CacheWarp: Software-based Fault Injection using Selective State Reset

Ruiyi Zhang, Lukas Gerlach, Daniel Weber, and Lorenz Hetterich,
CISPA Helmholtz Center for Information Security; Youheng Lü, *Independent;*
Andreas Kogler, *Graz University of Technology;* Michael Schwarz,
CISPA Helmholtz Center for Information Security

https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-ruiyi

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: CacheWarp: Software-based Fault Injection using Selective State Reset

Ruiyi Zhang
CISPA Helmholtz Center for Information Security

Lukas Gerlach
CISPA Helmholtz Center for Information Security

Daniel Weber
CISPA Helmholtz Center for Information Security

Lorenz Hetterich
Independent

Youheng Lü
Independent

Andreas Kogler
Graz University of Technology

Michael Schwarz
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

Our paper presents CacheWarp, a new software-based fault attack on AMD SEV-ES and SEV-SNP, exploiting the possibility to architecturally revert modified cache lines of guest VMs to their previous (stale) state. Unlike previous attacks on the integrity, CacheWarp is not mitigated on the newest SEV-SNP implementation, and it does not rely on specifics of the guest VM. CacheWarp only has to interrupt the VM at an attacker-chosen point to invalidate modified cache lines without them being written back to memory. This artifact demonstrates two attacking primitives, DropForge and TimeWarp, exploiting the explicit and implicit memory writes individually. We combine Prime+Probe and a stepping framework with Cachewarp to achieve reliable selective state resets, resulting in control-flow hijacking attacks. Finally, we show how CacheWarp can be mounted "blindly" to recover the full private key of RSA-CRT in the Intel IPP crypto library. This artifact includes all the code necessary to reproduce the proof-of-concepts.

A.2 Description & Requirements

This artifact includes five main experiments: 1) Last-level cache eviction; 2) Selective state reset; 3) Proof of Concept (PoC) of DropForge; 4) PoC of TimeWarp; 5) Bellcore Attack. All experiments require an AMD EPYC machine below the 4th generation. For all experiments except the first, you must be able to launch AMD SEV-ES/SEV-SNP VMs.

A.2.1 Security, privacy, and ethical concerns

Cache eviction does not pose any security risks. However, the instruction used for CacheWarp, "INVD", and APIC configuration, are likely to hang the machine if used without caution.

A.2.2 How to access

The artifact is available on GitHub: https://github.com/cispa/CacheWarp/tree/ae.

A.2.3 Hardware dependencies

This artifact requires AMD SEV VMs. Therefore, the artifact requires one of the following hardware:

- **a 2nd generation EPYC CPU with SEV-ES support**.
- **a 3rd generation EPYC CPU with SEV-SNP support**.

In the paper, we used an 8-core AMD EPYC 7252 CPU for SEV-ES and AMD EPYC 7313P and 7443 CPUs for SEV-SNP.

A.2.4 Software dependencies

All our experiments are tested on Ubuntu 22.04 LTS (Linux kernel 6.1.0). QEMU and OVMF are required to launch SEV guest VMs, as referenced from AMD's official GitHub repository[1]. Finally, we use the libtea framework to modify page table entries and configure the APIC timer[2].

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

To access the artifact, clone the repo https://github.com/cispa/CacheWarp.git and checkout to tag "ae".

[1] https://github.com/AMDESE/AMDSEV
[2] https://github.com/libtea/frameworks

```
git clone https://github.com/cispa/CacheWarp.git
cd CacheWarp
git checkout ae
```

To install dependency, you will need to install:

```
sudo apt install cpufrequtils msr-tools -y
```

To compile the "libtea", navigate to folder "frameworks/libtea":

```
git submodule update --init
cd frameworks/libtea
make libtea-x86-interrupts
```

To prepare the host OS and guest VM, following the guidance from AMD:

```
git clone https://github.com/AMDESE/AMDSEV.git
cd AMDSEV
git checkout sev-es (or snp-latest)
```

A.3.2 Basic Test

Test if the `INVD` instruction can be enabled.

```
sudo modprobe msr;
CUR=$(sudo rdmsr 0xc0010015);
ENABLED=$(printf "%x" $((0x$CUR & ~16)));
sudo wrmsr -a 0xc0010015 0x$ENABLED
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): We show how the hypervisor can selectively drop dirty cache lines, partially writing back the cache before its invalidation. This is proven by the experiment (E1, E2) described in Section 5.2, with the results shown in Figure 7.

(C2): With Dropforge (Section 4.3), we change the behavior of functions inside the victim VM.

(C3): With Timewarp (Section 4.4), we redirect return values to an earlier point in the victim program's control flow.

(C4): We demonstrate that CacheWarp can perform a Bellcore attack on RSA-CRT (Section 6.1), fully recovering the private key given a single faulty signature.

A.4.2 Experiments

(E1): Eviction [5 minutes]:

How to: Find a pattern to evict a L1/L2 cache line to the non-inclusive L3 cache and then evict it to memory.

Preparation: Enable huge pages and reduce the noise.

```
sudo sysctl -w vm.nr_hugepages=32

# Check cpufreq-info
sudo cpufreq-info -c 0 | grep "available"
# Denoise - fix the frequency
sudo cpufreq-set -c <CORE> -g userspace
sudo cpufreq-set -c <CORE> -f <FREQ>MHz
```

Execution: Run it with sudo.

```
cd l2-l3-prime
make
sudo ./hist <CORE>
```

Results: A reliable eviction pattern will be printed in the terminal.

(E2): Selective state reset [2 hours + (Additionally 1 day to launch AMD SEV VMs)]:

How to: Incorporate the eviction pattern obtained from E1 into the host kernel. To set up the victim, launch a SEV-ES/SNP machine. Next, compile and run "pocs/generic-writes-drop.c" inside the victim VM. The attacker chooses to drop memory modifications residing in only one cache set.

Preparation: Recompile the host kernel and launch a SEV-ES/SNP VM.

```
cd AMDSEV/build/linux
# Apply the kernel patch for the host

# Specify one core <X> to test this artifact
sudo cat /sys/devices/system/cpu/cpu<X>/cache/
    index3/shared_cpu_list

# Offline other cores within the CCX
echo 0 | sudo tee /sys/devices/system/cpu/cpu<
    Y>/online
```

Execution: Compile and execute in both the victim and attacker folders:

```
vm> sudo apt update && sudo apt -y install vim
     build-essential
vm> sudo sysctl -w vm.nr_hugepages=1
vm> gcc generic-writes-drop.c -O2 -o generic-
    writes-drop
hv> cd attacker && make

vm> ./generic-writes-drop
hv> sudo ./cachewarp_blind_drop 70 100 225
```

Results: You can observe a miscalculated sum in the terminal only with the correct cache set index (225).

```
## Expected Output (in the VM)
phys: 0x43c00000
phys: 0x43c03840
result: 39999998815 (similar, i.e., not
    40000000000)
```

(E3): DropForge [10 minutes]:

How to: Compile "dropforge.c" and run it inside the victim VM. The attacker iteratively drops each cache set.

Execution: Run the victim and attacker:

```
vm> gcc dropforge.c -o victim
vm> ./victim
hv> sudo ./invd.sh
```

Results: The attacker iteratively drops each cache set. A value will be printed to the terminal once the cache set containing the stack frame is dropped.

(E4): TimeWarp [10 minutes]:

How to: Drop the return address of the call instruction.

Preparation: Compile "timewarp.c" and run it inside the victim VM.

```
gcc timewarp.c -o timewarp
```

Execution: Run the victim and attacker poc:

```
vm> ./timewarp
hv> sudo ./invd.sh
```

Results: An unreachable path will be executed.

```
# Expected Output (in the VM)
"Win!"
```

(E5): Bellcore Attack:

How to: Using RSA-CRT to sign inside the guest VM until the hypervisor injects a fault, resulting in a faulty signature.

Preparation: Copy the folder "rsa-crt" into the guest VM and compile it.

```
vm> make all
```

Execution: Keep signing in a loop until a fault results in a different signature. If the signing loop does not end, repeat the drop multiple times from the hypervisor.

```
vm> make exploit
hv> sudo ./cachewarp_blind_drop 70 100 225 0
```

Results: The prime numbers P and Q will be recovered and printed to the terminal.

```
vm> python3 exploit.py
```

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/usenixsec2024/.