FRET Artifact Evaluation

This document provides instructions for reproducing the experimental results from the paper "Dynamic Fuzzing-Based Whole-System Timing Analysis".

Description

Expected Time: 5 days (configurable)
Hardware Requirements: 64 cores, 256-512GB RAM (recommended), minimum 8 cores, 32-64GB RAM
Software Requirements: Linux x86_64, Nix package manager

Claims Supported by Artifact

If you run the benchmarks as described below, it will produce the following files, which correspond to figures in the paper (link to the paper).

1. **Figure 3**: (Files: `sql_waters_seq_bytes`, `sql_polycopter_seq_dataflow_full`) This scenario is about mutating just input values. While multiple techniques find the worst case for both scenarios, FRET is the fastest to reach the maximum, particular in the second case. FRET also achieves the highest median result.

2. **Figure 4**: (Files: `sql_waters_seq_int`,`sql_release_seq_int`) This scenario is about mutating just interrupt times. While multiple techniques find the worst case for the second scenario, FRET achieves the highest response time on the first one.

3. **Figure 5**: (Files: `sql_release_seq_full`, `sql_waters_seq_full`) This scenario is about mutating both kinds of inputs simultaniously. Only FRET achieves the worst possible time when looking at the median results of the first scenario. For the second one, FRET alone reaches the highest response times. Thid demonstrates, that FRET's advantage over other techniques is particularly pronounced when both kinds of inputs are compared.

4. **Figure 6**: (File: `all_tasks`) This is a comparison of FRET's advantage over the best other technqiues on each task of the `waters` scenario. FRET ends up above every other technique on each task, validating the comparison in fig. 5 b).

Getting Started

Option 1: VirtualBox Images (Recommended)

Download our ready-made VM image from https://sys-sideshow.cs.tu-dortmund.de/downloads/rtss25/fret.ova

- **VM Configuration**: Allocate as much RAM as possible (minimum 32GB, recommended 512GB)
- **CPU Allocation**: One core per 4-8GB of RAM (recommended 64 cores, 256-512GB RAM)
- **Disk Space**: At least 100GB free space for results
- **Login**: Username: `osboxes.org`, Password: `osboxes.org`
- **Open a terminal in ~/FRET**
  - Load the environemnt using `direnv relaod` or `nix develop`
- **Ensure you have the right version**: `git checkout RTSS25-AE && git submodule update --init`

Option 2: Setup From Scratch

Prerequisites: Linux x86_64 system with Nix package manager installed

1. Clone the repository:

```
git clone https://git.cs.tu-dortmund.de/SYS-OSS/FRET
cd FRET
git checkout RTSS25-AE
git submodule update --init
```

2. Install Nix (if not already installed):

```
curl -L https://nixos.org/nix/install | sh
source ~/.nix-profile/etc/profile.d/nix.sh
```

3. Enter the development environment:

```
nix develop  # or nix-shell for older Nix versions
```

Option 3: Docker Setup

1. Clone the repository:

```
git clone --recursive https://git.cs.tu-dortmund.de/SYS-OSS/FRET
cd FRET
git checkout RTSS25-AE
git submodule update --init
```

2. See Docker/README.md

System Requirements

Minimum Configuration

- **CPU**: 8 cores (Intel/AMD x86_64)

- **Memory**: 32GB RAM
- **Storage**: 100GB free disk space
- **OS**: Linux distribution with Nix support

Recommended Configuration

- **CPU**: 64 cores with hyperthreading
- **Memory**: 256-512GB RAM (allows parallel execution of all configurations)
- **Time for a full evaluation**: 5 days

Building and Installation

Initial Setup

```
./one_time_setup.sh
```

This script builds the default configuration of FRET, along with target systems and additional tools for benchmarking. The script will: - Build QEMU with FRET patches - Compile target FreeRTOS applications - Build the FRET fuzzer and all dependencies - Set up the evaluation environment

Verification of Installation

After setup completes, verify the installation:

```
# Check FRET binary
cd LibAFL/fuzzers/FRET/benchmark
target/fret --help

# Check target systems
ls build
```

Step-by-Step Instructions (Optional)

See README.md for reference.

Manual Usage (Optional)

You can start using FRET manually. It requires the following inputs:

- **-k**: a FreeRTOS Kernel image
- **-c**: a csv file with configuration parameters per target kernel (LibAFL/fuzzers/FRET/benhcmark/target_symbols.csv)

```
cd LibAFL/fuzzers/FRET
# Help for arguments
cargo run -- --help
# Example
```

```
mkdir -p $TMPDIR
export DUMP=$(mktemp -d)
dd if=/dev/random of=$DUMP/input bs=8K count=1
# fuzz for 10 seconds
cargo run -- -k benchmark/build/waters_seq_full.elf \
    -c benchmark/target_symbols.csv \
    -n $DUMP/output -ta fuzz -t 10 --seed 123456
# Produce a trace for the worst case found
cargo run -- -k benchmark/build/waters_seq_full.elf \
    -c benchmark/target_symbols.csv \
    -n $DUMP/show -trg showmap -i $DUMP/output.case
# plot the result
gantt_driver $DUMP/show.trace.ron
# view the gantt chart
open $DUMP/show_job.html
```

Run the Evaluation

```
# Feel free to read the script
./run_eval.sh
```

This script will reproduce all figures 3-6 in the eval section of the paper. You can edit the environemnt variables at the top to change the following parameters:

- `CORES`: The number of (physical) cores of the VM / fuzzers running in parallel. You will need about 8GB of RAM per fuzzer.
- `RUNTIME`: Time spent on each fuzzing run in seconds (the default is 24h). 8h should be sufficient to see results similar to the paper.
- `TARGET_REPLICA_NUMBER`: The number of replicas for each regular configuration.
- `RANDOM_REPLICA_NUMBER`: The number of replicas for configurations with random fuzzing. These usually deviate very little from each other and thus can be reduced without affecting the results.
- `MULTIJOB_REPLICA_NUMBER`: The number of replicas for the figure that compares all techniques for each task of a system. This evluation consists of many configurations, so you can reduce this numer to save a lot of time.

For complete reproduction of paper results you can use the following configuration, which takes about 5 days on a 64 core machine:

```
# in run_eval.sh
export CORES=64 # Number of physical cores
export RUNTIME=86400 # 24 hours in seconds
export TARGET_REPLICA_NUMBER=12
```

```
export RANDOM_REPLICA_NUMBER=3
export MULTIJOB_REPLICA_NUMBER=3
```

Results

All results can be found in `LibAFL/fuzzers/FRET/benchmark` inside a directory `eval_xx-xx-xx`. The content should be the following:

- Plots inside the top-level of the directory. You can compare them to the paper according to the hints under "Claims Supported by Artifact" above.
- A directory `timedump`, which contains subdirectories for each fuzzer.
  - Insude you find files for configuration with different seeds.
  - `.time` contain response times of each execution.
  - `.case` contains the worst case found by the fuzzer.
  - `.trace.ron` contains tracing data of the worst case. Such data can be plotted into agantt chart using the tool `gantt_driver`

Please note that fuzzing is a stochastic process and therefore, some of the results may exhibit random variation from the figures in the paper.

An archive of our results is also provided under https://sys-sideshow.cs.tu-dortmund.de/downloads/rtss25/results.zip.

Troubleshooting

Common Issues

Out of Memory Errors

- Reduce `CORES` parameter to match available RAM (up to 8GB per core)
- Consider using fewer relicas per configuration

Build Failures

```
# Clean and rebuild
cd LibAFL/fuzzers/FRET
cargo clean
```

Missing Dependencies

```
# Ensure Nix environment is active
nix develop
# Or for older Nix versions
nix-shell
```

Incomplete Results

- Check that snakemake completed without errors
- Verify all target binaries were built successfully
- Ensure sufficient disk space (100GB+)
- If plots appear incomplete remove `bench.sqlie` from the benchmark directory.
- If all else fails: `snakemake -c1 full_clean`

Artifact Structure

Directory Layout

```
FRET/
+-- LibAFL/fuzzers/FRET/           # Main FRET fuzzer implementation
|   +-- src/                       # Source code
|   +-- benchmark/                 # Evaluation framework
+-- FreeRTOS/FreeRTOS/Demo/CORTEX_M3_MPS2_QEMU_GCC # Our FreeRTOS Demos
+-- one_time_setup.sh             # Initial setup script
+-- run_eval.sh                   # Main evaluation script
+-- AE.md                        # This document
```

Key Components

- **FRET Fuzzer**: Core fuzzing engine with RTOS-aware features
- **Target Applications**: Real-world FreeRTOS applications for testing
- **Evaluation Framework**: Snakemake-based automation for reproducible experiments

Customization

Adding New Target Applications

1. Add target configuration to `benchmark/target_symbols.csv`
2. Update `benchmark/Snakefile` with new target rules

Further information

Please consult README.md and LibAFL/fuzzers/FRET/ARCH.md

Contact Information

For questions about the artifact or issues during evaluation, please contact: - alwin.berger@tu-dortmund.de

License