A Artifact Appendix

A.1 Abstract

Using RETBleed, an unprivileged user can leak arbitrary memory from the system. The vulnerable systems are listed in Table 1 in the paper. RETBleed has an offline phase, where exploitation primitives are discovered by our framework. We ran the framework, designed exploits and proof of concept code on Ubuntu 5.8.0-63-generic. To match our results closely, an Intel i7-8700K (Coffee Lake) and an AMD EPYC 7252 (Zen2) are recommended. 16 GiB of RAM is recommended. To run our framework on the entire test suite (optional), we recommend 40 GiB of free disk space. However, we have included example output from the framework with particularly huge files omitted, which is 1-2 GiB and can be inspected instead.

We provide a snapshot of the current RETBleed repository, which hosts the majority of code used throughout the project. The repository is public, and detailed instructions are found in the `README.md` files inside the git repository.

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

- **Binary:** A Linux image is included to test the gadget scanner. Source code is included to build the other binaries used by the framework, PoCs, kernel modules and end-to-end exploits.
- **Run-time environment:** Ubuntu 20.04.3 LTS (Focal Fossa) with `linux-image-5.8.0-63-generic`. Most experiments are designed to run on bare-metal, not on a VM.
- **Hardware:** Intel Core generations 6–8; AMD Zen, Zen+ and Zen 2
- **Security, privacy, and ethical concerns:** Responsible disclosure ended on 12 July 2022.
- **Experiments:**
  - `./retbleed_zen/pocs/ret_bti` finds the patterns that cause BTB collisions.
  - `./retbleed_zen/pocs/cp_bti` shows that collisions happen across.
  - `./retbleed_intel/pocs/ret_bti` shows that returns go via BTB.
  - `./retbleed_intel/pocs/cp_bti` shows that we can train across kernel returns in user space.
  - `./rsb_depth_check` RSB use on AMD and Intel. For Intel, it also indicates that some other "near branch" prediction mechanism takes place.
  - `./zen_ras_vs_btb/` is illustrated in Figure 5. It shows that Return Address Stack (RAS, aka RSB) is not used on Zen 2 when there's a BTB entry. To evaluate Zen(+), `BTI_PATTERN` must be manually set.
  - `./ret_finder/` constitutes the part of framework to detect vulnerable return instructions in the kernel.
  - `./gadget_scanner/` was used to discover disclosure gadgets.
  - `./bhb_generate/` was used to trace taken branches preceding a vulnerable return in a kernel running inside a VM.
- **How much disk space required (approximately)?:** 300 MiB. 40 GiB to reproduce our framework output.
- **How much time is needed to prepare workflow (approximately)?:** Less than 1 hour.
- **How much time is needed to complete experiments (approximately)?:** Up to 12 hours.
- **Publicly available (explicitly provide evolving version reference)?:** https://github.com/comsec-group/retbleed/.
- **Workflow frameworks used?:** git, linux-test-project, BCC/eBPF, ftrace
- **Archived:** https://github.com/comsec-group/retbleed/releases/tag/sec22-artifact-final

A.3 Description

A.3.1 How to access

Clone using git, `git clone https://github.com/comsec-group/retbleed.git`. Also clone submodules, `git submodule update -init`

A.3.2 Hardware dependencies

Intel i7-8700K (Coffee Lake) and AMD EPYC 7252 (Zen 2) or similar. We were evaluating all experiments and exploits on bare-metal hardware. Running in a VM may pose unexpected challenges.

A.3.3 Software dependencies

Ubuntu focal, linux-image-5.8.0-63-generic, clang, python3, bcc, bpfcc-tools, pytest, pyelftools

A.4 Installation

Instructions available in *README.md* files. See repository for details. Software dependencies can be installed using `apt-get` and `pip3`. Linux test project included as a submodule that can be cloned using `git submodule update -init` from the repository

A.5 Experiment workflow

Instructions available in *README.md* files. See repository.

A.6 Evaluation and expected results

- **Reverse engineering of return instruction behavior.** Several experiments are included that reverse engineer return behavior.
- **Framework that finds vulnerable return instructions.** We include the framework that finds these. It should result in the numbers from Figure 11.

- **Poisoning kernel returns from an unprivileged process.** Our PoCs and exploits all do this.
- **Leaking arbitrary memory at 3.9 kB/s and 219 bytes/s on AMD Zen2 and Intel Coffee Lake respectively.** We provide instructions in the repository for how to run these PoCs. We also include exploits to leak */etc/shadow*. Furthermore, we also explain how we measure the leakage rate. The median the leakage rate should closely match with the expected results.

A.7 Experiment customization

We clarify in the READMEs provided the cases where certain preprocessor macros can/should be altered for additional results. For example, to run `rsb_depth_check` on AMD, uncomment L11 in `ret_chain.c`.

A.8 Notes

The documentation here is sparse, since everything written here has already been provided in the artifact project itself. Please use your own hardware. Should you not have access to hardware that is similar to ours, please contact us.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.