USENIX Security '26 Artifact Appendix: Meerkat: Pushing the Practical Limits of Dynamic Bisection with PoC Mutation

Joseph Bursey
University of California, Irvine

Christoph Sendner
University of California, Irvine

Ardalan Amiri Sani
University of California, Irvine

Zhiyun Qian
University of California, Riverside

A Artifact Appendix

A.1 Abstract

This appendix is provided to confer clear instructions in order to run and evaluate Meerkat, our dynamic bisection tool which utilizes PoC mutation during bisection. Using our instructions and scripts, our readers should be able to verify the claims made in the paper that Meerkat is able to reach the BiC in 34 more cases than Syz-bisect, and that it is able to do so in a comparable runtime.

A.2 Description & Requirements

The presented repository contains the entire source code for Meerkat, the dataset and results from our paper (`results/`), a fork of Syzkaller (`syzkaller/`), and scripts to build a Debian Stretch image (`image/stretch/`), as well as easy-to-use scripts to build and run everything from a single command. The setup script additionally pulls a set of compilers to be used during bisection from a separate Zenodo repository.

Meerkat has been tested on Ubuntu 20.04 and Ubuntu 24.04. It will likely run on other debian-based systems, but has not been tested. Meerkat should be run on and x86_64 machine, and can be run in a VM assuming nested virtualization is enabled. Meerkat uses 16 threads (`nproc` $>=$ 16) to build the kernel and boot VMs, but 20 is recommended. Lastly, Meerkat allocates up to 32GB of memory across 8 VMs, so the host system should have at least that much to prevent bottle necks.

A.2.1 Security, privacy, and ethical concerns

Meerkat does not pose any privacy or security-related risks to its users. Meerkat does not export any information from the host system, nor does it remove any safeguards from the system. All code and libraries are pulled from trusted sources. All other ethical concerns are discussed in the Ethical Considerations appendix.

An informational note: the `artifact-verify.sh` script adds the current user to the `kvm` group for the purpose of booting qemu VMs. Additionally, the `artifact-verify.sh` script creates a symbolic link for the missing library `libmpfr.so.4`, simply pointing it at the newer `libmpfr.so.6`. This is for the purpose of making the older compilers function on newer systems.

A.2.2 How to access

Meerkat has been made accessible on both Zenodo: https://doi.org/10.5281/zenodo.20317981, as well as GitHub: https://github.com/trusslab/meerkat. The compiler repository can be accessed on Zenodo as well: https://doi.org/10.5281/zenodo.20316000.

A.2.3 Hardware dependencies

Meerkat builds the Linux kernel and boots several VMs simultaneously. For this it expects a CPU capable of at least 16 threads (`nproc` $>=$ 16), as well as at least 32GB of memory. The Kernel repository and images can be rather large as well. We recommend around 32GB of free disk space.

A.2.4 Software dependencies

Meerkat is provided with easy-to-use scripts which will download and install all known dependencies. These include:

- Several packages pulled from `apt`: make, git, gcc, g++, ccache, build-essential, flex, bison, libncurses-dev, libelf-dev, libssl-dev, dwarves, libdw-dev, qemu-system-x86, libmpfr-dev, debootstrap, dpkg.
- A set of compilers pulled from our Zenodo repository.
- A recent version of Go.

A.2.5 Benchmarks

The 200 bug dataset is provided in the repository (`parse/bugs.csv`).

A.3 Set-up

Our artifact evaluation script (`artifact-verify.sh`) automatically sets up the environment, handling all dependencies to the best of our knowledge. Should the user desire to setup up the environment before running Meerkat, they may use the `setup.sh` script.

A.3.1 Installation

The `setup.sh` script discussed above handles all installation of Meerkat's dependencies. Meerkat itself does not need to be installed, merely built.

A.3.2 Basic Test

A basic functionality test can be conducted using:

```
./artifact-verify.sh functional
```

This will bisect a single bug that should complete in around 14 hours. Alternatively, if the evaluator does not want to bisect an entire bug at this stage, they may use the command:

```
./mk-manager.sh -s 2 -e 2 -i 1 \
-b parse/bugs.csv \
-F ff-test,poc-all-pocs,find-only
```

This will build and test at the anchor commit, and should complete in around 45 minutes. Documentation on `mk-manager.sh` can be found at `docs/mk-manager.md`.

As the output from Meerkat is quite long, please refer to our documentation provided in the repository at `docs/reading-results.md`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Meerkat is able to reach the BiC in 34 more cases than Syz-Bisect.

(C2): Meerkat completes bisection in a total runtime that is closely comparable to Syz-bisect.

A.4.2 Experiments

The experiments carried out in our paper took several months to complete computationally, as well as a considerable amount of human-hours to examine. The entire experiment can be recreated using the provided scripts, however we do not expect our users to do so. Instead, we provide a subset consisting of the 37 bugs where Meerkat was able to reach the BiC, but Syz-bisect could not (The difference between 34 and 37 is explained in `docs/README.md`). This should be enough to verify the result in **(C1)**, while not taking several months to complete.

(E1): [17+ compute-days + 1-2 human-hours]: Run Meerkat on the smaller 37 bug dataset, and observe several results from the bisection logs.
How to: Execute our `./artifact-verify.sh` script with the described preset, allow it to run for about 2 weeks, then read the results from the bisection logs.
Preparation: Optionally run `./setup.sh`. This step is automatically handled by `artifact-verify.sh`.
Execution: Execute the command:
`./artifact-verify.sh reproducible`
Please do this in `screen`/`tmux`. Based on our results, this process should take a little over 2 weeks, but it may take longer depending on the user's environment.
Results (C1): Observe the bisection logs in the files `wd-meerkat-1/log/bug####.log`. Compare the final results marked "Bisection Result:" to the ground truth hashes provided in `results/bics.csv`. In ideally all cases Meerkat has reached the correct BiC, but some variance is expected. You can further compare to the Syz-bisect results provided in `results/sb/`. The bisected hash is provided on the final line of each file.
Results (C2): Observe the same log files from above, this time noting the "Run Time:" result. It is recommended to export the times to a spreadsheet, formatting as a duration. Observe that the average and median runtimes are similar to what is presented in the paper. You can further compare against the provided Syz-bisect results. Grep for "total time:" to find the runtime in the log.

(E2): [several compute-months + several human-days]: Run the entire ablation study. We do not expect you to run this. This is only here to provide instruction if a reader requires it.
How to: Execute our `./artifact-verify.sh` with multiple presets in parallel, allow to run, then read the results from the log files.
Preparation: Run `./setup.sh`. This will stop parallel instances of `./artifact-verify.sh` from colliding at this step.
Execution: Execute the commands:
`./artifact-verify.sh mk full`
`./artifact-verify.sh mkm1p full`
`./artifact-verify.sh mknm full`
`./artifact-verify.sh mk1p full`
Run each instance in parallel when possible and spread across multiple computers/servers when possible.
Results: Observe the results in the same way as **(E1)**, accounting for different working directories.

A.5 Notes on Reusability

Extensive documentation is provided in our code repository, both on Zenodo and GitHub. Please refer to `docs/`.

A.6 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/usenixsec2026/.