USENIX'23 Artifact Appendix: <Mitigating Security Risks in Linux with KLAUS – A Method for Evaluating Patch Correctness –>

Yuhang Wu
yuhang.wu@northwestern.edu
Northwestern University

Zhenpeng Lin
zplin@u.northwestern.edu
Northwestern University

Yueqi Chen
yueqi.chen@colorado.edu
University of Colorado Boulder

Dang K Le
dang.le@northwestern.edu
Northwestern University

Dongliang Mu
dzm91@hust.edu.cn
Huazhong University of Science and Technology

Xinyu Xing
xinyu.xing@northwestern.edu
Northwestern University

A Artifact Appendix

A.1 Abstract

This artifact is applying for an **Artifacts Available** badge, an **Artifacts Functional** badge, and an **Results Reproduced** badge.

The artifact primarily consists of two parts: the source code of KLAUS, and the Docker runtime environment for KLAUS. These components encompass the specific implementation of the designs in our paper, and also offer a very user-friendly and convenient mode of operation. KLAUS is a framework for verifying the correctness of Linux kernel patches, mainly composed of a static analysis part (identifying AWRPs as proposed in our paper) and a dynamic fuzz testing part (Fuzzing).

Firstly, our open-source source code includes the source code for static analysis, the source code for automatic instrumentation, and the source code for the fuzzer. These source codes have good extensibility and will be beneficial for other researchers to conduct more in-depth research improvements or extensions. Subsequently, we encapsulate the entire KLAUS framework in a Docker image, for the convenience of all researchers and users. In this Appendix, we will provide the necessary explanations and some screenshots to facilitate the evaluation of our academic achievements.

A.2 Description & Requirements

Hardware, for evaluation purposes, it is recommended to use a multi-core CPU environment that supports Kernel-based Virtual Machine technology. Additionally, due to the fuzzing process, it is recommended to have a minimum of 4 CPU cores, at least 32GB of memory, and a minimum of 100GB of hard disk space.

Software, the experiment requires a system with X86/64 architecture that supports running a Docker environment. It is necessary to have a network environment that supports the installation of dependencies and accessing information and code from the syzkaller community and Google's hosted Git website.

A.2.1 Security, privacy, and ethical concerns

All experiments (static analysis/fuzzing) are conducted within Docker containers, but it is necessary to map a shared folder from the local machine to the Docker container to serve as data storage. This might result in the generation of malicious files in the shared folder; however, as long as they are not executed on the local machine, they will not cause any harm. Each instance of the Fuzzer runs inside QEMU within the Docker container and will not pose any threat to the local machine's system.

A.2.2 How to access

All the artifacts are available in https://github.com/wupco/KLAUS, the directory

- *1-Docker-env* are the runtime evaluations.
- *2-Syzpatch* are the major portion of the code used in our research.

A.2.3 Hardware dependencies

- CPU: 4 CPU cores with virtualization technology.

- Memory: 32GB or larger.
- Disk space: 100GB or larger.

A.2.4 Software dependencies

- Softwares: `Docker`.

A.2.5 Benchmarks

None.

A.3 Set-up

Please adhere to the instructions provided in the README.md file of our GitHub repository.

A.3.1 Installation

None.

A.3.2 Basic Test

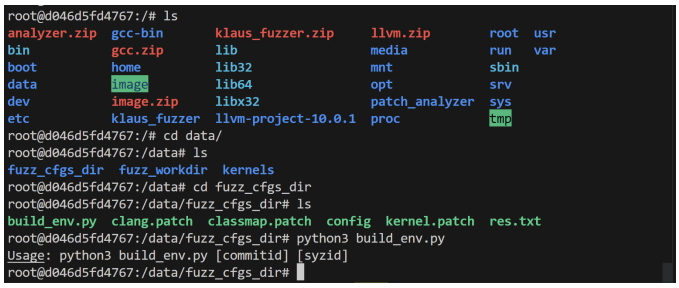

Figure 1: The files in the docker container.

After successfully building the Docker, execute it using the command `docker run -v $(pwd)/data:/data --rm -it --privileged klaus`. Upon executing this command, one should be inside the Docker container where commands can be executed freely. At this point, in the root directory of the container, there should be the directories essential for the experiment, namely `data`, `klaus_fuzzer`, `llvm-project-10.0.1`, `patch_analyzer`, `image`, and `gcc-bin`. Within the subdirectory `fuzz_cfgs_dir` of the `data` directory, one can execute the `build_env.py` file, which requires two arguments: `commitid`, representing the commit id of the buggy patch, and `syzid`, representing the bug report id of the bug that the patch addresses. For instance, to test the correctness of the patch located at https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=730c5fd42c1e, the required commitid is `730c5fd42c1e`, and the bug report https://syzkaller.appspot.com/bug?id=53b6555b27af2cae74e2fbdac6cadc73f9cb18aa id `syzid` is `53b6555b27af2cae74e2fbdac6cadc73f9cb18aa` that this patch fixes. This information can be observed in the Figure 1.

A.4 Evaluation workflow

KLAUS is utilized for verifying the correctness of Linux kernel patches. To achieve this objective, we employ a combination of static analysis and dynamic fuzzing techniques, which are the two critical components of KLAUS. It is important to note that our use of fuzzing technology is solely to validate that the AWRPs identified through static analysis are effective in assessing the correctness of Linux kernel patches; it is merely one implementation approach. Our primary contribution lies in the discovery and identification of AWRPs through static analysis. By successfully executing KLAUS, we anticipate generating information about the identified AWRPs, and also utilizing this information to automatically instrument the code pre-fuzzing. Ultimately, this will enable the successful launch of the fuzzer to evaluate the patch.

A.4.1 Major Claims

(C1): KLAUS will identify the AWRPs corresponding to each case in the ground truth dataset with respect to the patch.

(C2): The Fuzzer component of KLAUS can operate normally, and there is a high probability that it can trigger bugs resulting from errors in the patch.

A.4.2 Experiments

First, please follow the guide in our GitHub repository to properly set up the Docker environment. Subsequently, launch Docker, enter the Docker container, and execute the `build_env.py` file with the specified parameters. It is imperative to note that detailed information regarding our ground truth data is located in `Evaluation_Results.xlsx`. Upon the completion of static analysis and instrumentation, navigate to `/data/fuzz_cfgs_dir/[commitid]` and execute `fuzz_start.sh` to initiate the fuzzer. Information on AWRPs can be found in `prop.txt` and `cond.txt` within the `/data/kernels/[commitid]` directory, while the working directory of the fuzzer is located at `/data/fuzz_workdir`. Additionally, configuration information for running the fuzzer can be found in `/data/fuzz_cfgs_dir/[commitid]/config`. If it is necessary to empty and reset the environment under the `data` folder, `cleardata.sh` can be executed on the local machine. It is important to note that occasionally, when there is an error in applying `clang.patch` or `classmap.patch`, it can be ignored by pressing Enter directly.

(E1): Test whether the ground truth cases can be analyzed correctly.
Execution: execute `build_env.py` file with the specified parameters.
Results: `[commitid] has inst` will be reported in stdout. Information on AWRPs can be found in `prop.txt` and `cond.txt` within the `/data/kernels/[commitid]` directory.

(E2): Test whether the fuzzer can be run normally.

Execution: execute `fuzz_start.sh` file in `/data/fuzz_cfgs_dir/[commitid]`.

Results: The fuzzer will operate normally, and concurrently, the status of the fuzzer will be outputted to the stdout.

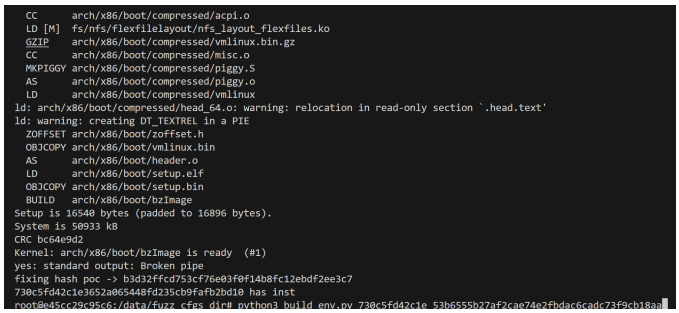

Figure 2: The expected result of the static analysis part.

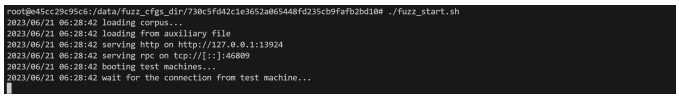

Figure 3: The fuzzer has been successfully executed.

For the results of the static analysis part, as shown in Figure 2, it successfully identified the AWRPs in the patch and instrumented the kernel code. Subsequently, when the fuzzer is executed, information and status during fuzzing will be displayed, as in Figure 3.