ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security ’25 Artifact Appendix: Relocate-Vote: Using Sparsity Information to Exploit Ciphertext Side-Channels

Yuqin Yan[†], Wei Huang[†‡], Ilya Grishchenko[†], Gururaj Saileshwar[†], Aastha Mehta[*], and David Lie[†]

[†]University of Toronto, [‡]Seneca Polytechnic, [*]University of British Columbia
yuqin.yan@mail.utoronto.ca, wei.huang1@senecapolytechnic.ca, gururaj@cs.toronto.edu,
aasthakm@cs.ubc.ca, {ilya.grishchenko,david.lie}@utoronto.ca

A Artifact Appendix

A.1 Abstract

This artifact includes the source code, scripts, and datasets used to support the main claims, figures, and tables presented in the paper. The paper demonstrates how a malicious hypervisor can infer sparsity patterns in the memory of victim confidential Virtual Machines (CVMs) protected by AMD SEV-SNP. This is accomplished by identifying encrypted memory blocks that contain either prevalent or non-prevalent values, exploiting ciphertext side channels via the `SNP_PAGE_MOVE` command—designed initially for resource management.

At a high level, the artifact provides: (1) a malicious hypervisor implemented as part of a modified Linux kernel, (2) code simulating the behavior of a victim CVM, (3) monitoring logic used by the hypervisor to observe the victim through controlled channels and ciphertext side channels, and (4) analysis tools to recover and evaluate sparsity information from the victim’s memory.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact is based on a modified kernel implementation, SEV-Step. Both the original version and our implementation prioritize the functionality of attack primitives over kernel stability. As a result, the system may experience unpredictable behavior, including kernel crashes, instability, and unresponsiveness. **WARNING:** This artifact is intended strictly for research and evaluation purposes. **Do not use it in production environments.**

Known limitations and usage guidelines:

- Do not run multiple VMs simultaneously.
- Do not assign multiple cores to a single VM, which may prevent the VM from booting.
- Avoid forcefully terminating an evaluation program, which may cause system issues.
- The CVMs cannot be correctly shut down sometimes. Specifically, you cannot terminate a running CVM by either Ctrl-C in the terminal with the “login” prompt or `pkill`. If this happens, you need to reboot the host machine.

A.2.2 How to access

The artifact is available at Zenodo via the following concept DOI: https://doi.org/10.5281/zenodo.15609905.

A.2.3 Hardware dependencies

The evaluation of the attack primitive leveraging the `SNP_PAGE_MOVE` command is conducted on AMD SEV-SNP platforms. Since AMD fixed an implementation bug in firmware version 1.55 (build 20), our evaluation requires firmware version 1.55.20 or later for efficient page relocation.

A.2.4 Software dependencies

The host platform is running Ubuntu 24.04.1 LTS.

Environments supporting `pip install`. Examples of such environments are Anaconda and `python-venv`. Recently, `pip install` in the native, raw environment was prohibited due to PEP 668—Marking Python base environments as “‘externally managed”.

Blender. Blender is used to manually assemble and visualize the pieces leaked from the OpenVDB operations performed by the victim. The installation instructions can be found at https://www.blender.org/download/.

A.2.5 Benchmarks

Dataset. We rely on the external dataset, CQ500. Although it is externally available at https://www.kaggle.com/datasets/crawford/qureai-headct, we have included the necessary components in our artifact package under `openvdb-leak/data/qureai-headct/` to ensure longterm accessibility. The distribution complies with the original license, CC BY-NC-SA 4.0. Both the license file (LICENSE) and a corresponding README.md are provided in the directory.

Weights of Large Language Model (LLM). Our experiments relied on the weights of ReLULlama-13b, available at https://huggingface.co/PowerInfer/ReluLLaMA-13B-PowerInfer-GGUF. For the ease of reproducibility, we include the weights in the artifact package. The weights are released under the llama2 license, and the license files LICENSE.txt and Notice are included in the package.

A.3 Set-up

A.3.1 Installation

The installation consists of four steps. Follow the detailed instructions in the "Installation" section of README.md.

1. Set RV_ROOT_DIR to the root directory of the artifact by executing the script in it:

```
source ./env.sh
```

2. Create the CVM images, aslr.qcow2 and big-disk.qcow2 (Table 1) in $RV_ROOT_DIR/vm-images-creation/. The first line takes around 10 minutes to execute. The second line can take around 30 minutes. The default username is ubuntu and the password is 123456.

```
$RV_ROOT_DIR/vm-images-creation/create_vm1.sh
$RV_ROOT_DIR/vm-images-creation/create_vm2.sh
```

3. Build and install the customized Linux kernel.

```
cd $RV_ROOT_DIR/sev-step
./build.sh ovmf && \
./build.sh qemu && \
./build.sh kernel
```

The above commands should generate kernel-packages/ in $RV_ROOT_DIR/sev-step/. Install the *.deb packages using the following command:

```
sudo dpkg -i \
    $RV_ROOT_DIR/sev-step/kernel-packages/*.deb
```

After installation, execute sudo update-grub to see if the following lines are in the output:

```
Found linux image: /boot/vmlinuz-5.19.0-rc6-sev-step-999ae99
Found initrd image: /boot/initrd.img-5.19.0-rc6-sev-step-999
     ae99
```

If so, the installation should be successful. Record its index in the output, where the index starts from 0. For example, if the output is:

```
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.0-49-generic
Found initrd image: /boot/initrd.img-6.8.0-49-generic
Found linux image: /boot/vmlinuz-5.19.0-rc6-sev-step-999ae99
Found initrd image: /boot/initrd.img-5.19.0-rc6-sev-step-999
     ae99
...
```

The index of the customized kernel is 2 (the third line, starting from 0). Now update the GRUB_DEFAULT in /etc/default/grub to boot into the new kernel by default.

```
GRUB_DEFAULT="1>[index]"
```

Here, [index] is the index of the new kernel in the GRUB menu just recorded. Then update GRUB configuration:

```
sudo update-grub
```

Reboot the machine and verify the kernel version with uname -r. It should output 5.19.0-rc6-sev-step-999ae99.

4. Compile the user-space projects for the experiments.

```
cd $RV_ROOT_DIR && make
```

A.3.2 Basic Test

Launch guest CVMs. Run the following command to launch **VM1** (Table 1):

```
$RV_ROOT_DIR/launch-vm1.sh
```

Upon a successful launch, a login prompt should appear in the terminal (pressing "Enter" is required sometimes). If the launch fails, check the script's output for potential troubleshooting instructions. Rebooting the host machine to reset the state if necessary.

For launching **VM2**, execute the following instruction in a new terminal (source ./env.sh first in it), which also kills **VM1** first due to the implementation limitation that **VM1** and **VM2** cannot be run simultaneously. When switching VMs, read the script output carefully for instructions and reboot the host machine if necessary.

```
$RV_ROOT_DIR/launch-vm2.sh
```

VM Name	qcow2 image	memoy-size (in MB)
VM1	aslr.qcow2	4096
VM2	big-disk.qcow2	4096

Table 1: CVMs used in the evaluation.

Test attack primitive. After launching a CVM, execute the following command in a different terminal than the one with the login prompt:

```
$RV_ROOT_DIR/t1_test-primitive.sh
```

If it succeeds, a green checkmark followed by "Successfully..." will appear on the screen when the script finishes.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The frequency distributions of plaintexts in the CVM are preserved when encrypted at the same system physical address (sPA), which can be exploited to learn the ciphertexts corresponding to the prevalent values in the CVM, such as zero (Section 1, Section 3.1).

(C2): The sparsity information in the guest page tables can be exploited to significantly reduce the number of possible guest virtual addresses (GVA) of the `glibc` library (Section 4, exemplified on `nginx`).

(C3): The sparsity information in the OpenVDB node buffers can be used to leak the structural information of the objects being processed under OpenVDB library operations (Section 5).

(C4): The sparsity information in the ReLU buffers can be decoded and utilized to leak information about tokens by placing probes on the blurred activation information (Section 6).

A.4.2 Experiments

Claim C1 is supported by experiment **E1-C1**. For claim C2, the main supporting experiment is **E2-C2**, plus an optional experiment **E3-C2**. In contrast to the main experiments, which complete within minutes, the experiments marked as (optional) take significantly longer (several hours) and are included for completeness. We support claim C3 with the main experiment **E4-C3** followed by optional experiments **E5-C3**, **E6-C3**, and **E7-C3**. Finally, we support claim C4 with the experiment **E9-C4** preceded by the experiment **E8-C4**.

(E1-C1): [**learning-zero-ciphertexts**] [1 compute-minutes + 3 human minutes] In this experiment, the attacker learns the ciphertexts corresponding to plaintext zeros in the CVM on the target page.
Preparation: (1) Launch a victim CVM:

```
$RV_ROOT_DIR/launch-vm1.sh # VM1 as an example
```

Execution: Once the CVM is launched, in another window (or shell, log-in session)[1], execute

```
$RV_ROOT_DIR/e1_ciphertext-learning.sh
```

Results: If it succeeds, a green check followed by "Successfully..." will appear on the screen when the script finishes. A detailed description of the contents of collected logs, including `$RV_ROOT_DIR/c1.log` and `$RV_ROOT_DIR/dmesg-output.log` is in `README.md`.

(E2-C2): [**ASLR**] [Around 6 compute-minutes + 5 human-minutes]: This experiment is a minimal demonstration of the ASLR de-randomization scenario, exemplified by de-randomizing the `glibc`'s base address of `nginx` with a single memory layout.
Preparation: (1) Launch the victim **VM1** for ASLR.

```
$RV_ROOT_DIR/launch-vm1.sh
```

Execution: Execute

```
$RV_ROOT_DIR/e2_aslr.sh
```

Results: The output logs are located in `$RV_ROOT_DIR/aslr/aslr-min-log/`. The detailed structure of this directory is provided in the "Description of the Output Logs" section of the `README.md` file. Upon successful execution, the directory `0/nginx-worker/` should contain a file whose name ends with `window_size_64.log`. The results of non-windowed and windowed trackings can be checked at the end of `0/nginx-worker/windowed_evaluation.log`, starting with the line "ASR metrics for service nginx-worker".

(E3-C2): [**ASLR-full (optional)**][5-6 compute-hours + 10 human-minutes] The spirit of this experiment is the same as E2, but extends to all five applications in Section 4 and evaluates 10 layouts for each. As a result, it takes around 50× more time than it takes for the **E2-C2** experiment.
Preparation: The same as **E2-C2**.
Execution: Execute

```
$RV_ROOT_DIR/e3_aslr-full.sh
```

Results: The output logs will be in `$RV_ROOT_DIR/aslr/aslr-all-log/`. They are similar to **E2-C2** but with more memory layouts and more applications for each layout. Upon completion, execute the following scripts to parse the logs and produce data for Table 2 in the paper.

```
cd $RV_ROOT_DIR/aslr
python ./scripts/aslr_track_analysis.py
```

(E4-C3): [**openvdb-traversal**] [10 compute-minutes, 15-25 human-minutes] This experiment leaks the object's structural information from OpenVDB's read-only traversal operation.
Preparation: (1) Launch the victim CVM **VM2**.

```
$RV_ROOT_DIR/launch-vm2.sh
```

Execution: Execute

```
$RV_ROOT_DIR/e4_openvdb-traversal.sh
```

Results: The scripts produce logs in `$RV_ROOT_DIR/openvdb-leak/openvdb-traversal-1-time/`. A detailed description of its structure is in `README.md` of the artifact package. In this directory, the file `output_all.txt` contains the page fault events during tracking. The scripts process the record of page-fault events and output `PLY` files (a 3D data format commonly used to

[1] Depending on your preference, you can achieve multiple shells by using `tmux` or logging in to a new session.

store point clouds and polygonal meshes). When the recovery succeeds, it should contain eight `PLY` files in `evaluation_traversal_extraction_<timestamp>_1/<ply-file-dir>/`.

Visualizing the extracted pieces and assembling them. Synchronize the directory `<ply-file-dir>` and the Blender project file `$RV_ROOT_DIR/openvdb-leak/blender_projects/traversal.blend` to your local machine. The object labeled "`original`" in the Blender project corresponds to the source object shown in Figure 8(a), with the source file provided as `$RV_ROOT_DIR/openvdb-leak/blender_projects/original.ply`.

Import the extracted `PLY` files in `<ply-file-dir>` into Blender and manually reassemble the eight fragments by adjusting their positions along the X, Y, and Z axes, as illustrated in `$RV_ROOT_DIR/openvdb-leak/blender_projects/blender-adjust-xyz.png`. Once assembled, toggle visibility to show only the extracted pieces, and export them as a single file, `traversal.ply`. We also provide analysis tools in `$RV_ROOT_DIR/openvdb-leak/blender_projects/`. For example, `visualize_traversal.py` visualizes `traversal.ply`, reproducing the effect shown in Figure 8. `p2p_dist_traversal.py` generates part of the comparison in Figure 9 of the paper, contrasting the distribution of nearest-neighbor distances between objects extracted from the traversal operation and randomly populated points.

(E5-C3): [**openvdb-traversal-3-trackings** (**optional**)] [Triple the time required for E4 + around 6 GB disk space for storing the logs.] Everything remains the same as in **E4-C3**, except the experiment is repeated three times for evaluation.

Preparation: The same as **E4-C3**.

Execution: The same as **E4-C3**'s execution, but pass a different number as the number of times to run as a parameter. Execute

```
$RV_ROOT_DIR/e5_openvdb-traversal-3.sh
```

Results: The same as **E4-C3** but applied to more tracking instances.

(E6-C3): [**openvdb-construction** (optional)] [Around 3 compute-hours + 15-25 human-minutes + around 80GB disk space for storing logs] This experiment leaks the object's structural information from OpenVDB's construction operation.

Preparation: The same as **E4-C3**.

Execution: Execute

```
$RV_ROOT_DIR/e6_openvdb-construction.sh
```

Results: The same as **E4-C3**, but applied to a different victim's operation.

(E7-C3): [**openvdb-construction-3-trackings (optional)**] [Triple the time required for E6-C3 + around 240 GB disk space for storing the logs.] Everything remains the same as in **E6-C3**, except the experiment is repeated three times.

Preparation: The same as **E4-C3**.

Execution: Execute

```
$RV_ROOT_DIR/e7_openvdb-construction-3.sh
```

Results: The same as **E6-C3** but applied to more tracking instances.

(E8-C4): [**sparse-llm-data**] [5–6 compute-hours + 3 human-minutes + around 230GB disk.] This experiment produces data for **E9-C4**.

Execution: Execute

```
$RV_ROOT_DIR/e8_sparse-llm-data.sh
```

Results: The result directory `$RV_ROOT_DIR/sparsellm-probe/powerinfer_reduced_activation_probes_results/` is generated.

(E9-C4): [**sparse-llm-figures**] [1 compute-minute + 3 human-minutes] Visualize the results of **E8-C4**.

Preparation: Finish **E8-C4**.

Execution: Execute

```
$RV_ROOT_DIR/e9_sparse-llm-figures.sh
```

Results: Figures 11 and 12 in the paper can be produced as `probe_results.png` and `world_large_reduce4.png` in the `sparsellm-probe` directory when the execution succeeds.

A.5 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/.