SyzScope: Revealing High-Risk Security Impacts of Fuzzer-Exposed Bugs in Linux kernel

Xiaochen Zou, Guoren Li, Weiteng Chen, Hang Zhang, and Zhiyun Qian, *UC Riverside*

https://www.usenix.org/conference/usenixsecurity22/presentation/zou

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

All experiments were conducted in Ubuntu-18.04 with 1TB memory and Intel(R) Xeon(R) Gold 6248 20 Core CPU @ 2.50GHz * 2. But it's ok if we don't have that much computing resource. The minimal configuration is at least 4 core CPU, 8G memory and at least 200G disk space. It's recommend to enable more CPU cores, they will speedup the compiling, fuzzing and symbolic execution significantly.

In our paper, we test more than 1000 bugs and each of them require 3 hours kernel fuzzing, 1 hour static analysis and 4 hours symbolic execution. If we plan to accomplish all 1000 cases, it costs more than two weeks, therefore we only choose a subset of them for Artifact Functional.

A.2 Artifact check-list (meta-information)

- **Program:** SyzScope now open source at `https://github.com/seclab-ucr/SyzScope/tree/b1a6e20783ba8c92dd33d508e469bc24eaacaab6`. This version is the one we conduct the experiment. It's recommend to download the docker container we provided. The details shows in the github README. If you have a fast internet speed, you may want to pull the `ready2go` docker image, otherwise `mini` docker image requires extra compliation.
- **Compilation:** If you pull the `mini` docker image or run SyzScope in your custom system, you have to compile the essential tools. Using command *python3 syzscope –install-requirements*. The detailed instructions can be found at our github page `https://github.com/seclab-ucr/SyzScope/`
- **Data set:** Since running all cases takes more than two weeks, we only prepare a subset of them for the artifact functional badge. The dateset we provide is the ones we got CVEs from: https://sites.google.com/view/syzscope/home. We made a google sheet of this dataset at `https://docs.google.com/spreadsheets/d/16tt4Mo40iyWeuxOXBpRtmV_Zjdddta9nIy-poEuq66E/edit?usp=sharing`
- **Run-time environment:** SyzScope is designed on Ubuntu 18.04, written by python3, C++, golang, and bash script. Every other Linux system should support SyzScope just fine. But we still recommend to use our docker image in case any environment differences.
- **Output:** We wrote detailed tutorial for how to read output from fuzzing, static analysis and symbolic execution. You can access them on

  `https://github.com/seclab-ucr/SyzScope/blob/master/tutorial/fuzzing.md`

  `https://github.com/seclab-ucr/SyzScope/blob/master/tutorial/static_taint_analysis.md`

  `https://github.com/seclab-ucr/SyzScope/blob/master/tutorial/sym_exec.md`
- **Experiments:** To run the experiment, you first need to prepare the case hash for SyzScope. Since we already give the dataset, you just need to simply copy and paste the case hash into a file (one hash per line), let's say the name of that file is *dataset*, and run SyzScope with `nohup python3 syzscope -i dataset -RP -KF -SA -SE -timeout-kernel-fuzzing 3 -timeout-static-analysis 3600 -timeout-symbolic-execution 14400 -guided -be-bully &`, If you have enough CPU cores, you can even try run multiple cases at the same time by specify `-pm`, for example `-pm 8` means run 8 cases at the same time. The log output will be written into nohup.out since we use `nohup` to make the process running in background.
- **How much disk space required (approximately)?:** SyzScope requires 24GB for essential packages and tools. Besides them, SyzScope may require 2GB for each case. Considering 8 cases in our dataset, it's better to have 20GB remaining. Therefore in total we suggest having 50GB reamining on your disk.
- **How much time is needed to complete experiments (approximately)?:** At maximum each case takes 3 hours kernel fuzzing, 1 hours static analysis and 4 hours symbolic execution (8 hours in total), but some cases may terminate early. If we run these 8 cases together `-pm 8`, we should finish all of them in 8 hours, if we run them one by one, it probably takes more than 3 days (64 hours).
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT License
- **Data licenses (if publicly available)?:** MIT License
- **Archived (provide DOI)?:** `https://github.com/seclab-ucr/SyzScope/tree/b1a6e20783ba8c92dd33d508e469bc24eaacaab6` is the stable version that conduct all experiments.

A.3 Description

A.3.1 How to access

Access the github repo at `https://github.com/seclab-ucr/SyzScope/tree/b1a6e20783ba8c92dd33d508e469bc24eaacaab6`. The current version is the one conduct all experiment. Another option is to download the docker image, you can find the instructions on github repo.

A.3.2 Hardware dependencies

The minimal configuration is at least 4 core CPU, 8G memory and at least 200G disk space. It's recommend to have more CPU cores, they will speedup the compiling, fuzzing and symbolic execution significantly. To run multiple cases at the same time, use `-pm` argument. We recommend you run `n` cases at the same time which n equals to the number of cores divide 4 (e.g, if you have 16 cores, we recommend you use `-pm 4` to run 4 cases that the same time)

A.3.3 Software dependencies

Software dependencies will be installed by running `python3 syzscope -install-requirements`. Or you can use the ready2go docker image within all dependencies installed

A.3.4 Data sets

https://docs.google.com/spreadsheets/d/16tt4Mo40iyWeuxOXBpRtmV_Zjdddta9nIy-poEuq66E/edit?usp=sharing

A.4 Installation

The detailed installation instructions are presented in the github repo https://github.com/seclab-ucr/SyzScope.

A.5 Experiment workflow

First, gather all cases we want SyzScope to run, copy the hash value from dataset page we provided, and paste them into a file, one hash per line. Second, run SyzScope with `nohup python3 syzscope -i dataset -RP -KF -SA -SE -timeout-kernel-fuzzing 3 -timeout-static-analysis 3600 -timeout-symbolic-execution 14400 -guided -be-bully &`, this process may take a long time. If you have more than 4 cores, you can run multiple cases at the same time by provide `-pm` arguments. For example, `nohup python3 syzscope -i dataset -RP -KF -SA -SE -timeout-kernel-fuzzing 3 -timeout-static-analysis 3600 -timeout-symbolic-execution 14400 -guided -be-bully -pm 6 &` runs 6 cases at the same time, but it requires at least 4*6 cores on your machines.

Third, cases that found high-risk impacts will be moved to `succeed` folder, cases that failed to find high-risk impacts will be moved to `completed` folders. Rerun those failed cases by using `-force`. For example, `python3 syzscope -i f99edaeec58ad40380ed5813d89e205861be2896 -RP -KF -SA -SE -timeout-kernel-fuzzing 3 -timeout-static-analysis 3600 -timeout-symbolic-execution 14400 -guided -be-bully -force`. If any error occurs, check out the common issues on our github page https://github.com/seclab-ucr/SyzScope/blob/master/tutorial/common_issues.md.

Final, check out the results from symbolic execution and compare it with the ones shown on our webpage https://sites.google.com/view/syzscope/home. The results of symbolic execution is in `work/succeed/xxx/sym-xxx/symbolic_execution.log`. The high-risk impacts stores under `work/succeed/xxx/sym-xxx/primitives`. For example, to verify case `ce5f07d6ec3b5050b8f0728a3b389aa510f2591b`, you will find a function pointer dereference impact at `work/succeed/ce5f07d/sym-ori/primitives/FPD-try_to_wake_up-0xffffffff8137be7d-17` which related to the one we present on our webpage https://sites.google.com/view/syzscope/kasan-use-after-free-read-in-io_async_task_func.

A.6 Evaluation and expected results

Due to race condition, some bugs may be hard to trigger or trigger different contexts. We just need to run multiple times to increase the possibility of bug reproducing.

The final component is symbolic execution. To verify the final results from symbolic execution, check out the file "symbolic_execution.log". (Read more details at https://github.com/seclab-ucr/SyzScope/blob/master/tutorial/sym_exec.md) The number of new impacts shown at the end of the file.

In terms of the results, you may have slightly different output due to different configuration of experiment machine. We ran all experiments on Ubuntu-18.04 with 1TB memory and Intel(R) Xeon(R) Gold 6248 20 Core CPU @2.50GHz * 2. If you use machine that less powerful than ours, you might have less high-risk impacts comparing to our results. One approach to verify our results is through the CVE we obtained. We create a page to document the detailed analysis about the cases that received CVE, and each of them has at least one high-risk impact. These high-risk impacts are the reasons for CVE obtainment, so if you can verify those high-risk impacts on your end, it means the results are reproduceable.

See the link to each detailed analysis on our dataset page.

A.7 Notes

`457491c4672d7b52c1007db213d93e47c711fae6` has multiple UAF contexts due to race condition. Our web page shows only one of them(`ucma_close`), but another UAF context(`ucma_destroy_id`) may also lead to control flow hijacking.

`f99edaeec58ad40380ed5813d89e205861be2896` may be hard to trigger. If it failed to run symbolic execution, try `python3 syzscope -i f99edaeec58ad40380ed5813d89e205861be2896 -RP -SE -timeout-symbolic-execution 14400 -guided -force`.

`4bf11aa05c4ca51ce0df86e500fce486552dc8d2` has an arbitrary value write on a local variable in `hci_extended_inquiry_result_evt` shown on our webpage. However we abandoned any local variable write due to short life span of local variable and they are merely exploitable. So Running SyzScope on this case now will no longer find any high-risk impact.