USENIX Security '26 Artifact Appendix: LoongLeak: Architectural Cross-Privilege-Boundary Data Leakage on LoongArch CPUs

Lorenz Hetterich, Tristan Hornetz, Fabian Thomas, Michael Schwarz
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

This artifact provides proof-of-concept (PoC) implementations for LoongLeak, an architectural vulnerability affecting Loongson 3A5000 and 3A6000 CPUs. The artifact includes several case studies: a basic leakage example, evaluation for leakage speed and accuracy, a covert channel between processes, a proof-of-concept breaking ASLR and leaking stack canaries, a proof-of-concept leaking partial root password hashes, and a proof-of-concept extracting disk encryption keys. Additionally, it contains a differential fuzzer used to discover the vulnerability.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact contains proof-of-concept exploits. Running these on your own machines (if you have Loongson hardware) will demonstrate that sensitive data (like kernel memory or other users' data) can be leaked. One PoC (`sudo_hash_poc`) specifically targets the root password hash. The vulnerability was responsibly disclosed and is under embargo until the first conference day.

A.2.2 How to access

The artifact is available on Zenodo:
https://zenodo.org/records/17974800

A.2.3 Hardware dependencies

The artifact requires Loongson CPUs based on the LoongArch ISA. Specifically:

- **Loongson 3A5000**: Used for most experiments (referenced as `lab86` in instructions).
- **Loongson 3A6000**: Used for the `sudo_hash_poc` experiment (referenced as `lab85`).

Reviewers are provided with SSH access to these two machines. The private key (`artifact`) and SSH configuration (`ssh_config`) are contained in the artifact and will be valid during the artifact evaluation period only.

A.2.4 Software dependencies

The following software is required on the reviewer's machine (for cross-compilation). The instructions below assume Ubuntu 24.04:

- `python3` (standard `python3` package)
- `termcolor` Python package
- `qemu-loongarch64` (`qemu-user` package)
- `loongarch64-linux-gnu-gcc` (`gcc-14-loongarch64-linux-gnu` package). After installation, create a symlink (as root):

```
ln -s /usr/bin/loongarch64-linux-gnu-gcc-14 \
    /usr/bin/loongarch64-linux-gnu-gcc
```

- `binutils-loongarch64-linux-gnu`
- `make`, `git`, `unzip`, `zip`

The provided machines run Loongnix-20.6 (Linux kernel 4.19) and have all required software installed.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Navigate to the root of the artifact. Most components can be compiled by running `make` in their respective subdirectories. For cross-compilation, ensure the `loongarch64-linux-gnu-gcc` compiler is in your path.

A.3.2 Basic Test

The most basic test is the `leak_primitive`. It demonstrates leakage within a single process.
Execution:

```
cd leak_primitive; make
scp primitive lab86:
ssh lab86 ./primitive
```

Expected results: The output should show multiple 32-byte blocks. Some rows should contain the sequence `1000 1001 ... 101f`, which is the data from the target cache set in the victim buffer.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): LoongLeak can leak data across security domains (kernel, other userspace processes) without side channels. (Proven by E1, E3, E4, E5, E6)
(C2): The leakage is fast, achieving rates beyond 300 MB/s. (Proven by E2)
(C3): LoongLeak allows targeted leakage of specific cache lines and ways. (Proven by E1)
(C4): LoongLeak can break traditional software defenses like ASLR and stack canaries. (Proven by E4)
(C5): LoongLeak can be discovered using differential fuzzing between hardware and QEMU. (Proven by E7)

A.4.2 Experiments

More detailed instructions for each experiment are provided in the `README.md` file within the artifact.
(E1): [leak_primitive] [5 human-minutes]: Basic demonstration of LoongLeak.
Preparation:

```
cd leak_primitive; make
scp primitive lab86:
```

Execution:

```
ssh lab86 ./primitive
```

Results: Successful leakage of the target pattern `1000 1001 ... 101f`.
(E2): [evaluate_primitive] [5 human-minutes]: Measure leakage speed and accuracy.
Preparation:

```
cd evaluate_primitive; make
scp eval_primitive lab86:
```

Execution:

```
ssh lab86 ./eval_primitive
```

Results: Output shows `correct` and `time`. Leakage rate (in $\frac{MB}{s}$) is calculated as:

$$\frac{16384000000 \cdot 100000000}{time \cdot 1024 \cdot 1024}$$

Expected rate >300$\frac{MB}{s}$, accuracy ($\frac{correct}{16384000000}$) >80%.
(E3): [covert_channel] [5 human-minutes]: Data transfer between two processes.
Preparation:

```
cd covert_channel; make;
scp sender receiver lab86:
ssh lab86 "echo 'Hello World' > in.txt"
```

Execution:
In terminal 1:

```
ssh lab86 taskset -c 0 ./receiver out.txt
```

In terminal 2:

```
ssh lab86 taskset -c 0 ./sender in.txt
```

The sender in terminal 2 might not terminate automatically. Thus, you may need to terminate it manually.
Results: `out.txt` on `lab86` matches the `in.txt`.
(E4): [aslr_break] [5 human-minutes + 15 compute-minutes]: Leak stack canary and break ASLR of FFmpeg.
Preparation:

```
scp -r aslr_break lab86:
```

On lab86:

```
cd aslr_break
make
git clone --recursive \
  https://github.com/FFmpeg/FFmpeg.git \
  --branch=n7.1.1 --depth=1
cd FFmpeg
git apply ../0001-Print-Stack-Canary.patch
./configure --target-os=linux \
  --arch=loongarch \
  --prefix=/home/reviewer/ffmpeg \
  --extra-cxxflags="-fstack-protector-all -g" \
  --extra-cflags="-fstack-protector-all -g" \
  --enable-gpl --disable-stripping \
  --enable-shared --disable-static
make -j$(nproc)
make install
```

Execution: In Terminal 1:

```
ssh lab86 'cd aslr_break;
  taskset -c 0 sh start_victim.sh'
```

In Terminal 2:

```
ssh lab86 'cd aslr_break;
    taskset -c 0 ./canary_leak;
    taskset -c 0 ./aslr_break $(pidof ffmpeg);
    killall ffmpeg'
```

Results: Leaked canary matches the one printed by the victim; `aslr_break` correctly estimates the stack address.

(E5): [sudo_hash_poc] [10 human-minutes]: Leak partial root password hash on 3A6000.

Preparation:

```
cd sudo_hash_poc; make;
scp sudo_hash_poc pwd.sh lab85:
```

Execution: In Terminal 1:

```
ssh lab85 "sh pwd.sh 1; killall sh"
```

In Terminal 2:

```
ssh lab85 taskset -c 0 ./sudo_hash_poc
```

Once you see leaked password hashes in terminal 2, you can terminate `sudo_hash_poc` in terminal 2 and `pwd.sh` in terminal 1.

Results: Output contains substrings of the root password hash (`$6$7oJFvh5yRlbpYQyG$nCzIdp3HfmVTxy...` in the provided setup).

(E6): [disk_encryption_leak] [10 human-minutes]: Recover AES key from kernel.

Preparation: Since preparing this proof-of-concept requires root, it is already prepared on the provided `lab86` machine. For more details on how to set it up on your own Loongson 3A5000 machine, refer to the `README.md` in the artifact.

Execution: Terminal 1:

```
ssh lab86 "cd disk_encryption_leak;
  ./attacker"
```

Terminal 2:

```
ssh lab86 "cd disk_encryption_leak;
  cd leak_primitive; ./primitive;
  killall attacker"
```

Make sure to terminate the attacker in terminal 1 once you are done.

Results: Leaked key matches the expected AES key used for disk encryption.

(E7): [fuzzer] [10 human-minutes + 15 compute-minutes]: Discover LoongLeak via fuzzing.

Preparation: All commands are executed in the `fuzzer` directory.

```
make fuzzer_sw fuzzer_hw
```

Execution:

1. Run the fuzzer on QEMU (on your device):

```
python3 fuzzer.py
```

2. Modify `QEMU=True` to `QEMU=False` in `fuzzer.py`.

3. Copy the required files to `lab86`:

```
scp fuzzer_hw fuzzer.py bitmap.py lab86:
```

4. Run the fuzzer on lab86:

```
ssh lab86 python3 fuzzer.py
```

5. Download the results:

```
ssh lab86 zip -r out_hardware out_hardware
scp lab86:out_hardware.zip .
ssh lab86 rm -rf out_hardware.zip out_hardware
unzip out_hardware.zip
```

6. Run the analysis:

```
python3 comparer.py
```

Results: Among others, `comparer.py` identifies semantic differences in floating-point or vector load instructions like `fld.s`, `fld.d`, and `vld`.

A.5 Version

This document is based on the LaTeX template for Artifact Evaluation V20231005. The submission, reviewing, and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.