

AIRBUGCATCHER: Automated Wireless Reproduction of IoT Bugs

1. Abstract

This artifact showcases a Python implementation to automatically and reliably identify over-the-air (OTA) attack vectors and reproduce bugs (i.e., crashes and hangs) in Commercial-off-the-shelf (COTS) IoT devices, as proposed in AIRBUGCATCHER paper. We also provide scripts that aid the reproduction of experiments depicted in the paper by generating figures and logging terminal outputs. Nonetheless, the original logs used in the experiments depicted in the paper are included in this artifact. The estimated time to run all the evaluations on a Beelink SER5 Mini PC with an AMD Ryzen 7 5800H processor and 12 GB memory is 175 hours. For quick evaluation of AIRBUGCATCHER, access to a remote machine is provided. Such a machine contains the required software environment and attaches hardware devices via USB. Throughout the artifact evaluation, we show that (i) AIRBUGCATCHER complements the security testing pipeline by automatically reproducing bugs and generating a minimal proof of concept code, (ii) our design strategies are effective, and (iii) AIRBUGCATCHER is hardware and protocol agnostic.

2. Description & Requirement

2.1. How to access

We have provided a remote machine for easier artifact evaluation. The following script can be used to connect to it via SSH. We will put the password for the SSH connection in the comment section after artifact submission.

```
ssh root@18.143.110.88 -p 37891
```

The source code is available at <https://anonymous.4open.science/r/air-bug-catcher-E5C2>.

2.2. Hardware dependencies

The following hardware development boards are used during our evaluation:

- ESP32-Ethernet-Kit - Bluetooth Central
- ESP32-WROOM-32 - Vulnerable Bluetooth Target
- CYW920735Q60EVB-01 - Vulnerable Bluetooth Target
- USRP B210 Software Defined Radio - 5G Base Station
- OnePlus Nord CE 2 - Vulnerable 5G Target
- SIMCom SIM8202G-M.2 - Vulnerable 5G Target
- ALFA AWUS036AC - Wi-Fi access point
- ESP-WROVER-KIT - Vulnerable Wi-Fi Target

All the listed development boards are connected to the remote machine as shown in Figure 1.

2.3. Software dependencies

AIRBUGCATCHER is utilizing a fuzzing tool WDissector [1], [2] and the research team developing WDissector has made the tool publicly available at: <https://hub.docker.com/r/megarbelini/5ghoul>. Kindly note that there may be some occurrences of WDissector development research team or member (e.g. ASSET group etc.) in the terminal output during the evaluation and within the log files provided in this artifact. We are using Armbian 23.02.2 operating system with kernel 5.15.0 version. We are also using Podman 3.4.4 version to run the container, while Docker should also work. AIRBUGCATCHER is written and tested in Python 3.12.3.

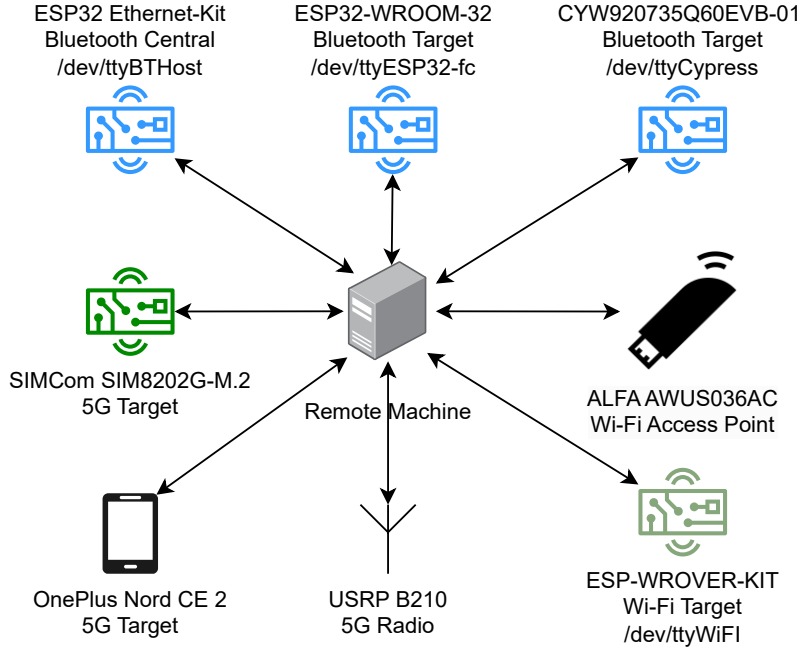


Figure 1: Attached Devices on Remote Machine

3. Setup

This section can be skipped if the provided remote machine is used. We recommend using a machine with at least 4 cores, 8GB memory and 40 GB disk space to run the experiments. The target devices (more details in the paper) need to be connected as well to run the experiments. While the experiments for the different targets can be conducted in parallel, more computation resources are needed in such case. Furthermore, we have also prepared a script to setup Podman and container on a Debian-based system.

```
wget "https://anonymous.4open.science/api/repo/air-bug-catcher-E5C2/file/\
  setup_container.sh?v=cdcc5417&download=true" -O setup_container.sh
bash setup_container.sh
```

After installing Podman and starting the container, we then enter the container and setup AIRBUGCATCHER as follows:

```
sudo podman exec -it airbugcatcher_eval bash

# The following commands are to be executed within container.
wget "https://anonymous.4open.science/api/repo/air-bug-catcher-E5C2/file/\
  setup_airbugcatcher.sh?v=817247ae&download=true" -O setup_airbugcatcher.sh
bash setup_airbugcatcher.sh
```

Some target devices need to be flashed with a certain vulnerable firmware version, and we have flashed the desired versions of firmware onto the target devices attached to the remote machine. The exact details on Flashing of firmware is out of the scope of this artifact, however, such instructions can be found in earlier work [1]. Moreover, in our evaluation of Wi-Fi implementation bugs, a kernel module is needed for the ALFA AWUS036AC dongle to function. To compile the kernel module, we can first copy its source code from the container to the host. Subsequently, the source code needs to be compiled and the module needs to be loaded on the host. The following steps illustrate the overall process:

```
# Execute on the host.
sudo podman exec -it airbugcatcher_eval tar -czf rtl8812au.tar.gz \
  -C /home/user/wdissector/src/drivers/wifi/ rtl8812au/
podman cp airbugcatcher_eval:/home/user/wdissector/rtl8812au.tar.gz /tmp
cd /tmp
tar -xzf rtl8812au.tar.gz
cd rtl8812au
sudo apt install build-essential
```

```
make
sudo bash load.sh
```


4. Evaluation Workflow

We plan to conduct our evaluations in the same order as the research questions (RQs) in the paper. Overall, the entire evaluation takes about 175 hours to complete across all devices. Such a duration stems from the large number of different experiments conducted on five subject devices. We note that the evaluations of different devices can be carried out in parallel. This can significantly speed up the evaluation process. We have also explained how to run the experiment on one device only in Section 4.6. In the following subsections, we explain how our evaluations in each RQ are conducted. We also illustrate the scripts to help run experiments and generate statistics and figures in all RQs on all five target devices.

In each RQ, the experiments are conducted in the following sequential order: ESP32-WROOM-32, CYW920735Q60EVB-01, OnePlus Nord CE 2, SIMCom SIM8202G-M.2 and ESP-WROVER-KIT. For the scripts and logs provided with this artifact, the five devices may also be referred as `esp32_bt`, `cypruss_bt`, `oneplus_5g`, `simcom_5g` and `esp32_wifi`, respectively for simplicity.

We provide a helper script to show the progress the evaluation process, which is located in file `run_rq_progress.sh`. Simply run this file in a new window (inside `/home/user/wdissector/modules/airbugcatcher` folder) when running either of `run_rq1.sh`, `run_rq3.sh`, `run_rq4.sh` and `run_rq5.sh`. The following step are provided for illustration and Figure 2 shows how the progress bar looks like.

```
# Run the following command in a new window other than
# the window where the experiment is running.
cd /home/user/wdissector/modules/airbugcatcher
bash run_rq_progress.sh
```



```
RQ1 experiment progress | 27% in 1s End in ~3s
Current running sub-experiment: cypruss_bt_rq1, progress 2/5
```

Figure 2: Screenshot of Progress Bar

Note: We note that the kernel module required by ALFA AWUS036AC Wi-Fi dongle is not very stable on the remote machine and can sometimes make the system unresponsive. The unresponsiveness usually happens after running experiments on the Wi-Fi target for some time and the issue will not happen when no Wi-Fi experiments are running. To ensure the completeness of our artifact evaluation, we still attach the Wi-Fi target to the remote machine. If the remote machine is unresponsive or inaccessible, kindly reach out to us and we will reboot the remote machine with utmost priority.

4.1. Experiment of RQ1

The purpose of this experiment is to validate the effectiveness of AIRBUGCATCHER against five devices employing three protocols, as shown in Section 5 (**RQ1**) of the paper. The fuzzing logs including packets trace and target crash logs (if available) are used in this experiment. All three kinds of fuzzed packets (i.e., mutation, replay and flooding) are included during the test scenario generation. This experiment runs on ESP32-WROOM-32, CYW920735Q60EVB-01, OnePlus Nord CE 2, SIMCom SIM8202G-M.2 ESP-WROVER-KIT sequentially and needs about 14 hours to finish.

```
cd /home/user/wdissector/modules/airbugcatcher
bash run_rq1.sh
```

When the experiments are running, there will be some simple output from terminal as shown in Figure 3a and we write the more detailed logs to files. These files can be found in folder `/home/user/wdissector/modules/airbugcatcher/eval_results/RQ1`. For example, the log file of device ESP32-WROOM-32 for RQ1 is shown in Figure 3b. We currently output experiment logs to files instead of standard output (stdout) except for some basic logs as shown Figure 3b. There are two kinds of log files: (i) detailed log file containing the output from both AirBugCatcher and WDissector, which may be very large (ii) dedicated log file containing only AirBugCatcher output, which is neat and only shows the information like which test case is executing and what bug is triggered. For instance, when running RQ1 experiments, the detailed log for Bluetooth target ESP32-WROOM-32 is stored in `/home/user/wdissector/modules/airbugcatcher/eval_results/RQ1/esp32_bt/esp32_bt_rq1.log` and the location of the dedicated log file is indicated in the first line of the detailed log. Sample logs for Bluetooth target ESP32-WROOM-32 in RQ1 can be found in the folder

/home/user/wdissector/modules/airbugcatcher/our_results/RQ1/esp32_bt/. We note that the five sub-experiments (one for each target device) have their individual AIRBUGCATCHER log file. Upon completion of all five sub-experiments, the statistics of RQ1 results (Table 4 in the paper) will be generated and shown in the terminal. This is illustrated in Figure 3c.

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# bash run_rq1.sh
Running RQ1
Running RQ1 on esp32_bt
```

(a) Example Terminal Output

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# cat eval_results/RQ1/esp32_bt/esp32_bt_rq1.log
AirBugCatcher log is saved in /home/user/wdissector/modules/airbugcatcher/logs/ae_08_28_12_22_zrzq.log
Start AirBugCatcher
Discovering crashes...
[Machine] Global Config Loaded: global_config.json
[Machine] Final Config Loaded: /home/user/wdissector/configs/bt_config.json
Enabling Core dump for this process: ulimit -c unlimited
sysctl: setting key "kernel.core_pattern": Read-only file system
WDissector Lib (Wireshark) 4.1.0 (v4.1.0rc0-673-g3baf3216acca)
Profile "WDissector" loaded
[Machine] Mapping Rules loaded: 9
[Machine] --> "btsdp"
[Machine] --> "bta2dp"
[Machine] --> "btavrcp"
```

(b) Example Log

```
===== RQ1 Statistics =====
Format INT (INT + INT) means <total number> (<number of crash> + <number of hang>)
esp32_bt:      unique bugs: 16 (14 + 2), reproduced: 16 (14 + 2), not reproduced: 0 (0 + 0)
               expected: 11 (9 + 2), unexpected: 5 (5 + 0)
               max # mutation: 3, max # replay: 2, # test case: 332, time: 6 hr 44 min

cypress_bt:    unique bugs: 6 (4 + 2), reproduced: 5 (3 + 2), not reproduced: 1 (1 + 0)
               expected: 4 (3 + 1), unexpected: 1 (0 + 1)
               max # mutation: 2, max # replay: 2, # test case: 46, time: 1 hr 26 min

oneplus_5g:    unique bugs: 14 (13 + 1), reproduced: 13 (13 + 0), not reproduced: 1 (0 + 1)
               expected: 13 (13 + 0), unexpected: 0 (0 + 0)
               max # mutation: 3, max # replay: 0, # test case: 82, time: 2 hr 10 min
```

(c) Example Statistics Output

Figure 3: Screenshots for Running RQ1 Experiments

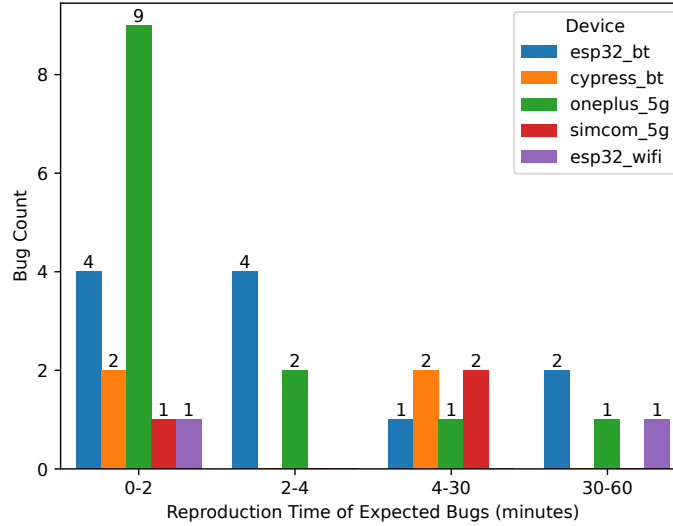
4.2. Experiment of RQ2

The purpose of this experiment is to investigate the efficiency of AIRBUGCATCHER to reproduce bugs. This is accomplished by measuring the time taken to reproduce the expected bugs. This experiment has the same setup as RQ1, meaning that it can utilize the logs from RQ1 instead of running the experiment again. Thus, to get the statistics for RQ2, the experiments of RQ1 may be conducted first, as detailed in Section 4.1. Then, the following scripts can be used to generate the RQ2 statistics, as shown in Figure 4.

```
cd /home/user/wdissector/modules/airbugcatcher
source .venv/bin/activate
python -m eval_scripts.analyze_rq2_results eval_results/RQ1
```

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# python -m eval_scripts.analyze_rq2_results eval_results/RQ1
Authorization required, but no authorization protocol specified
===== RQ2 Statistics =====
The figure for RQ2 is saved in eval_results/RQ1/rq2_figure.pdf, you can download via SFTP.
```

(a) Example Terminal Output



(b) Example Figure

Figure 4: Screenshots for Running RQ2 Experiments

4.3. Experiment of RQ3

The purpose of this experiment is to justify the effectiveness of different design options of AIRBUGCATCHER. We evaluate this from two aspects: (i) whether to include target device crash log in the analysis and (ii) types of fuzzed packets used during test scenario generation. We conduct this experiment on all five target devices. The results in the paper only include two devices, but the provided scripts allow the evaluation on all target devices. However, we note that some devices (e.g., CYW920735Q60EVB and SIMCom SIM8202G-M.2) do not produce logs when a bug occurs and some devices (e.g. OnePlus Nord CE 2 and SIMCom SIM8202G-M.2) are not fuzzed with replayed packets. Thus, not all devices are evaluated with all possible combination of AIRBUGCATCHER design options. Specifically, for Bluetooth target devices, ESP32-Ethernet-Kit has 2*3 sub-experiments and CYW920735Q60EVB-01 has 1*3 sub-experiments. For 5G target devices, OnePlus Nord CE 2 has 2*1 sub-experiments while SIMCom SIM8202G-M.2 has 1*1 sub-experiment. Finally, the Wi-Fi target device ESP-WROVER-KIT has 2*3 sub-experiments. In total, this experiment consists of 18 sub-experiments and needs about 86 hours to finish. The following script helps to run RQ3 experiments and the statistics will be printed to the terminal once experiments finish. The detailed logs can be found in folder /home/user/wdissector/modules/airbugcatcher/eval_results/RQ2. Some screenshots are shown in Figure 5 for illustration.

```
cd /home/user/wdissector/modules/airbugcatcher
bash run_rq3.sh
```

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# bash run_rq3.sh
Running RQ3
Running RQ3 on esp32_bt
```

(a) Example Terminal Output

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# python -m eval_scripts.analyze_rq3_results our_results/RQ3
===== RQ3 Statistics =====

esp32_bt w/o log has 29 unique bugs
esp32_bt experiments w/o log in RQ3 takes a Max. Time of 18 hr 40 min
esp32_bt w/o log, mutation only triggers      12 / 16 (Expected / Unexpected) bugs
esp32_bt w/o log, mutation + replay triggers  13 / 14 (Expected / Unexpected) bugs
esp32_bt w/o log, all triggers                15 / 14 (Expected / Unexpected) bugs
esp32_bt with log has 16 unique bugs
esp32_bt experiments with log in RQ3 takes a Max. Time of 7 hr 54 min
esp32_bt with log, mutation only triggers      9 / 6 (Expected / Unexpected) bugs
esp32_bt with log, mutation + replay triggers  10 / 6 (Expected / Unexpected) bugs
esp32_bt with log, all triggers                11 / 5 (Expected / Unexpected) bugs
```

(b) Example Statistics Output

Figure 5: Screenshots for Running RQ3 Experiments

4.4. Experiment of RQ4

The purpose of this experiment is to understand the sensitivity of AIRBUGCATCHER’s effectiveness based on the different execution parameters. The evaluated execution parameters include the maximum number of fuzzed packets in a test scenario (Max_{fp} in the paper) and the maximum execution time dedicated for a bug group (Max_{tt} in the paper). The performance is measured by the number of expected and unexpected bugs reproduced. This experiment consists of 30 sub-experiments and takes about 47 hours to finish. The scripts below help to conduct RQ4 experiments and the detailed logs can be found in folder /home/user/wdissector/modules/airbugcatcher/eval_results/RQ4. Some screenshots are shown in Figure 6 for illustration only.

```
cd /home/user/wdissector/modules/airbugcatcher
bash run_rq4.sh
```

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# bash run_rq4.sh
Running RQ4
Running RQ4 on esp32_bt
```

(a) Example Terminal Output

```
===== RQ4 Statistics =====

esp32_bt with Max_fpg=3 and Max_tt=10min triggers      9 expected bugs and      5 unexpected bugs.
esp32_bt with Max_fpg=3 and Max_tt=20min triggers     11 expected bugs and      5 unexpected bugs.
esp32_bt with Max_fpg=3 and Max_tt=40min triggers     11 expected bugs and      5 unexpected bugs.
esp32_bt with Max_fpg=1 and Max_tt=60min triggers      9 expected bugs and      6 unexpected bugs.
esp32_bt with Max_fpg=2 and Max_tt=60min triggers     11 expected bugs and      5 unexpected bugs.
esp32_bt with Max_fpg=3 and Max_tt=60min triggers     11 expected bugs and      5 unexpected bugs.
```

(b) Example Statistics Output

Figure 6: Screenshots for Running RQ4 Experiments

4.5. Experiment of RQ5

We intend to compare our AIRBUGCATCHER approach against a simple approach to reproduce bugs. Such an approach attempts to replay all TX (transmitted) packets from the fuzzing log. In this experiment, we utilize WDissector to help establish the wireless connection to the target devices and script execution part from AIRBUGCATCHER to run the replay scripts. The script execution timeout is set to one minute, which is the same value used in our other experiments. We run the replay script five times on each device. Overall, this experiment includes 25 sub-experiments and needs about 28 hours

to complete. The scripts below help to conduct RQ5 experiments and the detailed logs can be found in folder `/home/user/wdissector/modules/airbugcatcher/eval_results/RQ5`. Some screenshots are shown in Figure 7 for illustration.

```
cd /home/user/wdissector/modules/airbugcatcher
bash run_rq5.sh
```

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# bash run_rq5.sh
Running RQ5
Running RQ5 on esp32_bt
```

(a) Example Terminal Output

```
===== RQ5 Statistics =====

Device esp32_bt triggers 13 bugs in RQ5 trial 1.
Device esp32_bt triggers 16 bugs in RQ5 trial 2.
Device esp32_bt triggers 23 bugs in RQ5 trial 3.
Device esp32_bt triggers 13 bugs in RQ5 trial 4.
Device esp32_bt triggers 16 bugs in RQ5 trial 5.

Device cypress_bt triggers 0 bugs in RQ5 trial 1.
Device cypress_bt triggers 0 bugs in RQ5 trial 2.
Device cypress_bt triggers 0 bugs in RQ5 trial 3.
Device cypress_bt triggers 0 bugs in RQ5 trial 4.
Device cypress_bt triggers 0 bugs in RQ5 trial 5.
```

(b) Example Statistics Output

Figure 7: Screenshots for Running RQ5 Experiments

4.6. Experiment Execution on One Device

To run our tool on one device only, we need to first prepare the configuration files for the target, then run the experiments. The configuration files for different protocols are different and the related commands to change the configuration files can be found in file `/home/user/wdissector/modules/airbugcatcher/run_utils.sh`. Moreover, the scripts to run experiments on each device for each RQ can be found in the folder `/home/user/wdissector/modules/airbugcatcher/eval_scripts`.

For instance, to run RQ1 on Bluetooth target ESP32-WROOM-32 only, the following command can be used:

```
cd /home/user/wdissector/modules/airbugcatcher
cp eval_scripts/wdissector_configs/esp32_bt/global_config.json \
/home/user/wdissector/configs/global_config.json
cp eval_scripts/wdissector_configs/esp32_bt/bt_config.json \
/home/user/wdissector/configs/bt_config.json
source .venv/bin/activate
python -m eval_scripts.RQ1.esp32_bt.esp32_bt_rq1
```

5. Evaluation Results

We include the original logs from our evaluation conducted in this artifact. This can be found in folder `/home/user/wdissector/modules/airbugcatcher/our_results/`. The statistics in the paper can be derived from the logs and we have created scripts to aid the analysis. While preparing the scripts for the automation of analysis, we noted two minor errors in Table 4 of the accepted version (minor revision) of the paper. This was due to a manual approach involved in preparing the evaluation table from the detailed logs. Specifically, the total number of test cases (# Test Case in Table 4

of the paper) for ESP32-WROOM-32 and Cypress Board were reported to be 426 and 69, respectively. The correct numbers should be 332 and 46, respectively and we will amend this in the revised and camera-ready paper. We also note that the variation in the number of test cases does not affect the findings by AIRBUGCATCHER.

We also note that the reported the number of triggered bugs for ESP32-WROOM-32 in **RQ5** of the paper (five) is different from what we show in the results in this artifact (13, 16, 23, 13 and 16). The reason stems from the dynamic nature of wireless protocols and that the original paper has run the simple replay approach (baseline) only once. However, we created a script to run the simple replay approach five times, as required by the minor revision.

We have provided the following scripts to help generate the statistics for RQ1 to RQ5 automatically.

```
cd /home/user/wdissector/modules/airbugcatcher
source .venv/bin/activate

# Generate RQ1 statistics, Table 4 in the paper
python -m eval_scripts.analyze_rq1_results our_results/RQ1

# Generate RQ2 statistics, R2 statistics generation is utilizing RQ1 results
# Figure 6 in the paper
python -m eval_scripts.analyze_rq2_results our_results/RQ1

# Generate RQ3 statistics, Table 5 in the paper
python -m eval_scripts.analyze_rq3_results our_results/RQ3

# Generate RQ4 statistics, Table 6 in the paper
python -m eval_scripts.analyze_rq4_results our_results/RQ4

# Generate RQ5 statistics, Section 5, RQ5 in the paper
python -m eval_scripts.analyze_rq5_results our_results/RQ5
```

We have also provided the following script to analyze the fuzzing logs from target devices and generate statistics, as shown in Table 3 of the paper. The script may take some time to finish because some fuzzing logs are large. The example output of the scripts are presented in Figure 8. We note that we made some errors in indicating the field **Duration** for CYW920735Q60EVB-01 (4 hr 32 min in the paper vs 17 hr 10 min) and the field **# Replays** for ESP-WROVER-KIT (1387 in the paper vs 1837) in the Table 3 in the paper. While this will be corrected in our revised paper, such does not affect the findings by AIRBUGCATCHER.

```
cd /home/user/wdissector/modules/airbugcatcher
source .venv/bin/activate
python -m eval_scripts.gen_fuzzing_logs_stats
```

```
(.venv) root@f57a1304a829:~/wdissector/modules/airbugcatcher# python -m eval_scripts.gen_fuzzing_logs_stats
esp32_bt fuzzing log statistics: 7860 mutations, 12645 replays, 190 crashes, 15 hr 47 min
cypress_bt fuzzing log statistics: 2083 mutations, 2691 replays, 12 crashes, 17 hr 10 min
oneplus_5g fuzzing log statistics: 46992 mutations, 0 replays, 30 crashes, 13 hr 18 min
simcom_5g fuzzing log statistics: 9867 mutations, 0 replays, 5 crashes, 9 hr 59 min
esp32_wifi fuzzing log statistics: 2549 mutations, 1837 replays, 5 crashes, 1 hr 26 min
```

Figure 8: Fuzzing Logs Analysis

References

- [1] Matheus E. Garbelini, Vaibhav Bedi, Sudipta Chattopadhyay, Sumei Sun, and Ernest Kurniawan. BrakTooth: Causing havoc on bluetooth link manager via directed fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1025–1042, Boston, MA, August 2022. USENIX Association.
- [2] Zewen Shang, Matheus E. Garbelini, and Sudipta Chattopadhyay. U-fuzz: Stateful fuzzing of iot protocols on cots devices. *17th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2024.