Fuzzing BusyBox: Leveraging LLM and Crash Reuse for Embedded Bug Unearthing

Asmita, *University of California, Davis;* Yaroslav Oliinyk and Michael Scott, *NetRise;* Ryan Tsang, Chongzhou Fang, and Houman Homayoun, *University of California, Davis*

https://www.usenix.org/conference/usenixsecurity24/presentation/asmita

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security ’24 Artifact Appendix: Fuzzing BusyBox: Leveraging LLM and Crash Reuse for Embedded Bug Unearthing

Asmita[1], Yaroslav Oliinyk[2], Michael Scott[2], Ryan Tsang[1],
Chongzhou Fang[1], Houman Homayoun[1]

[1]University of California, Davis [2]NetRise

A Artifact Appendix

In our paper, we developed a basic python-based automation framework to perform fuzzing on a large-batch of BusyBox ELFs. As mentioned in the paper in **Section 4.2.1**, we made that available on Github.

A.1 Abstract

We provide the automation script to perform fuzzing on a large batch of BusyBox target binaries using AFL++. It is provided in *automation_src folder*. Note : Currently it is for busybox awk applet fuzzing, change *afl_fuzz_command* in *afl_fuzz.py* in case of different applet. Supported target architecture : x86_64 and ARM_32. *fuzz_multiple_targets.py* is the main script that takes in a bunch of collected BusyBox target binaries, perform fuzzing on each target using AFL++ till the runtime provided by the user. And after fuzzing is done, it stores the fuzzing stats (json) of all the target in the output directory.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

As per our knowledge, there is no security risk involved in using this framework.

A.2.2 How to access

Artifact is available on Github

A.2.3 Hardware dependencies

None

A.2.4 Software dependencies

Linux OS, dependent on AFL++ Qemu mode - Link, For ARM32 based BusyBox ELFs, there are some arm dependencies which is provided in *arm_dependencies* folder. We have hotsed the docker image for ARM32 based ELFs : *asmitaj08/afl-qemu-arm*

A.2.5 Benchmarks

None

A.3 Set-up

A.3.1 Installation

For x86 based BusyBox ELFs, it follows the steps of AFL++ installation for Qemu mode. Whereas in case of ARM32 based BusyBox , one can directly pull the provided docker *asmitaj08/afl-qemu-arm*

Then use the command :

python3 fuzz_multiple_targets.py –input /path/to/binary/collection –arch ARM_32/x86_64 –corpus /path/to/corpus –output /path/for/output –afl-path path/of/afl –run-time required_runtime –depend arm_dependecies_in_case_of_arm fuzz_multiple_targets.py

A.3.2 Basic Test

After performing the above installation, and command execution , it takes in a bunch of collected BusyBox target binaries, perform fuzzing on each target using AFL++ till the runtime provided by the user. And after fuzzing is done, it stores the fuzzing stats (json) of all the target in the output directory. We have provided a sample example under the "demo_folder".

A.4 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/usenixsec2024/`.