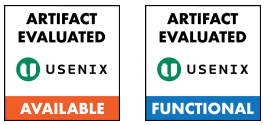

USENIX Security '26 Artifact Appendix: Side-Channel Attacks on Open vSwitch

Daewoo Kim
University of Waterloo

Sihang Liu
University of Waterloo

A Artifact Appendix

A.1 Abstract

Virtualization is widely adopted in cloud systems to manage resource sharing among users. A virtualized environment usually deploys a virtual switch within the host system to enable virtual machines to communicate with each other and with the physical network. The Open vSwitch (OVS) is one of the most popular software-based virtual switches. It maintains a cache hierarchy to accelerate packet forwarding from the host to virtual machines.

We provide code that characterizes the OVS caching system from a security perspective and identifies key attack primitives. Based on these primitives, the artifact demonstrates three remote attacks that break isolation guarantees in virtualized environments. First, the artifact enables a remote covert channel in which a sender transmits bits from outside the server while a receiver observes cache behavior inside the server. Second, it presents a novel header recovery attack that probes microflow cache entries and recovers a remote user's IP address and port number using the known hash function. Third, the artifact demonstrates a remote packet-rate monitoring attack that exploits megaflow subtables to infer a remote victim's packet rate. Finally, three mitigation strategies that successfully prevent attacks are included.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All experiments are done in local, isolated environments. We do not identify any risk to users of the artifact associated with this artifact evaluation procedure.

A.2.2 How to access

Our artifact can be downloaded from Zenodo at https://doi.org/10.5281/zenodo.17965902.

A.2.3 Hardware dependencies

The experiments require the following hardware platform:

- Two servers, each equipped with at least a 1 Gbps NIC.
- A 1 Gbps one-hop ethernet connection between the two servers.

A.2.4 Software dependencies

The experiments require the following software platform:

- Ubuntu 22.04.
- Linux kernel version 6.5.0.
- Open vSwitch version 2.17.9 (`https://www.openvswitch.org/download/`).
- DPDK version 21.11.9 (`https://core.dpdk.org/download/`).
- Memcached version 1.6.21 (`https://github.com/memcached/memcached/wiki/ReleaseNotes`).

A.2.5 Benchmarks

The experiments require the following workloads:

- **ClassBench-ng** (`https://classbench-ng.github.io/`).
- **UNSW-NB15** (`https://research.unsw.edu.au/projects/unsw-nb15-dataset`).

A.3 Set-up

A.3.1 Installation

`Sudo` permission may be required to execute the following commands. First, run `install_ovs_dpdk.sh` on server1 to install Open vSwitch version 2.17.9 with DPDK version 21.11.9. Then, run `vm_setup.sh` on server1 to create two virtual machines and attach them to an OVS bridge.

A.3.2 Basic Test

We provide a simple test to verify that all required dependencies are installed correctly. Run `Basic_Test/0_Basic_Test.sh`. The output should include a measured RTT.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Figure 7 presents the RTT distributions for microflow cache hits and megaflow cache hits, demonstrating that their RTTs are distinguishable.
(C2): Table 2 presents the results of covert data transmission using the microflow cache and the megaflow cache.
(C3): Table 4 presents the results of recovering the user's IP address and port number.
(C4): Figure 20 presents the results of monitoring the user's packet rate.
(C5): Table 5 presents the evaluation results of three defense mechanisms.

A.4.2 Experiments

(E1): [Characterization] [1 compute-hour]: Characterization logs the RTT to a remote server by sending Memcached GET requests. It measures RTTs in the case of a microflow cache hit and RTTs in the case of a megaflow cache hit.
Preparation: Run `vm_setup.sh` on server1.
Execution: Run `Characterization/1_Characterization.sh` on server2.
Results: The results should match Figure 7. The average RTTs is used as a threshold in the following experiments.

(E2): [Covert Channel] [1 compute-hour]: Covert Channel consists of a sender and a receiver. It randomly generates an 8-bit payload and transmits a 10-bit sequence with a 10 header. To transmit a bit 1, the sender sends packets to the Memcached server; to transmit a bit 0, it remains idle. The receiver continuously measures RTTs to the Memcached server. Reception starts when a high RTT is detected, which indicates the beginning of the header. For each bit, the receiver averages the RTT of 100 measurements, and logs the recovered bit based on the average.
Preparation: Both server1 and server2 run two virtual machines each. On server1, run the receiver VM and a Memcached VM. On server2, run the sender VM and a Memcached VM. Use `vm_setup.sh` to set up the VMs on both servers. Configure Memcached with the -l 0.0.0.0 option so that it listens on all available network interfaces.
Execution: Run `Attacks/2_Covert_channel_receiver.sh` on server1 and `Attacks/2_Covert_channel_sender.sh` on server2.
Results: The results should match Table 2.

(E3): [Header Recovery] [1 compute-hour]: Header recovery first probes all 8192 microflow cache entries by sending packets to the Memcached server and records the RTTs. The two entries with the highest RTTs correspond to entries used by the victim. To identify the remaining victim entries, the code repeats the probing process using different packet flows and identifies another two high RTT entries. Lastly, it reverse-engineers the victim's IP address and port number using the four detected entries.
Preparation: Both server1 and server2 run two virtual machines each. On server1, run the victim VM and a Memcached VM. On server2, run the attacker VM and a Memcached VM. Use `vm_setup.sh` to set up the VMs on both servers. Configure Memcached with the -l 0.0.0.0 option so that it listens on all available network interfaces.
Execution: Run `Attacks/3_Header_recovery_victim.sh` on server1 and `Attacks/3_Header_recovery.sh` on server2.
Results: The results should match Table 4.

(E4): [Rate Monitoring] [1 compute-hour]: Rate monitoring logs the range of monitored packet rates. It periodically sends packets to the Memcached server to control the ordering of megaflow cache subtables. The setup includes four threshold subtables and one subtable shared with the victim. By measuring RTTs to these five subtables, it infers the range of the victim's packet rate. This range is adaptively doubled or halved based on the result from the previous round.
Preparation: Both server1 and server2 run two virtual machines each. On server1, run the victim VM and a Memcached VM. On server2, run the attacker VM and a Memcached VM. Use `vm_setup.sh` to set up the VMs on both servers. Configure Memcached with the -l 0.0.0.0 option so that it listens on all available network interfaces.
Execution: Run `Attacks/4_Rate_monitoring_victim.sh` on server1 and `Attacks/4_Rate_monitoring.sh` on server2.
Results: The results should match Figure 20.

(E5): [Mitigation] [1 human-hour + 3 compute-hour]:
Preparation: Repeat the installation after replacing `dpif-netdev.c` and `dpif-netdev-private-dfc.h`. The logic for the microflow cache and megaflow cache can be modified in these two files.
Execution: Repeat E1, E2, E3, and E4 on server2.
Results: The results should match Table 5.

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/usenixsec2026/.