and Elisa Bertino. Gotta detect 'em all: Fake base station and multi-step attack detection in cellular networks, 2025.

[49] *The Mobile Economy 2024.* https://www.gsma.com/mobileeconomy/wp-content/uploads/2023/03/270223-The-Mobile-Economy-2024.pdf.

[50] *5G - 4G-5G Subscribers March 2022 – Quarterly update.* https://gsacom.com/paper/4g-5g-subscribers-march-2022-quarterly-update/.

[51] *Mobile subscriptions outlook.* https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-subscriptions-outlook.

[52] *Tracking the 4G Decade.* https://blog.telegeography.com/tracking-the-4g-decade.

[53] *Number of LTE subscriptions worldwide from 2018 to 2023 (in billions)**. https://www.statista.com/statistics/206615.

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

[55] Simon Erni, Martin Kotuliak, Patrick Leu, Marc Roeschlin, and Srdjan Capkun. Adaptover: adaptive overshadowing attacks in cellular networks. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 743–755, 2022.

[56] Norbert Ludant and Guevara Noubir. Sigunder: a stealthy 5g low power attack and defenses. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 250–260, New York, NY, USA, 2021. Association for Computing Machinery.

[57] Qinli Zhang, Liangdong Qu, and Zhaowen Li. Attribute reduction based on d-s evidence theory in a hybrid information system. *International Journal of Approximate Reasoning*, 148:202–234, 2022.

[58] Yongchuan Tang, Xu Zhang, Ying Zhou, Yubo Huang, and Deyun Zhou. A new correlation belief function in dempster-shafer evidence theory and its application in classification. *Scientific Reports*, 13(1):7609, May 2023.

[59] *3GPP*. https://www.3gpp.org/.

[60] Syed Rafiul Hussain, Imtiaz Karim, Abdullah Al Ishtiaq, Omar Chowdhury, and Elisa Bertino. Noncompliance as deviant behavior: An automated black-box noncompliance checker for 4g lte cellular devices. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1082–1099, New York, NY, USA, 2021. Association for Computing Machinery.

[61] *Dempster–Shafer theory*. https://en.wikipedia.org/wiki/Dempster%E2%80%93Shafer_theory.

[62] *Open5GS*. https://github.com/open5gs/open5gs.

[63] *srsRAN*. https://github.com/srsran/srsRAN.

[64] *OpenAirInterface*. https://gitlab.eurecom.fr/oai/.

[65] *tshark*. https://www.wireshark.org/docs/man-pages/tshark.html.

[66] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, page 202–215, New York, NY, USA, 2016. Association for Computing Machinery.

[67] *tensorflow-lite*. https://www.tensorflow.org/lite/guide/inference?utm_campaign=Thoughts%20on%20HCI%20and%20Applied%20AI%20&utm_medium=email&utm_source=Revue%20newsletter.

[68] *flutter*. https://docs.flutter.dev/.

[69] *USRP B210 SDR Kit - Dual Channel Transceiver (70 MHz - 6GHz) - Ettus Research*. https://www.ettus.com/all-products/ub210-kit/.

[70] George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.

[71] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977.

[72] Ankush Singla, Syed Rafiul Hussain, Omar Chowdhury, Elisa Bertino, and Ninghui Li. Protecting the 4g and 5g cellular paging protocols against security and privacy attacks. *Proceedings on Privacy Enhancing Technologies*, 2020:126 – 142, 2020.

[73] Thijs Heijligenberg, David Rupprecht, and Katharina Kohls. The attacks aren't alright: Large-scale simulation of fake base station attacks and detections. In *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*, CSET '24, page 54–64, New York, NY, USA, 2024. Association for Computing Machinery.

[74] Adrian Dabrowski, Georg Petzl, and Edgar R. Weippl. The messenger shoots back: Network operator based imsi catcher detection. In *International Symposium on Recent Advances in Intrusion Detection*, 2016.

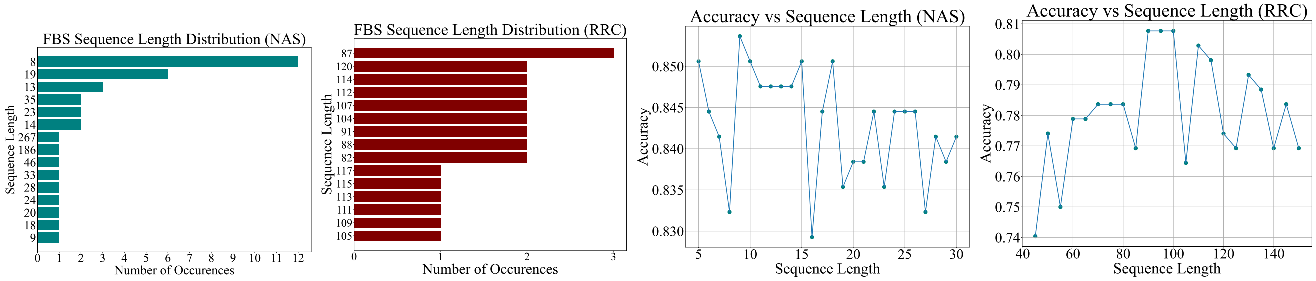
A Algorithms for FBS and MSA Detection

The FBS detection algorithm is described in Algorithm 1, and the MSA recognition algorithm is described in Algorithm 2.

B Experimental Setup

B.1 Model Hyperparameters.

Stateful LSTM w/ attention. The Stateful LSTM w/ attention consists of two parallel *LSTM* layers with 64 units and a *sigmoid* activation function configured to return sequences.



(a) NAS sequence length (b) RRC sequence length (c) Accuracy in NAS (d) Accuracy in RRC

Figure 6: Distribution and impact of NAS and RRC sequence length in FBS detection

Attack Type	System	Benign	Attach Reject	IMSI Catching	Service Reject	TAU Reject	Measurement Report	Paging with IMSI	Auth Failure	Numb Attack
Attach Reject	FBSDetector	9.35	**67.27**	1.24	6.23	6.20	0.81	2.71	4.64	1.56
	Phoenix	9.12	2.14	7.39	3.66	3.48	8.24	3.03	18.21	**44.72**
IMSI Catching	FBSDetector	3.83	2.45	**47.49**	15.97	7.53	4.14	5.76	0.42	0.66
	Phoenix	4.16	**26.26**	5.71	16.16	21.74	6.62	3.88	14.32	1.15
Service Reject	FBSDetector	1.44	14.18	4.26	**51.77**	10.30	2.14	11.49	7.32	8.79
	Phoenix	8.67	8.82	5.38	7.14	4.74	0.26	**51.95**	6.93	6.12
TAU Reject	FBSDetector	1.35	2.88	8.37	1.13	**78.34**	3.90	3.19	0.09	0.71
	Phoenix	9.28	5.98	1.84	**60.22**	5.52	0.15	1.61	3.89	11.49
Measurement Report	FBSDetector	3.02	0.51	11.81	0.52	1.74	**69.78**	8.03	3.31	1.24
	Phoenix	2.67	17.92	3.54	2.48	17.39	10.72	7.59	7.39	**30.29**
Paging with IMSI	FBSDetector	6.55	4.69	10.32	1.06	9.54	3.76	**50.92**	3.94	9.18
	Phoenix	4.43	1.68	0.71	0.52	4.76	4.23	23.14	11.5	**49.03**
Authentication Failure	FBSDetector	6.97	0.54	6.9	0.19	0.99	0.18	1.12	**72.32**	10.79
	Phoenix	5.01	**63.20**	0.34	1.75	1.47	8.07	5.81	11.17	3.18
Numb Attack	FBSDetector	9.5	9.13	10.17	2.07	13.07	6.26	0.5	5.13	**44.16**
	Phoenix	7.21	0.63	9.45	6.46	12.63	**28.34**	4.1	14.23	16.95

Table 9: Cross-validation comparison with Phoenix [3] for an adaptive adversary. The numbers represent percentage of the attack packets being classified as benign and different other attack packets. FBSDetector accurately classifies majority of the packets to it original attack (numbers shown in bold) whereas Phoenix misclassifies to other attacks (numbers shown in red).

Algorithm 1 Stateful LSTM w/ Attention

```
Input: Labeled dataset: FBSAD, hyperparameter: $len_{seq}$
Output: Classified traces indicating FBS activity
procedure StatefulLSTM($x_t$)
    Initialize LSTM parameters $\theta_s$
    Set stateful = true, return_sequences = true
    for each timestep $t$ do
        $h_t, c_t \leftarrow \text{LSTM}(x_t, h_{t-1}, c_{t-1}; \theta_s)$
    end for
    return $h_t$
end procedure
procedure LSTMwithAttention($x_t$)
    Initialize LSTM parameters $\theta_a$
    Set return_sequences = true
    $H \leftarrow \text{LSTM}(x_t; \theta_a)$
    $c_t \leftarrow$ context vector from attention mechanism over $H$
    $h'_t \leftarrow \tanh(W_c[c_t; H_t] + b_c)$
    return $h'_t$
end procedure
$x_t \leftarrow$ input(sequence_length = $len_{seq}$)
$h_t^{\text{stateful}} \leftarrow \text{StatefulLSTM}(x_t)$
$h_t^{\text{attention}} \leftarrow \text{LSTMwithAttention}(x_t)$
$y_t = \text{Dense}(\text{concat}(h_t^{\text{stateful}}, h_t^{\text{attention}}))$
Train model on loss $\mathcal{L}(y, \hat{y})$, propagate back
```

Algorithm 2 MSA Recognition Model

```
Input: Labeled dataset: MSAD
Output: Graph Model (GM)
procedure Graph Learning
    Variables: Graph $G(V, E)$
    Create start node $V_1$ with the first packet $p_1$
    for each subsequent packet $p_i$ in MSAD do
        if $V_p$ not in $G$ then
            Create a node $V_p$
        end if
        Add an incoming edge $E_p$ from $V_{p-1}$ to $V_p$
        Label $E_p$ with $L_p$, the Label for Packet $p$
    end for
    $GM = \text{train}(G)$
end procedure
return GM
```

One additional attention layer is added to the LSTM w/ attention and the *stateful* hyperparameter is set to true for the stateful LSTM. The subsequent layer after these parallel layers are concatenated is a dense layer with a single unit and *sigmoid* activation. For optimization, the *stochastic gradient descent (sgd)* was chosen, with a *mean squared error (mse)* loss function. The model's performance is assessed using *accuracy* as the main metric, complemented by the inclusion of a custom metric, *false positives*, to evaluate its classification capabilities further.

GraphSAGE. The graph model features a single *SAGEConv* layer. The *SAGEConv* layer utilizes 2 attention heads to capture graph-based relationships. The model's architecture is encapsulated within a *PyTorch* Module, with the forward function defining the flow of information through the single layer. Logarithmic *softmax* is employed for the final layer's output activation.

Other graph models. We used the same configurations as the *GraphSAGE* model for the other graph models with their own convolutional layer. For example, for the Graph Attention Network, we used *GATConv*, and for the Graph Convolutional Network, we used *GCNConv*.

Other classification models. The Random Forest Classifier and the Decision Tree Classifier were configured with a Gini criterion, a maximum depth of 3. For the XGBoost Classifier, Support Vector Classifier (SVC), K-Nearest-Neighbors (KNN) Classifier, Gaussian Naive Bayes and Logistic Regression we adopted default configurations.

Convolutional Neural Network (CNN). The CNN architecture comprises two *Conv1D* layers with 32 and 64 filters, respectively, followed by a *ReLU* activation function. *MaxPooling1D* layers with a pool size of 2 were inserted after each convolutional layer to downsample the spatial dimensions. A *GlobalAveragePooling1D* layer was then employed to aggregate the spatial information across the entire sequence. Subsequently, two Dense layers were added with 64 units and *ReLU* activation in the first, and a single unit with a *sigmoid* activation in the final layer for classification. The model was compiled using the Adam optimizer, binary cross-entropy loss function, and accuracy as the metric for performance evaluation.

Feedforward Neural Network (FNN). The FNN architecture consists of three Dense layers, with the first two layers containing 64 units each and utilizing the *ReLU* activation function. The final dense layer, with a single unit and a *sigmoid* activation function, is employed for classification. The model was compiled using the *Adam* optimizer, *binary cross-entropy* as the loss function, and *accuracy* as the metric for assessing its performance.

B.2 Impact of sequence length

The distribution of the length of the FBS generated packet sequences and the impact of sequence length on the detection performance is shown in Figure 6

Attack	Phoenix									FBSDetector		
	DFA			MM			PLTL			Acc	Prec	Rec
	Acc	Prec	rec	Acc	Prec	Rec	Acc	Prec	Rec			
Attach Reject	0.487	0.35	0.799	0.89	0.86	0.79	0.868	0.70	0.767	**0.95**	**0.97**	**0.95**
IMSI Catching	0.667	0.538	0.876	0.785	0.79	0.858	0.798	0.81	0.797	**0.98**	**0.94**	**0.97**
Service Reject	0.712	0.704	0.721	0.797	0.725	0.753	0.871	0.81	0.844	**0.95**	**0.96**	**0.93**
TAU Reject	0.627	0.95	0.756	0.763	0.865	0.715	0.789	0.803	0.751	**0.94**	**0.95**	**0.92**
Measurement Report	0.445	0.434	0.456	0.766	0.766	0.845	0.878	0.864	0.871	**0.97**	**0.95**	**0.97**
Paging with IMSI	0.574	0.634	0.918	0.783	0.765	0.81	0.84	0.822	0.786	**0.94**	**0.96**	**0.95**
Authentication Failure	0.802	0.671	0.897	0.805	0.79	0.749	0.849	0.788	0.863	**0.98**	**0.96**	**0.95**
Numb Attack	0.817	0.811	0.799	0.846	0.711	0.818	0.732	0.833	0.722	**0.97**	**0.99**	**0.95**

Table 10: Comparison between FBSDetector and Phoenix

Sl	**Attack**	**TP**	**TN**	**FP**	**FN**
1	Authentication relay attack	46.08	48.49	2.29	3.14
2	Bidding down with *AttachReject*	52.2	42.56	2.05	3.19
3	Paging channel hijacking attack	51.18	38.6	3.41	6.81
4	Location tracking via measurement reports	51.93	41.52	2.29	4.26
5	Capability Hijacking	50.62	40.58	3.46	5.34
6	Incarceration with *rrcReestablishReject*	49.65	42.81	3.73	3.81
7	Lullaby attack using *rrcReestablishRequest*	44.16	46.36	3.36	6.12
8	Bidding down with *ServiceReject*	48.1	44.57	3.49	3.84
9	Mobile Network Mapping (MNmap)	53.05	40.99	2.09	3.87
10	Energy Depletion attack	52.44	40.95	2.12	4.49
11	Lullaby attack with *rrcResume*	44.79	46.85	3.89	4.47
12	Stealthy Kickoff Attack	52.71	37.84	4.71	4.74
13	Incarceration with *rrcReject* and *rrcRelease*	48.01	42.84	4.77	4.38
14	IMSI catching	39.17	53.36	2.69	4.78
15	NAS counter Desynch attack	49.81	42.49	3.04	4.66
16	X2 signalling flood	42.75	49.46	4.72	3.07
17	Handover hijacking	40.2	51.8	3.1	4.9
18	RRC replay attack	44.16	46.89	4.93	4.02
19	Lullaby attack with *rrcReconfiguration*	47.24	45.9	2.76	4.1
20	Bidding down with *TAUReject*	50.52	42.27	3.43	3.78
21	Panic Attack	50.52	42.27	3.43	3.78

Table 11: MSAs detection performance breakdown (in percentage)

C Comparison with existing solutions

The comparison between FBSDetector and Phoenix [3] is shown in Table 10.