ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE | ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL | ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security ’25 Artifact Appendix: BLuEMan: A Stateful Simulation-based Fuzzing Framework for Open-Source RTOS Bluetooth Low Energy Protocol Stacks

Wei-Che Kao
National Yang Ming Chiao Tung University

Yen-Chia Chen
National Yang Ming Chiao Tung University

Yu-Sheng Lin
National Yang Ming Chiao Tung University

Yu-Cheng Yang
National Yang Ming Chiao Tung University

Chi-Yu Li
National Yang Ming Chiao Tung University

Chun-Ying Huang
National Yang Ming Chiao Tung University

A Artifact Appendix

A.1 Abstract

Bluetooth Low Energy (BLE) is widely used for low-power, short-range communication but remains vulnerable in some implementations. Existing fuzzing methods often lack scalability and platform flexibility. To address this, we present BLuEMan, a simulation-based fuzzing framework combining a Real-Time Operating System (RTOS) with a software-based physical layer simulator. BLuEMan runs actual BLE stacks and emulates device interactions, enabling faster, scalable testing across platforms. It achieves fuzzing speeds up to 18x and 162.3x faster than existing simulation- and platform-based methods, respectively. It has uncovered four new CVE-assigned vulnerabilities, offering an efficient path for BLE security testing.

This artifact provides the source codes, scripts, and instructions to reproduce the primary evaluation results present in the paper, namely packet sending rate, fuzzing coverage, packet selection strategy, and vulnerability discovery.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact poses no security or privacy risks to artifact reviewers, testing runtime, or their users. The tools run everything in a container and can be run in parallel on multiple instances when the resources are affordable.

A.2.2 How to access

The artifacts are publicly accessible at the following platforms, including Zenodo, GitHub, and DockerHub.

- Zenodo: `https://doi.org/10.5281/zenodo.15601101`
- GitHub: `https://github.com/zoolab-org/blueman.artifact`
- DockerHub: `https://hub.docker.com/r/zoolab/blueman`

A.2.3 Hardware dependencies

Evaluating the artifact requires a machine equipped with an Intel x86-64 compatible CPU. It is because BLuEMan depends on BabbleSim, which currently supports only x86-64 CPU architecture. Our experiments use a server equipped with an Intel(R) Xeon(R) Gold 5118 CPU and 48 GB RAM, running Debian 12 Linux OS. To minimize the cost of reproducing the results, we omit the experiments that require special embedded hardware devices or depends on other papers’ artifacts.

A.2.4 Software dependencies

We pack all the required software dependencies into a single docker-based container. Please install a compatible docker runtime from either the official Docker website or the docker package that comes with a Linux distribution.

A.2.5 Benchmarks

None. No data is required by the experiments with the artifact reported in our paper.

A.3 Set-up

A.3.1 Installation

- Step 1. Clone the repository from GitHub or download the release files from zenodo.

```
git clone --recursive
  ↪  https://github.com/zoolab-org/blueman.artifact.git
```

- Step 2. Switch to the `./blueman.artifact` folder and ensure that the scripts have correct executable permission.

```
chmod +x ./build.sh ./run.sh
```

- Step 3. Build the container image that is required to run the experiments.

```
./build.sh
```

The script creates a container image named `blueman_demo`. It may take 10–60 minutes to build the image, depending on the hardware performance and network bandwidth. Alternatively, users may download our pre-built image from the DockerHub. The commands to pull the image and apply a correct tag for running the scripts are:

```
docker pull zoolab/blueman
docker tag zoolab/blueman blueman_demo
```

A.3.2 Basic Test

Once the container image is ready, use the '`run.sh`' script to run the experiments. The script comes with a few options to choose different runtime preferences, as shown in Figure 1. We merge multiple available options into a single line to save the spaces in the document.

```
Usage: ./run.sh <action> <mutator> [packet selection strategies]
↪  <execution duration> <output_dir>

Available actions:
  - gatt_write_peripheral, gatt_write_central
  - hr_peripheral, hr_central,
  - sm_pairing_peripheral, sm_pairing_central
  - le_credit_server, le_credit_client
  - ots_peripheral, otc_central

Available mutators:
  - field, afl, random

Available packet selection strategies (for field mutator only):
  - FIXED_PROB_10, FIXED_PROB_25, FIXED_PROB_50, FIXED_PROB_75,
  ↪  FIXED_PROB_100, SELECTIVE_25_75, SELECTIVE_75_25
  - RANDOM_PROB, MIXED_PROB

Available execution duration (in minutes):
  - 1, 10, 60, 120, 360, 720, 1440, 43200
```

Figure 1: The usage of the `run.sh` script.

Note that the `run.sh` script preserves outputs in the folder specified in the last argument. Please ensure that the folder exists in the filesystem. If multiple tests are run in parallel, please remember to use a different directory for each test.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): BLuEMan runs much faster than other existing approaches. This is proven by the experiment (E1), where we show that BLuEMan achieves a higher packet sending rate, as presented in Section 6.1. Note that the numbers for SweynTooth and BTFuzz are reported from the corresponding papers.

(C2): The field-aware mutation approach achieves higher coverage than random mutation and AFL-only mutators. This is proven by the experiment (E2), where we show the reported coverage for each evaluated approach. Due to the nature of randomness involved in the fuzzer implementations, the reported coverage numbers may differ for each invocation. However, the trends should be aligned with the results presented in Section 6.3.

(C3): The packet selection strategy shows diverse performance in terms of coverage exploration. This is proven by the experiment (E3), where we show the reported coverage for each evaluated packet selection strategy. Due to the nature of randomness involved in the fuzzer implementations, the reported coverage numbers may differ for each invocation. The results should be similar to those presented in Section 6.4.

(C4): BLuEMan has the capability to uncover new bugs (CVE-2024-3332) in the Zephyr BLE protocol stack. This is proven by the experiment (E4). Because all the CVEs are reported from different versions of Zephyr, we choose the fastest one for the reproduction process.

A.4.2 Experiments

(E1): *[Packet Rate] [30 human-minute + 1 compute-hour + 20GB disk]: Run BLuEMan and observe the packet sending rates.*
How to: Run the fuzzing process and check the status message output from the tool.
Preparation: Follow the steps described in Section A.3 to create the required docker image.
Execution: Run BLuEMan using the command:

```
mkdir /tmp/output-folder
./run.sh ots_peripheral field FIXED_PROB_50 10
↪  "/tmp/output-folder"
```

Please ensure that `/tmp/output-folder` can be mounted into the docker instance and is writable by the instance.
Results: A sample output message is shown below. The `total_elapse` field shows the elapsed time (in microseconds) for running the fuzzer, and the `total_packet_count` shows how many packets have been sent during the fuzzing process.

```
start_time: 1748974762595148, cur_time:
↪  1748974825155642, total_elapse: 62560494,
↪  round_elapse: 2676575
round 28, exec_counts 280, current corpus:
↪  ./output/seed//id_00003248_0000000000000065_000045,
↪  attack_error_count 0, crash_count 0,
↪  timeout_count: 0, coverage: 3593, max coverage:
↪  4223, queue size: 88, total_packet_count: 26540
```

(E2): *[Mutation Approach] [1 human-hour + n×(1∼24) compute-hour, where n is the number of traces collected for plotting the results + 60GB disk]: Run BLuEMan and measure the coverage for different mutation approaches.*
How to: Run the fuzzing process and collect the results from the corresponding output folders. Please note that in our paper, each plotted curve represents three 24-hour runs. Users may run the experiments for a shorter time to facilitate the reproduction process and then check the output results.
Preparation: Follow the steps described in Section A.3 to create the required docker image.

Execution: We recommend to run BLuEMan using the simplified command:

```
./auto_gen_rq3.sh <action> <duration>
```

The command invokes BLuEMan, uses `<action>` as the interacting app for generating testing corpora, and runs the testing for `<duration>` minutes. The available `<action>` and `<duration>` are exactly the same as those listed by the '`run.sh`' script. Note that the '`./auto_gen_rq3.sh`' command utilizes 4 CPU cores in parallel. Fuzzing is a computationally intensive process, and execution speed may vary depending on available computing resources. Users may selectively run commands as specified in the `EVALUATION_RQ3.md` document to suit their environment better.

Results: An example output from a one-minute test of the `sm_pairing_central` app is shown below.

```
$ ./auto_gen_rq3.sh sm_pairing_central 1
==> Running: sm_pairing_central field
==> Running: sm_pairing_central afl
==> Running: sm_pairing_central random
==> Generating coverage plot for sm_pairing_central
Written /root/plot/csv/afl.csv
Written /root/plot/csv/field.csv
Written /root/plot/csv/random.csv
Reading field.csv
Reading random.csv
Reading afl.csv
Coverage plot saved
==> Coverage plot generation completed for
 ↪ gatt_write_peripheral, duration time: 1 minutes.
 ↪ The plot is saved in
 ↪ sm_pairing_central_1_rq3/eval_sm_pairing_central/cov/
```

The generated PDF file is stored in the `sm_pairing_central_rq3/eval_sm_pairing_central/cov/` directory on the host.

(E3): *[Packet Selection Strategy] [1 human-hour + n×(1∼2) compute-hour, where n is the number of traces collected for plotting the results + 60GB disk]: Run BLuEMan and measure the coverage for different packet selection strategies.*

How to: Run the fuzzing process and collect the results from the corresponding output folders. Please note that in our paper, each plotted curve represents 2 hours of testing. Users may run the experiments for a shorter time to facilitate the reproduction process and then check the output results.

Preparation: Follow the steps described in Section A.3 to create the required docker image.

Execution: We recommend to run BLuEMan using the simplified command:

```
./auto_gen_rq4.sh <action> <duration>
```

The command behaves similar to '`./auto_gen_rq3.sh`'. Similarly, users may selectively run commands as specified in the `EVALUATION_RQ4.md` document to suit their environment better. Note that the '`./auto_gen_rq4.sh`' command utilizes 10 CPU cores in parallel.

Results: An example output from a one-minute test of the `ots_peripheral` app is shown below.

```
$ ./auto_gen_rq4.sh ots_peripheral 1
==> Running: ots_peripheral FIXED_PROB_10
... (omitted for saving spaces)
==> Generating coverage plot for ots_peripheral
Written /root/plot/csv/field_FIXED_PROB_10.csv
... (omitted for saving spaces)
Reading field_FIXED_PROB_10.csv
... (omitted for saving spaces)
Coverage plot saved
==> Coverage plot generation completed for
 ↪ ots_peripheral, , duration time: 1 minutes. The
 ↪ plot is saved in
 ↪ ots_peripheral_1_rq4/eval_ots_peripheral/cov/
```

The generated PDF file is stored in the `ots_peripheral_rq4/eval_ots_peripheral/cov/` directory on the host.

(E4): *[Vulnerability Discovery] [30 human-minute + 0.5 compute-hour + 20GB disk]: Run BLuEMan and wait for reported crashes due to triggered vulnerabilities.*

How to: Run the fuzzing process and observe the outputs from the console logs and the corresponding output folders.

Preparation: Follow the steps described in Section A.3 to create the required docker image.

Execution: Run BLuEMan using the command:

```
mkdir /tmp/output-folder
./run.sh gatt_write_central field FIXED_PROB_50 60
 ↪ /tmp/output-folder
```

The command uses `gatt_write_central` as the interacting app for generating testing corpora. Note that the CVE-2024-3332 vulnerability can only be triggered using the app.

Results: An example output from a 60-minute test of the `gatt_write_central` app is shown below.

```
$ mkdir /tmp/output-folder
$ ./run.sh gatt_write_central field FIXED_PROB_50 60
 ↪ /tmp/output-folder
...
start_time: 1748962190058918, cur_time:
 ↪ 1748962489441826, total_elapse: 299382908,
 ↪ round_elapse: 17087627
round 21, exec_counts 210, current corpus:
 ↪ ./output/seed//id_00001858_0000000000000105_000070,
 ↪ attack_error_count 5, crash_count 1,
 ↪ timeout_count: 58, coverage: 3040, max coverage:
 ↪ 4531, queue size: 105, total_packet_count: 17253
```

When a crash is detected, the total number of detected crashes is shown in the `crash_count` field.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.