USENIX Security '25 Artifact Appendix: System Register Hijacking: Compromising Kernel Integrity By Turning System Registers Against the System

Jennifer Miller[*], Manas Ghandat[*], Kyle Zeng[*], Hongkai Chen[*], Abdelouahab (Habs) Benchikh[*]
Tiffany Bao[*], Ruoyu Wang[*], Adam Doupé[*], Yan Shoshitaishvili[*]
[*]Arizona State University
{jmill,mghandat,zengyhkyle,hongkai.chen,abenchik,tbao,fishw,doupe,yans}@asu.edu

A Artifact Appendix

A.1 Abstract

Our paper proposes System Register Hijacking (SRH) as a class of exploitation techniques. As part of the paper, we performed several experiments to determine the prevalence of such gadgets and their applicability. To this end, our artifacts include the following components:

- Scripts for measuring the occurrence of system instructions that may serve as SRH gadgets.
- Kernel module-based PoCs used to demonstrate individual techniques.
- Modified exploits included in our evaluation.
- The setup for reproducing the case study on bypassing FineIBT.
- An implementation of an AVX timing side channel, described in previous work but currently lacking an open-source implementation: https://arxiv.org/abs/2304.07940, usable for breaking KASLR on modern systems.

A.2 Description & Requirements

Our evaluation was primarily conducted on three systems: 1. AMD Ryzen 7 PRO 6850U with 32GB RAM, 2. AMD Ryzen Threadripper 7960X with 32GB RAM, 3. 11th Gen Intel Core i7-1185G7 with 32GB RAM, all of which running Ubuntu 22.04. However, a single Intel x86-64 system with 32GB+ of RAM should be sufficient for evaluation.

A.2.1 Security, privacy, and ethical concerns

Reviewers should take precautions when working with the kernel exploits included as part of our evaluation. Running these exploits outside of a virtualized environment on unpatched Linux kernel versions may result in system instability.

A.2.2 How to access

Our artifacts can be accessed at the following location: https://doi.org/10.5281/zenodo.14728440. The link is associated with a compressed archive of the artifacts. Decompressing and extracting that file will yield an artifacts directory containing a README file that further describes the organizational structure of the artifacts.

A.2.3 Hardware dependencies

Our artifacts expect evaluators to have a modern x86-64 system. The PoCs expect to be ran using hardware-accelerated virtualization features supported by the major x86-64 CPU vendors. Our FineIBT evaluation requires access to a bare-metal x86-64 system with an Intel CPU supporting the IBT feature.

A.2.4 Software dependencies

Our artifacts require that evaluators have access to a Linux system with Docker (https://www.docker.com/) installed. We expect that evaluators to be able to install commonly used software packages on their system, e.g., qemu-system, gcc, and python3.

A.2.5 Benchmarks

None.

A.3 Set-up

Each component should provide either a script that handles software dependencies for the evaluator, or rely only on software packages that are widely used.

A.3.1 Installation

The only major dependency that will not be installed by setup scripts for the experiments themselves is Docker. Evaluators should follow the instructions provided on

Docker's website https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository to set up Docker if they do not already have it installed.

A.3.2 Basic Test

As a basic test, evaluators should attempt to pull and run an Ubuntu 22.04 Docker image. This can be done by running the "docker pull ubuntu:22.04" and "docker run -it ubuntu:22.04 bash" commands. The result of these commands should be that the evaluator has a Bash shell in an Ubuntu 22.04 container.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Ability of "swapgs Stack Pivoting" to Bypass FineIBT. This is proven by Experiment (E4), which relates to Section 8.3 of the paper.

A.4.2 Experiments

(E1): [Gadget Analysis] [10 human-minutes + 8 compute-hours + 3GB disk]: Downloads kernel packages from Ubuntu and Fedora repositories, analyzes the kernel images for System Register Hijacking gadgets, and produces a csv file of resulting gadgets
How to: In the "analyzer" directory of the artifacts, execute the "analyze.sh" script. It will run the entire analysis.
Results: The README describes how to use the scripts that were used to generate the tables found in the paper, the tables in the paper should closely match the output from this analysis.

(E2): [Kernel Module PoCs] [20 human-minutes + 1 compute-hour]:
How to: In the "eval-module" directory there is a subdirectory for each architecture: "x86" and "arm". Each architecture has its own setup. The x86 directory contains a "setup.sh" script that will install several apt and python packages, fetch an Ubuntu kernel image, and build a debian filesystem image. From there the "run-50.sh" script can be used to semi-automate the evaluation of individual technique reliability. For ARM, the "setup" directory in the directory of each technique contains a "startvm" script that will boot a vm containing the PoC, which can be run by executing "./poc" in the VM's shell.
Results: Results from running the experiment can be compared with the results included in the paper in Table 2.

(E3): [CVE-Based PoCs] [10 human-minutes + 4 compute-hours]:
How to: In the "eval-cve" directory there is a script called "run-eval.sh" that will run the exploit for each kCTF based CVE we modified 50 times and collect the success rates. The script can be run with either "modified" or "original" as an argument to evalutate the stability of either our modified exploits or the exploits using the control flow hijacking approach found in the original exploits. The older CVE, which was pulled from the RetSpill (https://github.com/sefcom/RetSpill) dataset, can be found in the "eval-cve-retspill" directory. This directory contains a script, "start.sh" that will spawn a containerized environment to evaluate the exploit in.
Results: The results of this experiment can be compared with the results found in Table 3 of the paper.

(E4): [FineIBT Evaluation] [2 human-hours + 1 compute-hour]:
How to: In the "eval-fineibt/vuln-experiment" directory, there is an "iso" directory that should be turned into an ISO file via the "grub-mkrescue" command found in the README. The ISO should be loaded onto a bootable drive and be booted into on a baremetal system. Baremetal is necessary because IBT virtualization is not currently available. Once booted into the ISO, the evaluator will be placed into an initramfs containing our modified exploit at "/exploit".
Results: The exploit should sucessfully and reliably print the flag and then attempt to shut down the system.

(E5): [Sidechannel] [10 human-minutes + 5 compute-minutes]: ...
How to: In the "sidechannel" directory, run "start.sh". This will build and drop the evaluator into a containerized environment. In the container the "run.sh" script can be used to boot a VM in which to test the side-channel. Running "./kaslr" in the VM should result in the KASLR base being printed, which can be compared with the KASLR base that gets logged to the console on boot.
Results: The side-channel should leak the base address of the kernel.

A.5 Notes on Reusability

The header-files used for several of the CVE-based exploits for the 'swapgs Stack Pivoting' technique can be reused to apply the technique to other exploits. The sidechannel implementation provided can be reused to demonstrate the ability to break KASLR in future research.

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/usenixsec2025/.