HOEDUR: Embedded Firmware Fuzzing using Multi-Stream Inputs

Tobias Scharnowski and Simon Wörner, *CISPA Helmholtz Center for Information Security;* Felix Buchmann, *Ruhr University Bochum;* Nils Bars, Moritz Schloegel, and Thorsten Holz, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity23/presentation/scharnowski

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Hoedur: Embedded Firmware Fuzzing using Multi-Stream Inputs

Tobias Scharnowski[1], Simon Wörner[1], Felix Buchmann[2], Nils Bars[1],Moritz Schloegel[1], and Thorsten Holz[1]
[1]CISPA Helmholtz Center for Information Security
[2]Ruhr-Universität Bochum

A Artifact Appendix

A.1 Abstract

Hoedur is a rehosting-based firmware fuzzer that introduces a multi-stream input format that extends the concepts contained in previous rehosting-based fuzzing works such as P2IM, uEmu, and Fuzzware. This format helps the fuzzer to mutate its input effectively.

We provide GitHub repositories containing data and scripts to install our prototype, to automatically reproduce our new firmware targets/configurations, and to auto-generate configurations to rerun our experiments based on the available computation resources. Our experiments run in Docker containers. These can be rebuilt via the provided scripts, or prebuilt Docker containers can be used to rerun our experiments. To facilitate the reproduction, we provide an alternative experiment profile that reduces the CPU requirements while still, in our view, supporting our major claims.

In our experiments, we compare our fuzzer against itself in different configurations (with and without multi-stream inputs) and against the related work Fuzzware given that this tool outperformed P2IM and uEmu in an evaluation. We test Hoedur's speed and reliability in finding known bugs, producing code coverage, and finding previously unknown vulnerabilities.

A.2 Description & Requirements

Access Privileges. The user under which the experiments are supposed to run requires access to Docker. `root` privileges are also required to configure the environment for running the related work `Fuzzware` (see scripts/fuzzware/set_limits_and_prepare_afl.sh).

OS and software. Our fuzzing experiments require Linux hosts. We recommend a recent Ubuntu Server installation. On these hosts, Python 3, rsync, and Docker must be installed and the user must be part of the `docker` group.

Computation Resources. Our original experiments took around 30 CPU years to run. To provide a less resource intensive option, we created an alternative experiment profile for the artifact evaluation that should reproduce our major findings in around 3 CPU years worth of computation time. If changes to these configurations are desired, we provide a configuration format that allows customizing the experiment duration and the repetition counts.

To rerun the experiments within 15 days, the equivalent of the following computation resources will be required:

1. Reduced experiment: 2 servers, 50 physical cores each.
2. Full experiment: 20 servers, 50 physical cores each.

Storage. In total, the experiments will produce a maximum of 300 GB of data which needs to be pulled onto one server to compute and aggregate all metrics.

A.2.1 Security, privacy, and ethical concerns

Running the experiments will require root access and access to Docker on the reproduction machines. No security mechanisms are disabled, and the machines are not exposed to additional security risks.

Hoedur has found previously-unknown vulnerabilities which have been responsibly disclosed to the respective vendors. Our prototype may find additional vulnerabilities, possibly even in the provided targets. Please handle these findings responsibly as well.

A.2.2 How to access

The code and experiment data are available on GitHub under github.com/fuzzware-fuzzer/hoedur-experiments. We created the tag `sec23-ae-accepted` as a stable reference.

From there, the complete reproduction is done in Docker containers. These can be built from Dockerfiles, or our prebuilt containers can be used. The repository README contains all information required to build and run the experiments.

A.2.3 Hardware dependencies

None (no special hardware, for general compute requirements, see Descriptions & Requirements above).

A.2.4 Software dependencies

To run the experiments, a Linux distribution is required. We recommend a recent version of Ubuntu Server. On these hosts, Python 3, rsync, sudo, and Docker need to be installed and set up for the user. For the optional step of rebuilding our target binaries, a small number of Python packets need to be installed via the provided requirements.txt file.

A.2.5 Benchmarks

As a part of our firmware fuzzing targets, we use the firmware binaries previously published in the P2IM, uEmu, and Fuzzware experiments. We include copies of these binaries in the hoedur-experiments GitHub repository. Section 6.1 of the paper explains that we adapt targets from the coverage measurement data set by applying binary patches. You can find these patches along with comments and reproduction scripts at 02-coverage-est-data-set/binary-patching.

A.3 Set-up

All of our experiments use Docker as an execution environment. Docker needs to be installed on the system, and the user under which the experiments will be run needs to be added to the `Docker` group. Also, ensure that Python 3 is installed on the system. In case you would like to rebuild our target binaries using the reproduce_targets_and_configs.py script, `pip` is required.

For each host on which the related work FUZZWARE might be run, prepare the system to run FUZZWARE by running scripts/fuzzware/set_limits_and_prepare_afl.sh as `root`.

The hosts to be used for the experiment are configured in experiment-config/available_hosts.txt (see also experiment-config/README.md) and looks like the following:

```
my-host-1 <number_of_physical_cores_1>
my-host-2 <number_of_physical_cores_2>
```

For example, if everything should be run and metrics be generated on the same, single host with 70 cores, the configuration is as simple as:

```
localhost 70
```

If two servers (`my-host-1` and `my-host-2`) with 50 cores each are to be used, and the results should be aggregated on `host-1`, then the configuration would look like the following:

```
localhost 50
my-host-2 50
```

To allow some of the experiment scripts to access each host via `SSH`/`rsync`, create an SSH config entry in `~/.ssh/config`:

```
Host my-host-1
    Hostname 12.34.56.78
    User user
    IdentityFile ~/.ssh/my_privkey.key
    AddKeysToAgent yes
```

A.3.1 Installation

To install Hoedur, first, make the Docker containers available to your system. To re-build the Docker containers, run install.py. To obtain the pre-built Docker containers, run `./install.py --prebuilt`.

As an optional step, you may wish to reproduce our new target firmware binaries and auto-converted configurations and bug detection hooks. To reproduce these, run reproduce_targets_and_configs.py.

A.3.2 Basic Test

To make sure that the basic installation has been successful, first run:

```
./scripts/check_install.py
```

You may also like to run one of the bug reproducing inputs provided in the experiments repository. To run the reproducer for CVE-2022-41873, run:

```
cd ./04-prev-unknown-vulns/repro-run-scripts
./run_reproducer_CVE-2022-41873.sh
./run_reproducer_CVE-2022-41873.sh --trace
```

This should indicate that the bug `new-Bug-CVE-2022-41873` has been triggered by the reproducing input.

A.4 Evaluation workflow

Our experiments test four different aspects of Hoedur: The ability to trigger bugs, to produce code coverage, to make previously unavailable mutation types (by the example of dictionaries) effective, and to find previously unknown bugs.

We organized the hoedur-experiments repository in such a way that one subdirectory corresponds to one section in the evaluation of the paper. We provide scripts to run these experiments, and try to facilitate this process by providing the additional utility generate_host_run_config.py which accepts a description of computation resources (SSH-available hosts as well as the number of cores to use on each host) and generates run scripts for each host that together allow reproducing our results. We also created experiment profiles that reproduce the major insights of our experiments while reducing the CPU requirements.

A.4.1 Major Claims

(C1): HOEDUR outperforms the state of the art reference FUZZWARE in terms of its ability to find bugs quickly. This is proven by the experiment *Bug Finding Ability* described in Section 6.2 in the paper and the results of which are reported in Table 1 and Table 2. This corresponds to the directory 01-bug-finding-ability in the artifact.

(C2): HOEDUR is either on par with or outperforms the state of the art reference FUZZWARE and its version SINGLE-STREAM-HOEDUR (which does not use our multi-stream input representation) in terms of code coverage. This is proven by the experiment *Code Coverage: Established Data Set* described in Section 6.3 in the paper, whose results are reported in Figure 6 and Figure 8. This corresponds to the directory 02-coverage-est-data-set in the artifact.

(C3): Our multi-stream input representation allows using dictionary mutations effectively. These dictionaries do not provide a significant benefit when using a flat binary input format. This is proven by the experiment *Advanced Mutations via Dictionaries* described in Section 6.4 in the paper, whose results are reported in Figure 7. This corresponds to the directory 03-advanced-mutations in the artifact.

(C4): HOEDUR is able to find previously unknown vulnerabilities. This is proven by the experiments *Bug Finding Ability* and *Finding Unknown Vulnerabilities* described in Section 6.2 and Section 6.5 in the paper. The results are reported in Table 2 and Table 3. This corresponds to the directories 01-bug-finding-ability and 04-prev-unknown-vulns in the artifact.

A.4.2 Experiments

In summary, the experiment workflow is the following:

1. Configure the experiment hosts via available_hosts.txt.
2. Install and setup: install.py and set_limits_and_prepare_afl.sh on each experiment host.
3. Generate the experiment run configurations: generate_host_run_config.py. Upload via `--upload`.
4. Run experiment configs on the respective experiments hosts (make sure to use `tmux` or similar).
5. Pool data together: sync_experiment_data.py
6. Generate metrics: compute_metrics.py.
7. Inspect the results (see per-experiment description).

Now we include more details on each step. Steps 1 to 6 need to be taken once to generate the data corresponding to the results reported in our paper for the different experiments. From there, the final experiment-specific step is to check the relevant output for each experiment manually.

We provide the utilities available_hosts.txt, generate_host_run_config.py and run_experiment.py to help with scheduling and running the fuzzing experiments for **E1**, **E2**, **E3** and **E4**. Before running these experiments, we assume that the Docker containers are already installed on each host on which any parts of the experiments are supposed to be run (install.py), and that each host is available via an SSH configuration (see Section A.3.1) and set up to run the reference fuzzer FUZZWARE (scripts/fuzzware/set_limits_and_prepare_afl.sh).

The first manual step after this installation is to determine the hosts on which to run the experiments and to configure them in available_hosts.txt. The fuzzing run experiment configurations can then be generated using generate_host_run_config.py. The experiment configuration `YAML` files will be located in experiment-config/host-run-configs as `<hostname>.yml`. These configurations need to be copied to the corresponding hosts. Uploading the configurations can be done either manually or by running:

```
./generate_host_run_config.py --upload
```

Given the experiment configuration file on a host, the experiment can be started via:

```
./run_experiment.py <HOST_CONFIG>.yml
```

Please note that these are long-running experiments, such that `tmux` or similar tools should be used to run them. After running the fuzzers to completion, the workflow will be to synchronize all results into the `hoedur-experiments` directory of the main host via sync_experiment_data.py and running the post processing script compute_metrics.py to generate all metrics.

From here, the generated results need to be confirmed by inspecting the `results` directory of each experiment. We document which files within the `results` directory contain the data from our paper for each experiment in the descriptions below. You may also refer to the README of each experiment directory of the published hoedur-experiments repository for more info on the available data.

(E1): *[01-bug-finding-ability] [60 human-minutes + 750 compute-days + max 100GB disk]:*

How to: Reproduce claim **C1** / Section 6.2 in the paper.

Preparation: Generate run configurations using generate_host_run_config.py (see above) and copy them to the respective hosts. Make sure the required Docker containers are installed and the hosts are set up on all experiment hosts via install.py and set_limits_and_prepare_afl.sh (see above).

Execution: Run the fuzzers using the run scripts. Ensure a stable execution environment when running via SSH, such as `tmux`. After the fuzzers have finished executing, sync the results via sync_experiment_data.py and then compute all remaining metadata via compute_metrics.py.

Results: After generating all metadata for the completed experiments, the results can be found in 01-bug-finding-ability/results/bug-discovery-timings. In

this directory, the bug raw discovery timing data can be found in `timings.txt` and `timings.json` and represent the data also contained in Table 1 in the paper. The table representations as present in the paper can be found in the `results` directory under table_1_cve_discovery_timings.tex and table_2_add_bugs_discovery_timings.tex.

The overall results should resemble those in the paper: It is expected that HOEDUR finds bugs more quickly than FUZZWARE overall. Keep in mind that due to the non-deterministic nature of fuzzing, these numbers may vary. Also keep in mind that based on the experiment profile used, the number of iterations and fuzzing duration may be altered from the original experiments. For example, the default experiment configuration shortens the fuzzing runs from the original 15-day duration. For the exact numbers of each experiment configuration/profile, please refer to the `experiment-config` README. To fully rerun the original configuration, please use the corresponding, pre-supplied experiment configuration/profile name `full-eval`.

(E2): *[02-coverage-est-data-set] [60 human-minutes + 180 compute-days + max 100GB disk]:*
How to: Reproduce claim **C2 /** Section 6.3 in the paper.
Preparation: Already done in **E1**.
Execution: Already done in **E1**.
Results: After generating all metadata for the completed experiments, the results can be found in 02-coverage-est-data-set/results/coverage. The raw plot data can be found in the directory tree in compressed format under `charts/<fuzzer_name>`. It contains the raw data also shown in Figure 6 in the paper. The graphical plot representations as present in Figure 6 and Figure 8 in the paper can be found in the parent directory under figure_6_baseline_coverage_plot.pdf and figure_8_appendix_baseline_coverage_plot.pdf, respectively. It is expected that HOEDUR is on par with or generates more coverage than FUZZWARE and SINGLE-STREAM-HOEDUR. See also the general disclaimer about fuzzing experiment variance and experiment configurations/profiles under item **Results** of **E1**.

(E3): *[03-advanced-mutations] [60 human-minutes + 40 compute-days + max 50GB disk]:*
How to: Reproduce claim **C3 /** Section 6.4 in the paper.
Preparation: Already done in **E1**.
Execution: Already done in **E1**.
Results: After generating all metadata for the completed experiments, the results can be found in 03-advanced-mutations/results/coverage. The raw plot data can be found in the directory tree in compressed format under `charts/<fuzzer_name>`. It contains the raw data also shown in Figure 7 in the paper. The graphical plot representations, as present in Figure 6 and Figure 8 in the paper, can be found in the parent directory under figure_7_baseline_dict_coverage_plot.pdf. It is expected that HOEDUR+DICT performs best overall, while HOEDUR+DICT does not provide the same level of improvement over its single-stream version.

(E4): *[04-prev-unknown-vulns] [60 human-minutes + 50 compute-days + max 10GB disk]:*
How to: Reproduce claim **C4 /** Section 6.1 and Section 6.5 in the paper.
Preparation: Already done in **E1**.
Execution: Already done in **E1**.
Results: For this experiment we provide pre-extracted samples of crashing inputs that trigger each reported bug in the repository. For a full overview of the reported bugs, see the tables in 04-prev-unknown-vulns/README.md. Some of the new bugs have been found in the CVE target set of FUZZWARE. As such, the corresponding bug reproducing inputs can be found in 01-bug-finding-ability/results/bug-reproducers. The other bug reproducing inputs can be found in 04-prev-unknown-vulns/results/bug-reproducers.

As part of the reproduction, we also run HOEDUR on the remaining targets for a limited amount of time. As some new bugs require more computation power for HOEDUR to find and are found with some variance, it is expected that this will find some of the newly reported bugs, but may not reproduce all of them. The results can be found in 04-prev-unknown-vulns/results/bug-reproducers. The reproducers can be found in the directory tree of `bug-reproducers`. They represent triggers for the bugs listed in Table 3 in the paper. It is expected that HOEDUR triggers at least some of the bugs within the fuzzer reruns and as a result, a set of reproducers can be found and run via the scripts located in 04-prev-unknown-vulns/repro-run-scripts.

A.5 Notes on Reusability

We designed our experiments to be extendable and configurable in the computation resources required. Our configurations allow adding more experiments, such as `05-my-other-experiment`. Our scripts are built to be able to compute metadata for new fuzzing runs, such that one should only need to add another experiment and metrics as configurations. Coverage metadata, alongside their plots, can be configured to be computed using the provided scripts and setup. Some additional metrics are already computed, even though they are not referenced in this document. After running the experiments, one can find these metrics under the `results` experiment sub-directories.

Regarding the fuzzer itself, our prototype is published open source under https://github.com/fuzzware-fuzzer/hoedur. The source code separates the emulator from the fuzzer rather strictly, such that the community may make use of both components.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.