USENIX Security '25 Artifact Appendix: SNI5GECT: A Practical Approach to Inject aNRchy into 5G NR

Shijie Luo, Matheus E. Garbelini, Sudipta Chattopadhyay, and Jianying Zhou
Singapore University of Technology and Design

A Artifact Appendix

A.1 Abstract

As part of our commitment to Open Science, we provide the full implementation of all components described in the SNI5GECT paper. This includes source code, binaries, and scripts necessary to evaluate 5G over-the-air sniffing and message injection using a standard PC (x86_64) running Ubuntu 22.04. Given the over-the-air nature of our evaluation, which relies on specific 5G target devices and a USRP Software Defined Radio (SDR), we also offer remote access to a machine equipped with this setup. Access is granted via SSH using a private key. In addition to the binaries, we provide the evaluation results and scripts required to reproduce the tables and figures presented in the SNI5GECT paper, by generating figures and terminal outputs. The evaluation procedure consists of a series of scripts, either included directly in the artifact or described in this appendix. Finally, the artifact features exploit scripts that demonstrate real-world attacks against Commercial Off-the-Shelf (COTS) 5G devices connected to the remote testbed.

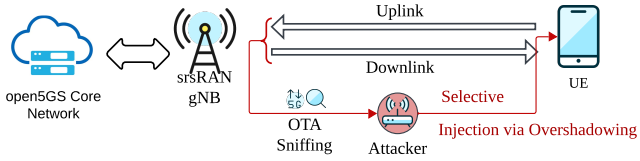

Figure 1: Overview of SNI5GECT evaluation

A.2 Description & Requirements

This artifact demonstrates the capabilities of our framework, including the setup for sniffing over-the-air 5G signals and injecting messages at specific stages of the communication process. To simplify testing, the remote evaluation machine is pre-configured with all necessary system libraries and dependencies required to run SNI5GECT. Hardware requirements are detailed in Section A.2.3, and instructions for building the system from source are provided in Section A.3.

A.2.1 Security, privacy, and ethical concerns

The test environment involves deploying a test 5G base station, which may cause nearby devices to connect to it. However, only devices using SIM cards configured with test Mobile Country Codes (MCC) and Mobile Network Codes (MNC) are affected. During evaluation, target phones may experience temporary effects such as modem crashes, fingerprinting, or network downgrade attacks. These effects are non-permanent and can be fully resolved by rebooting the device or reinserting the SIM card.

No persistent modifications are made to device firmware or operating systems, and no user data is accessed or stored at any point. We strongly recommend using dedicated test devices to avoid unintentional disruption to personal or production equipment.

A.2.2 How to Access

The released artifacts is publicly available on Zenodo: https://doi.org/10.5281/zenodo.15601773.
We also provide a server that has all the hardware and software dependencies. Access to the evaluation server is provided via a Cloudflare SSH tunnel. Please ensure that the cloudflared binary is installed on the client machine; it can be downloaded from the official Cloudflare website: https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/downloads/.

Next, configure SSH client by adding the following entry to ~/.ssh/config file:

```
Host sni5gect-artifacts
    HostName sni5gect.roskey.net
    User root
    ProxyCommand cloudflared access ssh --hostname %h
    IdentityFile ~/.ssh/artifact.key
```

Once configured, the reviewer can access the evaluation environment using:

```
chmod 0600 ~/.ssh/artifact.key
ssh sni5gect-artifacts
```

To transfer PDF or PCAP files to local machine, we recommend using SFTP. Additionally, the Docker container includes an x11vnc server for graphical access. Tools such as

RealVNC Viewer can be used to connect to the graphical interface at `localhost:5900` after setting up a port forwarding using ssh.

```
ssh -L 5900:localhost:5900 sni5gect-artifacts
```

A.2.3 Hardware dependencies

The following hardware components are required to evaluate the SNI5GECT framework:

- **USRP B210 SDR** – Used to run the legitimate srsRAN base station.
- **USRP B210 SDR** – Dedicated to the SNI5GECT framework for over-the-air sniffing and injection.
- **OnePlus Nord CE2** (Patch version: 2023-05-05) – Vulnerable User Equipment (UE) used for evaluation. (adb id: UWEUW4XG8XCA8PWS)
- **Pixel 7** (Patch version: 2023-05-05) – Vulnerable User Equipment (UE) used for evaluation. (adb id: 27211FDH20096Z)
- **Samsung S22** (Patch version: 2024-06-01) – Vulnerable User Equipment (UE) used for evaluation. (adb id: R5CT720QT7H)
- **Huawei P40** (Patch version: 2024-02-01) – Vulnerable User Equipment (UE) used for evaluation. (adb id: K5J0220312001992)
- **Fibocom FM150-AE USB modem** – Vulnerable User Equipment (UE) used for evaluation.

All listed devices are physically connected to the remote evaluation machine and pre-configured for remote experimentation.

A.2.4 Software dependencies

All software dependencies required to run the SNI5GECT framework are specified in the provided Dockerfile. These dependencies are intended to run inside a lightweight Ubuntu 22.04 Docker container without a GUI or other heavy background services. The main runtime components include:

- 5ghoul-5g-nr-attacks – commit `9739994` (included in the artifact).
- SNI5GECT– Modified from srsRAN 4G.
- srsRAN Project – Version `release_24_10_1` (used for the legitimate 5G base station).
- Open5GS – Core network implementation for srsRAN legitimate base station.
- Mongodb – Required by Open5GS for credential storage.
- QCSuper – Tools to communicate with the Qualcomm modems to get the raw frames received.

A.2.5 Benchmarks

We compared the DCI sniffing performance with NR-Scope srsran branch commit version: 2f30b0a3. https://github.com/PrincetonUniversity/NR-Scope

A.3 Getting Started

This section details the steps required to set up the same evaluation environment used in our experiments. To streamline the process and prevent dependency mismatches, we provide a pre-configured `Dockerfile` and `docker-compose.yml`. The `Dockerfile` initializes the environment using the `ubuntu:22.04` base image and installs all necessary components to run the SNI5GECT framework.

A.3.1 Installation (Optional)

To set up the environment, run the following commands from the root of the artifact directory. These steps will install all dependencies, compile the SNI5GECT framework, and start the srsRAN base station, Open5GS core network, and QCSuper in a fully functional state:

```
# Load a pre-build docker container
wget https://zenodo.org/records/15601773/files/sni5\
     gect-artifacts-docker.tar.gz
docker load < sni5gect-artifacts-docker.tar.gz
# Or build the docker container from scratch
wget https://zenodo.org/records/15601773/files/Sni5\
     Gect-source-code.zip
unzip Sni5Gect-source-code.zip
cd Sni5Gect-5GNR-sniffing-and-exploitation-main
docker compose build artifacts --build-arg GITHUB_TOKEN=\
     github_pat_11AH6MR7A02x31orqrUICh_uSzIpTz1u5syNaR2Q3\
     CThfLfofbjC94lKHfDRM9188QLFU2MHCNAViBTdG4
```

Next, download the evaluation results, update the path in the `docker-compose.yml` and start the docker container:

```
wget https://zenodo.org/records/15601773/files/sni5\
     gect-evaluation-results.zip
unzip sni5gect-evaluation-results.zip
docker compose up -d
```

Then to access the container, use the following:

```
docker exec -it artifacts bash
```

The folder structure inside the `/root` directory is listed as follows:

```
.
|-- sni5gect  # Sni5Gect project root folder
|   |-- bin -> /root/wdissector/bin
|   |-- build # Sni5Gect build output folder
|   |-- configs   # Configuration files for Sni5Gect
```

```
|   |-- logs      # Output for logs and PCAP files
|   |-- modules   # Sni5Gect exploit modules
|   |-- scripts   # Scripts to get sniffing performance
|   |-- shadower  # Source code for Sni5Gect framework
|   |   |-- hdr
|   |   |-- modules # Source code of exploit modules
|   |   |-- src
            # Broadcast Worker implementation
|   |   |   |-- broadcast_worker.cc
            # GNB DL Injector implementation
|   |   |   |-- gnb_dl_worker.cc
            # GNB UL Worker implementation
|   |   |   |-- gnb_ul_worker.cc
            # Distributes received subframes to components
|   |   |   |-- scheduler.cc
            # Syncher implementation
|   |   |   |-- syncer.cc
            # UE DL Worker implementation
|   |   |   |-- ue_dl_worker.cc
            # UE Tracker implementation
|   |   |   |-- ue_tracker.cc
            # wDissector wrapper
|   |   |   |-- wd_worker.cc
|-- srsran     # Evaluation legitimate 5G base station
|-- open5gs    # Evaluation legitimate Core network
|-- qcsuper    # QCSuper to intercept raw frames
|-- wdissector # wDissector dependency
# Evaluation results presented in the paper
|-- evaluation_results
|   |-- dci_evaluation # DCI sniffing result
|   |-- scripts        # Utils scripts
|   |-- sni5gect_5g_attacks  # Results for different \
     attacks and different phones
|   |-- sni5gect_injection_different_distance
|   |-- sni5gect_injection_different_state
|   |-- sni5gect_sniffing_srsran # Sniffing performance
|   |-- sni5gect_uplink_sniffing
|       |-- different_distance
|       |-- different_ta_offset
```

A.3.2 Basic Test of Radio Devices

To verify that the USRP B210 SDR devices are correctly connected to the host machine, run the following command:

```
uhd_find_devices
```

The expected output should list both SDR devices, similar to the following Figure 2. If the devices are not detected correctly, please ensure they are properly connected and powered. Next, update the configuration file at the following location:

```
/root/sni5gect/configs/config-srsran-n78-20MHz.conf
```

Ensure the USRP device is correctly specified, for example:

```
[source]
source_type = uhd
source_module = build/shadower/libuhd_source.so
source_params = type=b200,serial=3218CC4
```

Step 1: In terminal 1, start the Open5GS Core Network:

```
cd /root/open5gs
./build/tests/app/app -c open5gs.yaml
```

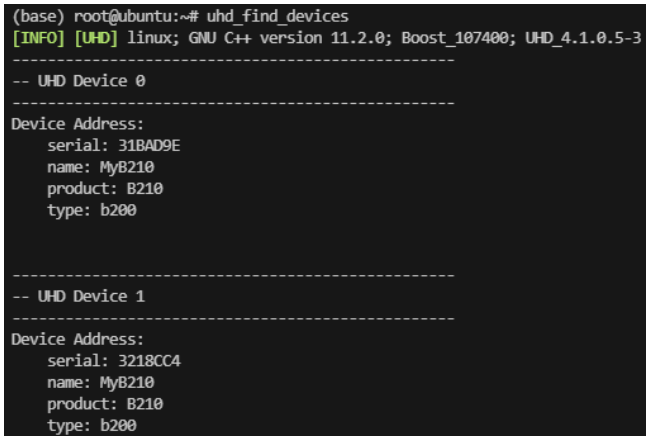

Figure 2: List UHD Devices

Step 2: Open a new terminal, start the srsRAN Base Station:

```
cd /root/srsran/
./build/apps/gnb/gnb -c srsran.conf
```

To confirm that the base station is running correctly, look for the line containing base station frequency information as shown in Figure 3.

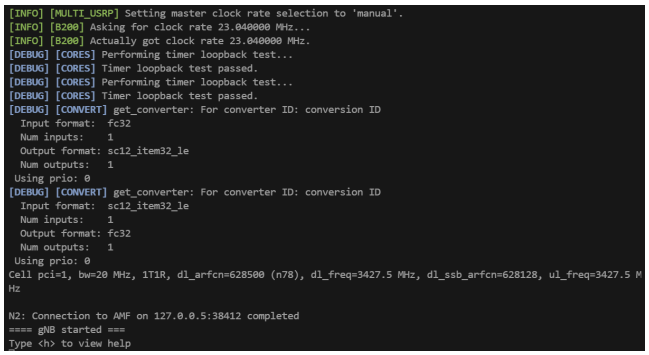

Figure 3: Successful started srsRAN

Step 3: Open a new terminal, start the SNI5GECT Framework as follows:

```
cd /root/sni5gect
./build/shadower/shadower \
configs/config-srsran-n78-20MHz.conf
```

If successful, the output in Figure 4, containing text `MIB applied to all workers` and `SIB1 applied to all workers` confirms that the SNI5GECT framework is running and ready to intercept phones connecting to the legitimate base station.

Step 4: To check if the phones are connected to the computer, open a new terminal and run the following command:

```
adb devices -l
```

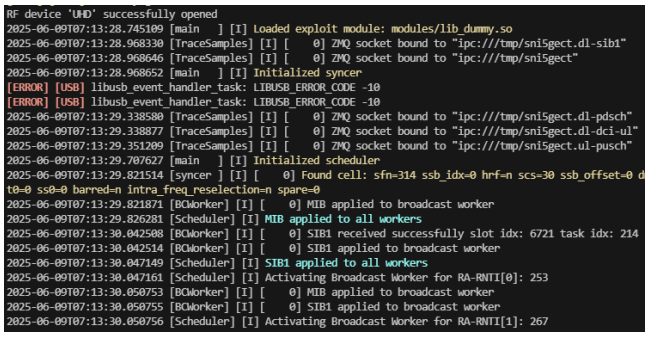

Figure 4: SNI5GECT pending UE connect

The following android devices in Figure 5 can be identified:

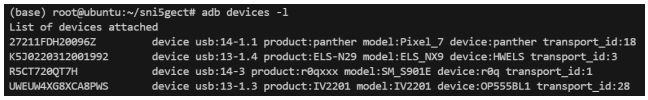

Figure 5: adb devices -l

If the device is connected correctly, the next step is to toggle the airplane mode and see if SNI5GECT captured the phone connecting to the base station.

To turn off airplane mode (connecting to 5G base station):

```
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode disable
```

To check the 5G connection status, please use the following command:

```
adb -s UWEUW4XG8XCA8PWS shell dumpsys telephony.registry |\
     grep mTelephonyDisplayInfo=TelephonyDisplayInfo

# Example Output: Connected to 5G NR
mTelephonyDisplayInfo=TelephonyDisplayInfo {network=NR, \
    overrideNetwork=NONE, isRoaming=false}

# Example Output: Connected to LTE/Emergency Service
mTelephonyDisplayInfo=TelephonyDisplayInfo {network=LTE, \
    overrideNetwork=NONE, isRoaming=false}

# Example Output: No connection
mTelephonyDisplayInfo=TelephonyDisplayInfo {network=\
    UNKNOWN, overrideNetwork=NONE, isRoaming=false}
```

To turn on airplane mode (disconnecting from 5G base station), use the following:

```
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode enable
```

On the SNI5GECT terminal in example Figure 6, if we can observe that SNI5GECT have detected a new UE with the keyword `Found new UE with tc-rnti`, then we can confirm that each component works well.

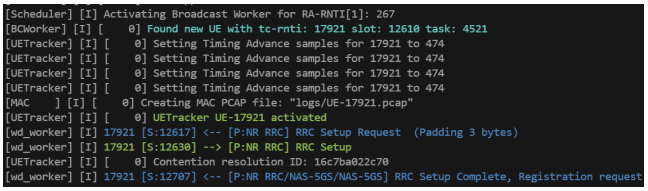

Figure 6: SNI5GECT detected UE

A.4 Evaluation workflow

In this section, we outline the experimental workflow used to evaluate the SNI5GECT framework. The evaluation is structured around three core objectives, each targeting a specific capability of the system. We describe how to reproduce these evaluations using the provided setup and explain how to generate the same figures and tables presented in the paper.

1. **DCI Sniffing Performance**: We begin by evaluating the SNI5GECT framework's ability to detect Downlink Control Information (DCI). DCIs are essential for scheduling both uplink and downlink transmissions, and their reliable decoding is a prerequisite for further message interpretation. This step assesses the robustness and accuracy of DCI detection.

2. **Message Sniffing Performance**: Building on the DCI results, we then assess the overall sniffing capabilities of the SNI5GECT framework. This includes decoding actual downlink and uplink messages exchanged between the User Equipment (UE) and the legitimate base station. This evaluation reveals how effectively the framework can reconstruct ongoing 5G communications and track session states.

3. **Injection and Attack Performance**: Finally, we evaluate the framework's active capabilities. Using QCSuper, we confirm whether the injected messages are successfully received by the USB modem. In addition to confirmation via QCSuper, we also perform over-the-air injection attacks in real-time. This phase demonstrates the effectiveness of SNI5GECT in manipulating live 5G communication between the base station and vulnerable UEs.

A.4.1 Major Claims

In this section, we summarize the major claims presented in the paper and highlight the corresponding expected evaluation outcomes.

(C1): Reliable DCI Sniffing Performance — We claim that SNI5GECT can reliably sniff Downlink Control Information (DCI), which forms the foundation for subsequent decoding of downlink and uplink messages. We expect the DCI sniffing success rate to exceed 90%, as shown in Figure 10.

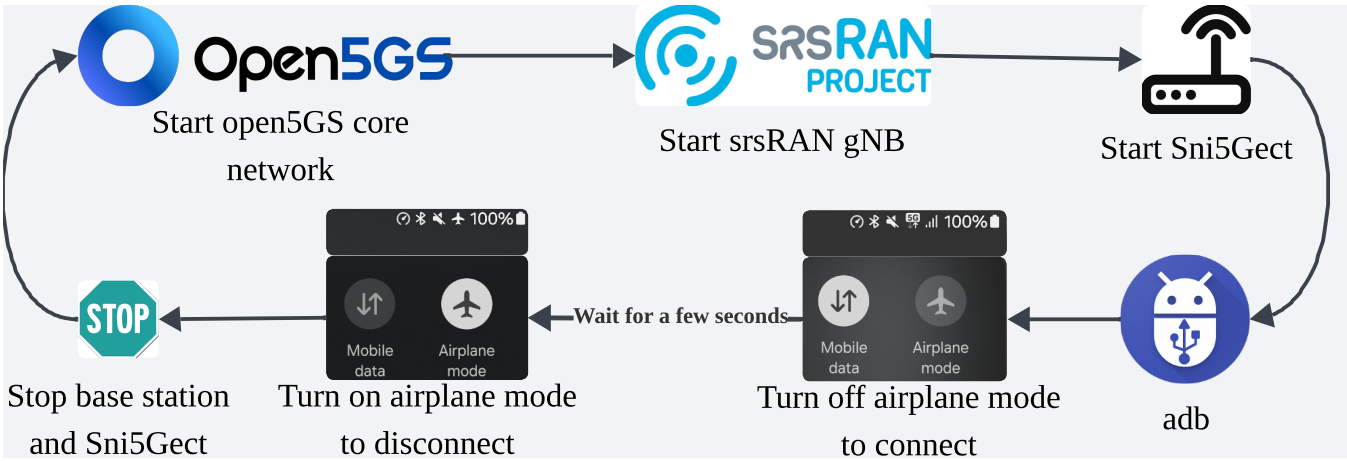

Figure 7: SNI5GECT Evaluation steps

(C2): Robust Sniffing Performance — We claim that the SNI5GECT framework offers robust overall sniffing performance. The expected overall message sniffing accuracy is around 80%, with uplink message decoding often achieving around 70% accuracy. These results correspond to Table 1, Figure 13, and Figure 14.

(C3): Effective Injection Capabilities — We claim that SNI5GECT can effectively inject messages into the communication stream of a target UE. The expected injection success rate can achieve around 60% or higher, as demonstrated in Table 3 and Table 2. This enables a range of successful attacks leveraging the injection capability. Results are shown in Figure 26 and Figure 27.

A.4.2 Evaluation Steps

The overall evaluation steps are shown in the Figure 7. In this section, we describe in detail about how to run each step.

1. **Clear the logs folder:** The logs folder should contain only the logs from the current evaluation session. Residual logs from previous sessions may affect the accuracy of the evaluation results. Please clear the logs folder before starting any experiment using the following command:

```
rm -rf /root/sni5gect/logs/*
```

2. **Start open5GS core network:** The open5GS core network is required to run the legitimate 5G base station. Firstly open a terminal and please refer to the following command to start the open5GS.

```
cd /root/open5gs
./build/tests/app/app -c open5gs.yaml
```

If the open5GS core network fails to start and shows that port 7777 is in use, please use the following command to kill existing open5gs and run the command above to start the open5GS core network again:

```
sudo pkill -9 app
sudo pkill -9 open5gs
```

The logs will be stored in the following file `/root/sni5gect/logs/open5gs.log`

3. **Start srsRAN gNB:** After the open5GS core network starts correctly, open a new terminal and use the following command to start the srsRAN base station.

```
cd /root/srsran/
./build/apps/gnb/gnb -c srsran.conf
```

The result `gnb.log` and `mac_nr.pcap` will be stored in folder `/root/sni5gect/logs/` for later evaluation.

4. **Start SNI5GECT:** After the srsRAN base station starts correctly, open a new terminal and start the SNI5GECT framework and pipe the command line output to the logs folder for later evaluation.

```
cd /root/sni5gect
./build/shadower/shadower configs/config-srsran-n78-20MHz.conf | tee logs/sni5gect.log
```

5. **Toggle airplane mode:** The airplane mode can be easily controlled via adb command. Please note that there are multiple phones connected to the server, so the evaluators have to specify the adb id to control a specific phone.

```
# Disable airplane mode (connect to 5G base station)
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode disable

# Turn on airplane mode (disconnect from 5G base \
    station)
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode enable
```

6. **Stop the base station and SNI5GECT:** Simply press Ctrl+C to stop each component.

A.4.3 Experiments

This section describes the experiments conducted to evaluate and support each claim. We also demonstrate how to use the provided scripts to analyze the provided test data and reproduce the tables and figures presented in the paper.

(E1): DCI Sniffing Evaluation: [60 human-minutes]

This experiment evaluates the DCI sniffing performance of the SNI5GECT framework by comparing the number of correctly sniffed DCI messages with the ground truth obtained from the srsRAN legitimate base station.

How to: Connect a UE (e.g., a phone) to the legitimate base station. Once SNI5GECT detects the target UE, it begins sniffing the DCI messages exchanged between the base station and the UE. After collecting data for 5 to 10 seconds, turn on the airplane mode, stop both SNI5GECT and the base station. (This is because some evaluation devices may unexpectedly drop their connection after approximately 15 seconds. The recommended 5–10 second capture window helps ensure that the connection remains stable throughout the evaluation period.) Then, compare the number of correctly sniffed DCIs against the ground truth logs generated by the base station.

Preparation: Edit the configuration file:

```
/root/sni5gect/configs/config-srsran-n78-20MHz.conf
```

Set the logging level for the worker to DEBUG and ensure the dummy exploit module is used for passive sniffing:

```
[log]
log_level = INFO
syncer_log_level = INFO
worker_log_level = DEBUG # Change the worker log \
    level to debug to retrieve the DCI information
bc_worker_log_level = INFO

[exploit]
module = modules/lib_dummy.so
```

Execution: The command used to run the evaluation is provided in Section A.4.2. Please following the command provided to run the evaluation and collect the logs. After data collection, stop both the base station and the SNI5GECT framework.

An example command-line output is shown in Figure 8. From the white text, we can observe the transmitted DCI information, including their formats, frequency and time allocations, as well as the modulation and coding scheme. The subsequent line demonstrates how SNI5GECT utilizes this scheduling information from the DCI to attempt decoding the actual messages.

Results: The logs for the above evaluation will be stored in the folder `/root/sni5gect/logs`. Please use the following script to compute the DCI sniffing success rate, it will give out the total number of DCIs from srsRAN, number of DCIs sniffed by SNI5GECT and the corresponding success rate in the terminal output. Example output is shown in Figure 9.

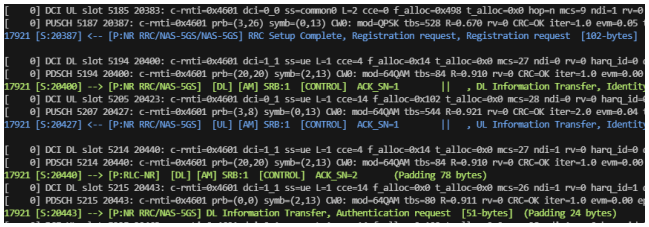

Figure 8: DCI sniffing example output

```
cd /root/sni5gect
python3 scripts/get_dci_sniffing_performance.py
```

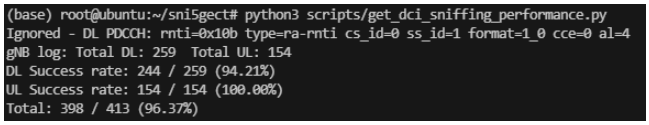

Figure 9: Get DCI sniffing performance example output

Figure 10 corresponds to **Figure 6: Success Rate of DCI Search** in the paper. Data and scripts are located at `/root/evaluation_results/dci_evaluation/`

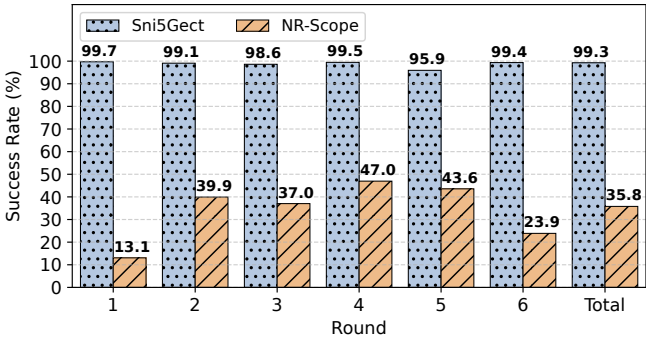

Figure 10: Success Rate of DCI search

To evaluate the existing result and plot the performance as shown in the paper, run the following commands:

```
cd /root/evaluation_results

# Get the DCI sniffing performance for Sni5Gect
python3 dci_evaluation/compare_sni5gect.py
# Result stored in: /root/evaluation_results/\
    dci_evaluation/sni5gect_eval_result.csv

# Get the DCI sniffing performance for NR-Scope
python3 dci_evaluation/compare_nrscope.py
# Result stored in: /root/evaluation_results/\
    dci_evaluation/nrscope_eval_result.csv
```

```
# Plot the figure based on the CSV data
cd /root/evaluation_results/dci_evaluation
python3 plot_result.py
# Result plot: /root/evaluation_results/\
    dci_evaluation/dci_sniffing_success_rate.pdf
```

(E2): Message Sniffing Evaluation: [60 human-minutes] This experiment evaluates the message sniffing performance of the SNI5GECT framework. It compares the decoded downlink and uplink messages with the ground truth collected from the srsRAN base station to measure overall accuracy.

How to: Connect a UE (phone) to the legitimate base station. Once SNI5GECT detects the target UE, it begins sniffing both downlink and uplink messages from the over-the-air signals exchanged between the base station and the UE. After collecting data for 5 to 10 seconds, stop both SNI5GECT and the base station. The resulting PCAP file will be stored in the `/root/sn5gect/logs/` directory. Finally, compare the sniffed messages with the ground truth obtained from the base station.

Preparation: Configure the `worker_log_level` to `INFO` and ensure the dummy exploit module is used for passive sniffing:

```
[log]
log_level = INFO
syncer_log_level = INFO
worker_log_level = INFO
bc_worker_log_level = INFO

[exploit]
module = modules/lib_dummy.so
```

Execution: To evaluate the performance of message sniffing, the process is the same as what we have shown in DCI sniffing. Please refer to the same commands presented in Section A.4.2. An example command-line output is shown in Figure 11. In the output, downlink messages sent from the base station to the UE are highlighted in green, while uplink messages sent from the UE to the base station are highlighted in blue.

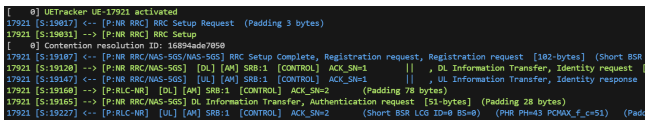

Figure 11: Sniffing Example output

Results: The logs for the above evaluation will be stored in the folder `/root/sni5gect/logs`, with the messages sniffed by SNI5GECT stored in the file with name for example `UE-17921.pcap`. Please use the following script to compute the message sniffing success rate, it will give out the total number of messages sent between srsRAN and the UE, as well as the messages sniffed by SNI5GECT and the corresponding success rate in the terminal output. Example output is shown in Figure 12.

```
cd /root/Sni5Gect
python3 scripts/get_sniffing_performance.py
```

```
(base) root@ubuntu:~/sni5gect# python3 scripts/get_sniffing_performance.py
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Parsing Sni5Gect pcap file: /root/sni5gect/logs/UE-17921.pcap
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
RNTI: 17921 MSIN: 0000064953:
DL Success rate: 222 / 222 (100.00%)
UL Success rate: 118 / 132 (89.39%)
```

Figure 12: Get sniffing performance example output

The paper presents three categories of message sniffing performance evaluation:

1. **Table 1**: Corresponds to **Table 2** in the paper and demonstrates the effectiveness of SNI5GECT in sniffing both downlink and uplink messages.
2. **Figure 13**: Corresponds to **Figure 4** in the paper and shows the success rate of uplink sniffing with respect to the distance between the UE and the base station.
3. **Figure 14**: Corresponds to **Figure 5** in the paper and illustrates the impact of varying TA (Timing Advance) command offsets on uplink sniffing performance.

Table 1: Effectiveness of SNI5GECT sniffing capability for different distance between the UE and the SNI5GECT SDR. Number of sniffed messages is provided alongside accuracy.

Device	Dist.	Sniffing Accuracy	Uplink	Downlink
OnePlus Nord CE 2	0 m	93.32% (4418)	75.77% (938)	99.54% (3480)
	1 m	92.20% (8074)	70.45% (1576)	99.66% (6498)
Huawei P40 Pro	0 m	94.86% (1180)	86.08% (371)	99.51% (809)
	1 m	94.74% (1262)	84.87% (387)	99.89% (875)
Pixel 7	0 m	86.55% (1094)	75.89% (491)	97.73% (603)
	1 m	90.71% (1768)	74.11% (435)	97.87% (1333)
Samsung Galaxy S22	0 m	93.15% (1061)	88.78% (467)	96.90% (594)
	1 m	97.51% (745)	94.68% (338)	100% (407)

Table 1: This table summarizes the effectiveness of SNI5GECT in capturing both downlink and uplink messages. The evaluation was conducted with the UE positioned at two distances from the base station: 0 meters and 1 meter. The corresponding experimental results are stored in the following directory:

```
/root/evaluation_results/sni5gect_sniffing_srsran
```

To regenerate the CSV file from the raw log data, run the following script, which parses the experiment results and generates the message sniffing performance metrics:

```
cd /root/evaluation_results
python3 sni5gect_sniffing_srsran/\
    get_sniffing_performance.py
# Output file: /root/evaluation_results/sni5\
    gect_sniffing_srsran/sniffing_performance.csv
```

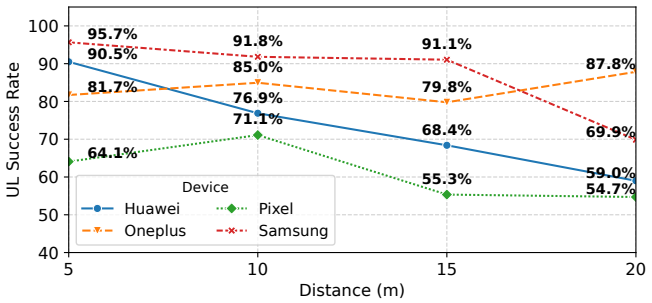

Figure 13: Success Rate of Uplink Sniffing w.r.t distance

Figure 13: Success Rate of Uplink Sniffing w.r.t distance This figure illustrates the success rate of uplink message sniffing as the distance between the UE and the sniffer increases. The raw experimental data used to generate this figure is stored in the following directory:

```
/root/evaluation_results/sni5gect_uplink_sniffing/\
    different_distance
```

To compute the success rate and generate Figure 13, execute the following scripts:

```
cd /root/evaluation_results/sni5gect_uplink_sniffing
python3 parse_logs_different_distance.py
# Output different_distance_detail.csv

python3 plot_different_distance.py
# Result figure: uplink_sniffing_different_distance.\
    pdf
```

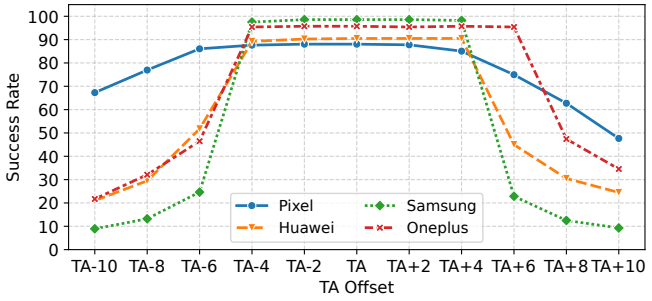

Figure 14: Uplink Sniffing w.r.t TA command offset

Figure 14: Uplink Sniffing w.r.t TA command offset This figure presents the uplink message sniffing success rate under varying Timing Advance (TA) command offsets. The raw data used for this analysis is located in the following directory:

```
/root/evaluation_results/sni5gect_uplink_sniffing/\
    different_ta_offset
```

To process the logs, calculate the success rates, and generate Figure 14, execute the following commands:

```
cd /root/evaluation_results/sni5gect_uplink_sniffing
python3 parse_logs_different_ta_offset.py
# Output different_ta_offset_detail.csv

python3 plot_different_ta.py
# Result figure: uplink_sniffing_different_ta.pdf
```

(E3): Message Injection Evaluation: [120 human-minutes] This experiment measures the effectiveness of the SNI5GECT framework in injecting messages towards a victim device. Using QCSuper and a USB modem, we confirm successful message reception by the victim UE.
How to: In this evaluation, we utilize QCSuper, a tool capable of interfacing directly with Qualcomm-based USB modems to capture raw over-the-air frames. This enables us to verify whether specific messages are received by the target UE. We use `cutecom` to interact with the USB modem over the serial interface in order to manually initiate and terminate 5G connections.
Preparation: To perform message injection during the 5G attach procedure, different exploit modules are required depending on the message type and protocol state. For example, to inject a NAS message immediately after a `Registration Request`, the module for `Registration Reject` should be used, since the initial connection typically begins with the UE sending a `Registration Request` message.

```
[exploit]
module = modules/lib_dg_registration_reject.so
```

Execution: First, start the base station as described in Section A.4.2. Then, launch QCSuper with the following command. This will capture a packet trace in `qcsuper.pcap` and display a live view of incoming frames from the USB modem. The reviewer can use client such as VNCViewer to connect to the vnc server and observe the live traffic.

```
cd /root/qcsuper
python3 qcsuper.py --usb-modem auto --pcap-dump /root\
    /sni5gect/logs/qcsuper.pcap --wireshark-live
```

Next, open a separate terminal and start `cutecom` to communicate with the USB modem via its serial interface. This can be done in the GUI from the vnc session which allows sending AT commands to trigger connection and disconnection events on the target UE.
Alternatively, you can use the terminal-based application `minicom`. To start a new session with `minicom`, run the following command:

```
minicom -D /dev/ttyUSB2
# Exit minicom:
# Ctrl+A followed by X, then press Enter to exit \
    minicom
```

Please refer to the following command to toggle the airplane mode of the USB modem.

```
# Check the modem starts and connects correctly
AT  # Expecting: OK
```

```
# Run cell search
AT+cops=?

# Connect to 5G
AT+cops=1,2,00101,12

# Disconnect from base station
AT+cops=2
```

An example command-line output is shown in Figure 15. After receiving the `Registration Request` message, the UE releases the connection. Following a timeout period, the base station sends an `RRC Release` message to terminate the connection.

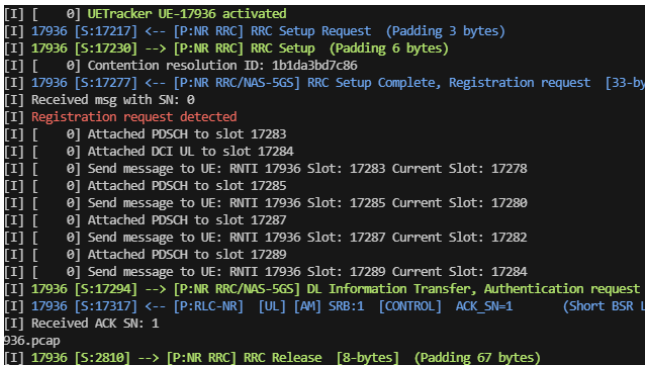

Figure 15: Message Injection stdout Example

An example result captured from QCSuper is shown in Figure 16. From the screenshot, we can confirm that the UE receives and accepts our injected `Registration Reject` message with the cause `N1 Mode not Allowed`.

Figure 16: Message Injection QCSuper Example

Results: The evaluation process above will store the result `qcsuper.pcap` to folder `/root/sni5gect/logs` by default. To get the success rate of the injection performance, please simply use the following command. This will print the total number of UE accepted injections over the total number of injections, as well as the success rate in percentage on the standard output terminal. The example output is shown in Figure 17

```
cd /root/sni5gect
python3 scripts/get_injection_performance.py
```

There are two sets of injection evaluation results presented in the paper:

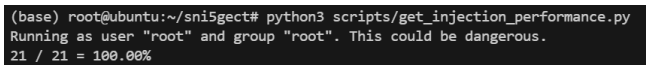

Figure 17: Get message injection performance

- **Table 3** corresponds to **Table 3: SNI5GECT Injection Performance**.
- **Table 2** corresponds to **Table 4: Injection Performance at Different Distances**.

Table 3 evaluates the injection success rate and message duplication behavior at various states of the 5G connection procedure. The raw experiment data are stored in the following directory:

```
/root/evaluation_results/sni5\
    gect_injection_different_state
```

To regenerate the CSV file used in the paper and produce the summarized table, run the following command:

```
# Result for column RRC Setup Request
cd /root/evaluation_results/sni5\
    gect_injection_different_state/\
    injection_after_rrc_setup
python3 evaluate.py

# Result for column Registration Request
cd /root/evaluation_results/sni5\
    gect_injection_different_state/\
    inject_after_registration_request
python3 evaluate.py

# Result for column Authentication Failure
cd /root/evaluation_results/sni5\
    gect_injection_different_state/\
    inject_after_authentication_failure
python3 evaluate.py

# Result for column Security Mode Complete
cd /root/evaluation_results/sni5\
    gect_injection_different_state/\
    inject_after_security_mode_complete
python3 evaluate.py
```

Table 2: Injection Performance at Different Distances.

Distance	Total	Success	Success Rate
5m	73	70	95.89%
10m	65	61	93.85%
15m	73	69	94.52%
20m	73	61	83.56%

Table 2 evaluates how physical distance between the attacker and the target UE impacts the injection success rate. The raw data are stored in the directory:

```
/root/evaluation_results/sni5\
    gect_injection_different_distance
```

To regenerate the CSV file containing the success and total counts for each round at different distances, run:

```
cd /root/evaluation_results/sni5\
    gect_injection_different_distance
python3 parse_logs.py
# Result: result.csv
```

(E4): Example Attacks Evaluation: [Several hours] This experiment demonstrates real-world attacks that leverage the message injection capabilities of the SNI5GECT framework. These attacks are conducted against a live victim UE to validate the framework's end-to-end exploitation potential.

How to: In this evaluation, we trigger the phones to connect to the target base station. At specific states during the connection process, SNI5GECT injects carefully crafted or intercepted messages into the communication stream. This can induce various effects, such as protocol downgrades, device fingerprinting, modem crashes, or authentication bypasses.

Preparation: For each type of attack, use a dedicated attack module. Refer to the list below for the appropriate modules corresponding to each attack. Uncomment the relevant line to enable the desired exploit module. Please refer to Table 4 for the function of each module and please note that only one exploit module should be loaded at a time.

```
# Sniffing
module = modules/lib_dummy.so
# Fingerprint
module = modules/lib_identity_request.so
# Sniff and extract Authentication Request
module = modules/\
    lib_dg_authentication_request_sniffer.so
# Downgrade Attacks
module = modules/lib_dg_authentication_replay.so
module = modules/lib_dg_registration_reject.so
# 5Ghoul attacks: crash the OnePlus MTK modem
module = modules/lib_mac_sch_mtk_rlc_crash.so
module = modules/lib_mac_sch_mtk_rrc_setup_crash_3.so
module = modules/lib_mac_sch_mtk_rrc_setup_crash_4.so
module = modules/lib_mac_sch_mtk_rrc_setup_crash_6.so
module = modules/lib_mac_sch_mtk_rrc_setup_crash_7.so
# Authentication Bypass: affect Pixel 7 Exynos modem
module = modules/lib_plaintext_registration_accept.so
```

Execution: Update the exploit modules as needed, then start the base station and SNI5GECT as described in Section A.4.2. Toggle airplane mode on the device to trigger a new connection.

```
# Trigger the phone to connect to the base station
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode disable
# Wait for a few seconds
# Disconnect the phone from the base station
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode enable
```

Results: The behaviors and impacts of different attacks vary. Some attacks result in effects such as fingerprinting, downgrade, or authentication bypass.

Registration Reject Attack: After the User Equipment (UE) sends a `Registration Request` message, SNI5GECT injects a `Registration Reject` message. Upon receiving and accepting this message, the UE disconnects from the base station. Subsequently, the base station may repeatedly attempt to resend messages such as `Identity Request`, `Authentication Request`, or `Security Mode Command`, but receive no response from the UE. An example command-line output is shown in Figure 18. We can see SNI5GECT sniffed the injected message `Registration Reject`, we can also observe the base station kept sending `Authentication Request` after the UE disconnects. If subsequent messages such as `Authentication Response` or `Security Mode Complete` appear, or if there are excessive occurrences of `UL-SCH` or `DL-SCH`, it indicates that the attack has failed. Please toggle airplane mode and attempt the attack again.

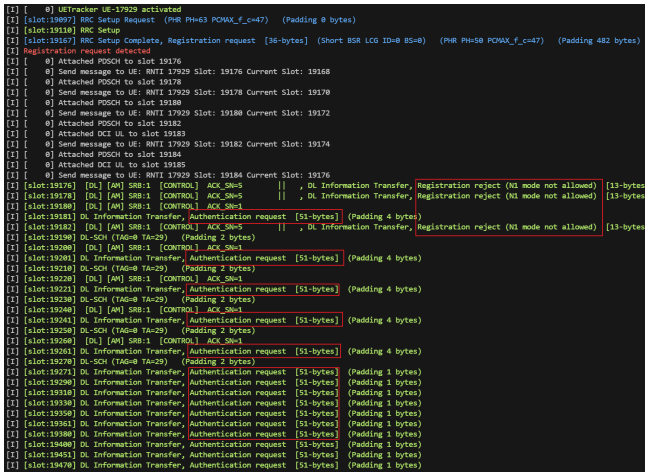

Figure 18: Registration Reject Example output

To run the evaluation, please follow the steps in Section A.4.2. The result is expected to be stored in `/root/sni5gect/logs` folder. To get the success rate of such attack, please use the following command. This will give out the success rate of `Registration Reject` attack in standard output terminal similar to Figure 19.

```
python3 scripts/get_registration_reject_performance.\
    py
```

Identity Request Attack: Following the UE's transmission of a `Registration Request` message, SNI5GECT injects an `Identity Request` message. When the victim UE receives and processes this message, it responds with an `Identity Response`. SNI5GECT then sniffs the `Identity Response` and extracts the `SUCI`, demonstrating a fingerprinting attack. An example command-line output is provided in Figure 20, we can observe the base station sends `Security Mode Command`, but the

Table 3: SNI5GECT Injection Performance.

Duplication #	RRC Setup Request			Registration Request			Authentication Failure			Security Mode Complete		
	# Messages	# Injected	Success Rate	# Messages	# Injected	Success Rate	# Messages	# Injected	Success Rate	# Messages	# Injected	Success Rate
1	356	306	85.96%	232	181	78.02%	47	29	61.70%	51	23	45.10%
2	251	219	87.25%	423	337	79.67%	72	60	83.33%	67	60	89.55%
3	316	272	86.08%	350	262	74.86%	42	28	66.67%	46	44	95.65%
4	234	202	86.32%	364	297	81.59%	71	51	71.83%	43	39	90.70%

Module File	Description / Notes
`lib_dummy.so`	Pure sniffing; no injection occurs.
`lib_identity_request.so`	Fingerprinting attack: Injects Identity Request and expects an Identity Response.
`lib_dg_authentication_request_sniffer.so`	Sniffs the Authentication Request message.
`lib_dg_authentication_replay.so`	Authentication replay attack: After sniffing, update the relevant file `shadower/modules/dg_authentication_replay.cc` with the captured Authentication Request and recompile (`ninja -C build`). **Note:** After a successful attack, the UE may not reconnect to the base station for 5+ minutes. Reboot the phone using `adb -s {id} reboot`.
`lib_dg_registration_reject.so`	Downgrade attack: Sends Registration Reject to trigger immediate downgrade.

`lib_mac_sch_mtk_rlc_crash.so`
`lib_mac_sch_mtk_rrc_setup_crash_3.so`
`lib_mac_sch_mtk_rrc_setup_crash_4.so`
`lib_mac_sch_mtk_rrc_setup_crash_6.so`
`lib_mac_sch_mtk_rrc_setup_crash_7.so`
	5Ghoul attacks: Crash attacks affecting OnePlus devices with MTK modems.

`lib_plaintext_registration_accept.so`	Authentication bypass: Only affects Pixel 7 devices with Exynos modem.

Table 4: Attack Modules and Their Descriptions

```
(base) root@ubuntu:~/sni5gect# python3 scripts/get_registration_reject_performance.py
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
Running as user "root" and group "root". This could be dangerous.
13 / 13 = 100.00%
```

Figure 19: Get Registration Reject Performance Output

```
[    0] UETracker UE-17922 activated
[slot:10777] RRC Setup Request  (PHR PH=63 PCMAX_f_c=47)   (Padding 0 bytes)
[slot:10790] RRC Setup
[    0] Contention resolution ID: 1271a2896270
[slot:10867] RRC Setup Complete, Registration request, Registration request  [94-bytes]  (Short BSR LCG ID=0
Sent identity request
[    0] Attached PDSCH to slot 10876
[    0] Send message to UE: RNTI 17922 Slot: 10876 Current Slot: 10868
[    0] Attached PDSCH to slot 10878
[    0] Send message to UE: RNTI 17922 Slot: 10878 Current Slot: 10870
[    0] Attached PDSCH to slot 10880
[    0] Send message to UE: RNTI 17922 Slot: 10880 Current Slot: 10872
[slot:10876]  [DL] [AM] SRB:1  [CONTROL]  ACK_SN=1        ||   , DL Information Transfer, Identity request
[slot:10878]  [DL] [AM] SRB:1  [CONTROL]  ACK_SN=1        ||   , DL Information Transfer, Identity request
[slot:10880]  [DL] [AM] SRB:1  [CONTROL]  ACK_SN=1
[slot:10885] Security Mode Command  [9-bytes]  (Padding 3 bytes)
[slot:10907]  [UL] [AM] SRB:1  [CONTROL]  ACK_SN=1        ||   , UL Information Transfer, Identity response
Identity: 0000064952
[slot:10920]  [DL] [AM] SRB:1  [CONTROL]  ACK_SN=2       (Padding 78 bytes)
[slot:10950] DL-SCH (TAG=0 TA=29)   (Padding 81 bytes)
[slot:12325] RRC Release  [8-bytes]  (Padding 71 bytes)
```

Figure 20: Identity Request Example output

```
(base) root@ubuntu:~/sni5gect# python3 scripts/get_identity_request_performance.py
6 / 10 = 60.00%
```

Figure 21: Get Identity Request Performance Output

UE replies with `Identity Response` and the extracted SUCI is shown in red. If this red message does not appear, it indicates that either the injection has failed or the sniffer did not successfully capture the response. Please toggle airplane mode to establish a new connection and rerun the attack.

To run the evaluation, please follow the steps in Section A.4.2. The result is expected to be stored in `/root/sni5gect/logs` folder. To get the success rate of such attack, please use the following command. This will give out the success rate of `Identity Request` attack in standard output terminal similar to Figure 21.

```
python3 scripts/get_identity_request_performance.py
```

Authentication Replay Attack: This is the most complex exploit in our evaluation and involves two stages: sniffing and replaying. In the first stage (sniffing), the module `lib_dg_authentication_request_sniffer.so` is used to capture a legitimate `Authentication Request` message sent from the base station to the UE. This sniffing step only needs to be performed once per specific UE; the captured `Authentication Request` can be reused in subsequent replay attempts.

In the second stage (replaying), when a `Registration Request` message is received, SNI5GECT replays the previously captured `Authentication Request`. The UE responds with an `Authentication Failure` (cause: `Synch Failure`) and starts timer T3520. After sniffing this response, SNI5GECT updates the RLC and PDCP sequence numbers and replays the `Authentication Request` again. This process may repeat multiple times. Eventually, after timer T3520 expires, the UE deems that the network has failed the authentication check, it locally releases the connection, and bars the active cell. If no other 5G base station is available, the UE ignores the currently available cell, downgrades to 4G and remains in downgraded state for 300 seconds, as specified in 3GPP TS 24.501 version 16.5.1 Release 16, section 5.4.1.2.4.5 (Abnormal cases in the UE). In the example output shown in Figure 22, the UE can be seen replying with the `Authentication Failure` message twice, and after replaying the `Authentication Request` message the third time, the UE disconnects, and the base station is keeps sending `RRC Release`. If messages such as `Authentication Response` or `UE Capability Enquiry`—which indicate progression in the connection—appear, or if there are excessive occurrences of `UL-SCH` or `DL-SCH`, it means the attack has failed. Please toggle airplane mode and try the attack again.

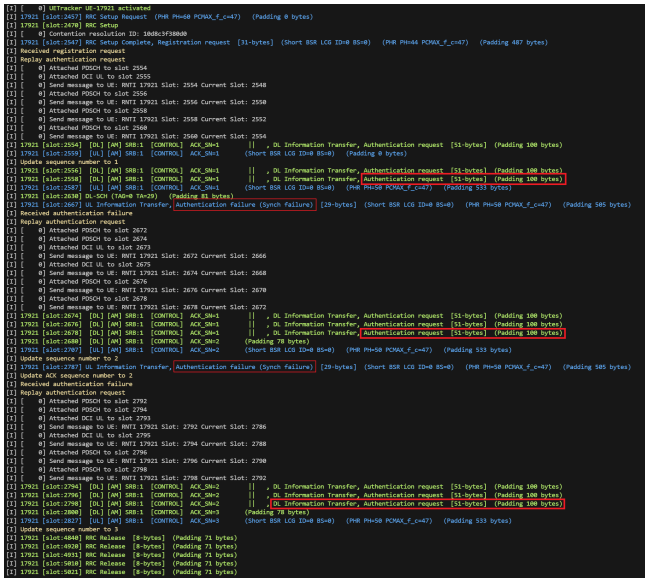

Figure 22: Authentication Replay Example output

To run the evaluation, please follow the steps in Section A.4.2. Please note that in this evaluation, we suggest using the open5GS from the wdissector directory.

```
cd /root/wdissector/
./3rd-party/open5gs-core/build/tests/app/app -c \
    configs/5gnr_gnb/open5gs.yaml
```

The result is expected to be stored in `/root/sni5gect/logs` folder. To get the success rate of such attack, please use the following command, which considers the attack as successful when `Authentication Failure` is received more than one time and the connection is not successfully established. This will give out the success rate of `Authentication Request` replay attack in standard output terminal.

```
python3 scripts/get_auth_replay_performance.py
```

5Ghoul Attacks: These attacks demonstrate how vulnerabilities discovered by the 5Ghoul framework can be exploited to crash target UE modems using SNI5GECT, resulting in Denial-of-Service (DoS) conditions.

```
# CVE-2023-20702
# module = modules/lib_mac_sch_mtk_rlc_crash.so
# CVE-2023-32843
# module = modules/lib_mac_sch_mtk_rrc_setup_crash_3.\
    so
# CVE-2023-32842
# module = modules/lib_mac_sch_mtk_rrc_setup_crash_4.\
    so
# CVE-2024-20003
# module = modules/lib_mac_sch_mtk_rrc_setup_crash_6.\
    so
# CVE-2023-32845
# module = modules/lib_mac_sch_mtk_rrc_setup_crash_7.\
    so
```

The current implementation targets OnePlus phones specifically. To prepare the device for testing, you may toggle airplane mode using the following command:

```
# Disable airplane mode to trigger connection
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode disable
# Enable airplane mode to drop connection
adb -s UWEUW4XG8XCA8PWS shell cmd connectivity \
    airplane-mode enable
```

The core idea of these attacks is to transmit malformed RLC or RRC Setup messages to the target UE. Upon processing these malformed messages, the UE crashes immediately. Full details of these exploits are available on the official 5Ghoul disclosure page: https://asset-group.github.io/disclosures/5ghoul/ To verify whether an attack has succeeded, monitor the device logs using logcat. Specifically, look for crash-related keywords such as `sModemReason`. Open a new terminal and run the following command to begin monitoring:

```
# Clear existing logcat logs
adb -s UWEUW4XG8XCA8PWS logcat -c
# Monitor logcat logs
adb -s UWEUW4XG8XCA8PWS logcat -b radio,crash,system,\
    main | grep sModemReason
```

```
# module = modules/lib_mac_sch_mtk_rlc_crash.so
# 07-04 10:51:37.365  1622  2198 D \
    MDMKernelUeventObserver: sModemReason:fid:457077\
    782;cause:[Fatal error(MPU_NOT_ALLOW)] err_code1\
    :0x0000001D err_code2:0x910D66F6 err_code3:0x910\
    D66E2 CaDeFa Supported

# module = modules/lib_mac_sch_mtk_rrc_setup_crash_3.\
    so
# 07-04 11:04:51.807  1622  2198 D \
    MDMKernelUeventObserver: sModemReason:fid:156734\
    6682;cause:[ASSERT] file:mcu/l1/nl1/internal/md9\
    7/src/rfd/nr_rfd_configdatabase.c line:4380 p1:0\
    x00000001

# module = modules/lib_mac_sch_mtk_rrc_setup_crash_4.\
    so
# 07-04 11:06:50.880  1622  2198 D \
    MDMKernelUeventObserver: sModemReason:fid:372552\
    4241;cause:[ASSERT] file:mcu/l1/mml1/mml1_endc/\
    src/mml1_endc_db_hdlr.c line:524 p1:0x91920c70

# module = modules/lib_mac_sch_mtk_rrc_setup_crash_6.\
    so
# 07-04 11:09:46.846  1622  2198 D \
    MDMKernelUeventObserver: sModemReason:fid:105190\
    3049;cause:[ASSERT] file:mcu/l1/nl1/internal/md9\
    7/src/ctrl/nr_ctrl_mgm.c line:16460 p1:0x0000000\
    0

# module = modules/lib_mac_sch_mtk_rrc_setup_crash_7.\
    so
# 07-04 11:10:42.422  1622  2198 D \
    MDMKernelUeventObserver: sModemReason:fid:100627\
    7484;cause:(MSONIC0) [ASSERT] file:dsp3/\
    coresonic/msonic/modem/brp/nr/nr_brp/src/\
    nr_brp_top_irq.c line:927
```

If the attack is successful, you will observe crash indicators, including the `sModemReason` keyword, as shown in Figure 23 and Figure 24. A large number of red error messages may appear, and the SNI5GECT tool itself may crash—this is expected behavior. Since SNI5GECT also receives the malformed RRC Setup message it generates, it may attempt to decode and apply the invalid configuration.

Please note: after a successful attack, the UE's modem will crash and may not reconnect to the base station immediately. Recovery typically requires either a waiting period (e.g., one minute) or manually rebooting the device to restore 5G connectivity.

5G AKA Bypass Attack: This attack is implemented in the `lib_plaintext_registration_accept.so` attack module, located in the modules folder. It is currently applicable only to the Pixel phone in our testbed. Please use the following command to toggle airplane mode on the Pixel device:

```
# Disable airplane mode to trigger connection
adb -s 27211FDH20096Z shell cmd connectivity \
    airplane-mode disable
# Enable airplane mode to drop connection
adb -s 27211FDH20096Z shell cmd connectivity \
    airplane-mode enable
```

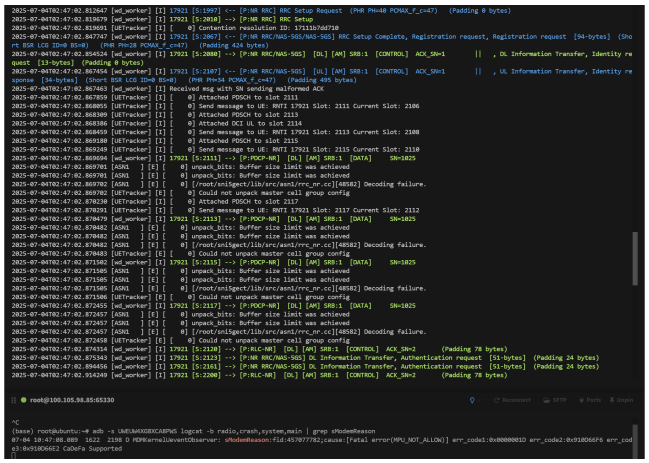

Figure 23: MTK RLC Crash

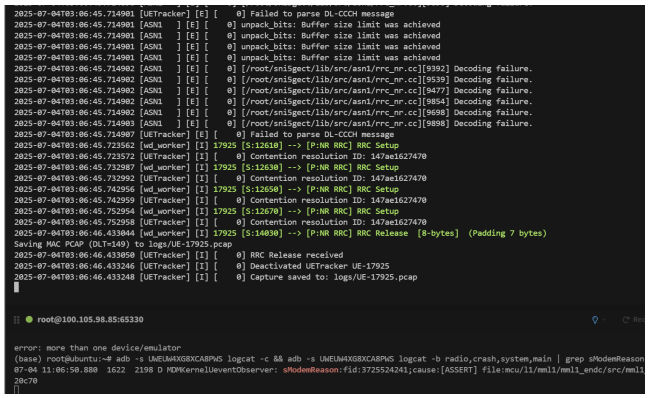

Figure 24: MTK RRC Setup Crash

The core logic of the attack is as follows: after the UE transmits a `Registration Request` message, SNI5GECT injects a `Registration Accept` message. Upon accepting this message, the victim UE responds with `Registration Complete` and a `PDU Session Establishment Request`. Since the core network receives these unexpected messages, it instructs the gNB to release the connection by sending an `RRC Release` message, thereby immediately terminating the connection. We can observe the aforementioned process in Figure 25. If messages such as `Authentication Response` or `Security Mode Complete` are observed and the UE proceeds with the connection, it indicates that the attack has failed. Please toggle airplane mode to establish a new connection and rerun the attack.

Evaluation results presented in the paper: The evaluation result presented in the paper is stored in the following structure in:

```
/root/evaluation_results/sni5gect_5g_attacks
```

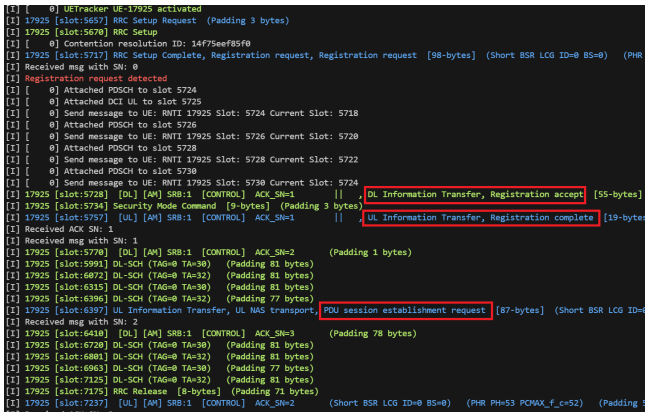

Figure 25: 5G AKA Bypass Example output

```
# Organized as <phone>_<base_station>/<exploit>_<\
    distance>m
.
|-- 5Ghoul_srsran
|   |-- mtk_rlc_crash
|   |-- mtk_rrc_setup_crash_3
|   |-- mtk_rrc_setup_crash_4
|   |-- mtk_rrc_setup_crash_6
|   |-- mtk_rrc_setup_crash_7
|-- 5Ghoul_effnet
|-- oneplus_srsran
|   |-- authentication_replay_0m
|   |-- authentication_replay_1m
|   |-- identity_request_0m
|   |-- identity_request_1m
|   |-- registration_reject_0m
|   |-- registration_reject_1m
|-- oneplus_effnet
|   |-- authentication_replay_0m
|   |-- authentication_replay_1m
|   |-- identity_request_0m
|   |-- identity_request_1m
|   |-- registration_reject_0m
|   |-- registration_reject_1m
|-- huawei_srsran
|-- huawei_effnet
|-- pixel_srsran
|-- pixel_effnet
|-- samsung_srsran
|-- samsung_effnet
|-- registration_accept
```

To generate the same table as presented in the paper, please refer to the following command:

```
cd /root/evaluation_results
python3 sni5gect_5g_attacks/\
    calculate_registration_reject_success_rate.py
# Result CSV: identity_request_results.csv

python3 sni5gect_5g_attacks/\
    calculate_registration_reject_success_rate.py
# Result CSV: registration_reject_results.csv

python3 sni5gect_5g_attacks/\
    calculate_authentication_replay_success_rate.py
# Result CSV: authentication_request_results.csv
```

Figure 26 corresponds to **Figure 7: Attack Success Rates by Device, Distance, and Attack Type using srsRAN as legitimate gNB** and Figure 27 corresponds to **Figure 8: Attack Success Rates by Device, Distance, and Attack Type using Effnet as legitimate gNB**. To plot these two figures, please refer to the following commands:

```
cd /root/evaluation_results/sni5gect_5g_attacks
python3 update_attack_json_data.py
# Format the data for plot

python3 plot_srsran_result.py
# data_attack_success_rate_srsran.pdf

python3 plot_effnet_result.py
# data_attack_success_rate_effnet.pdf
```

A.5 Notes on Reusability

SNI5GECT provides a tool for DCI sniffing, message sniffing and message injection attacks. Additionally, this can be useful in the following tasks:

1. **Attack evaluation without rogue base station:** Unlike approaches that relies on a rogue base station that is assumed to have man-in-the-middle capabilities, the SNI5GECT framework enables evaluation of 5G attacks by passively sniffing and stealthily injecting messages into live sessions. This allows the researchers to evaluate their attacks under more realistic conditions.

2. **Throughput analysis:** This can be done by utilizing the steps shown in DCI sniffing evaluation. In particular, such an evaluation can provide the DCIs sent from the base station to the UE, from which we can learn the amount of traffics communicated between the base station and the UE. This makes it useful for throughput analysis over-the-air.

3. **Traffic analysis:** This can be accomplished by utilizing the message sniffing capability shown in message sniffing evaluation. For example, the user can analyze the traffic between the base station and the UE, to research on privacy tracking related topics.

4. **Anomaly detection:** This artifact provides a significant amount of PCAP from both the sniffer and the base station during the message injection attacks. To the best of our knowledge, this is the first open artifact providing such dataset with over-the-air injection attacks. This can be potentially used for anomaly detection tasks.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodol-

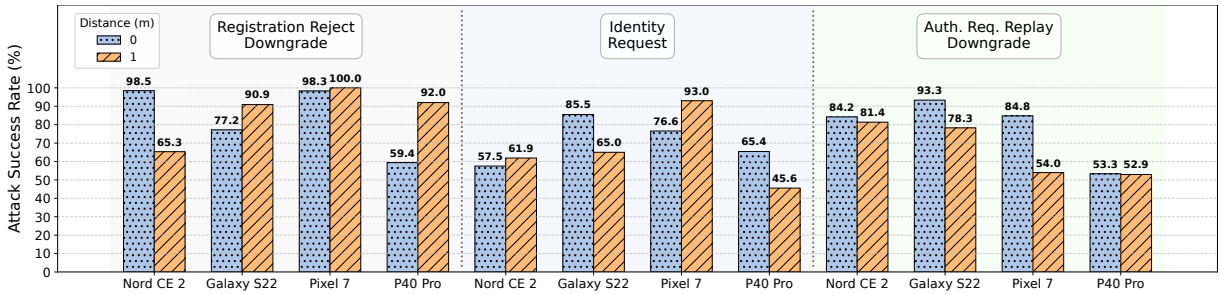

Figure 26: Attack Success Rates by Device, Distance, and Attack Type using **srsRAN** as legitimate gNB.

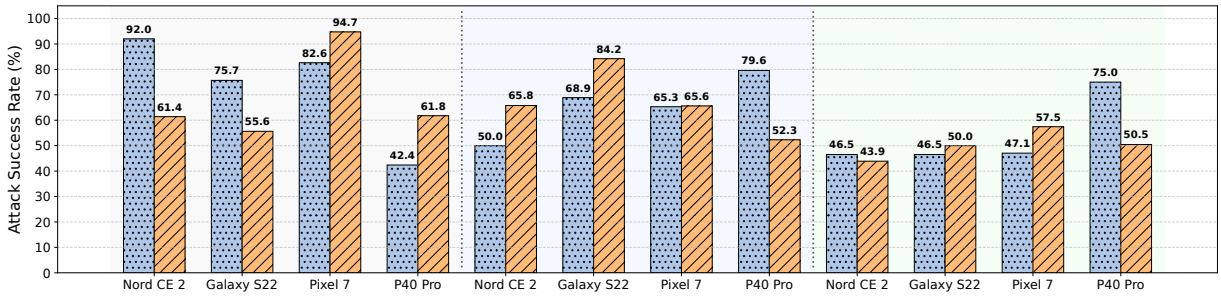

Figure 27: Attack Success Rates by Device, Distance, and Attack Type using **Effnet** as legitimate gNB.

ogy followed for the evaluation of this artifact can be found at
https://secartifacts.github.io/usenixsec2025/.