USENIX Security '26 Artifact Appendix: Injected and Leaked: Actively Inducing Side-Channel Leakage Using Electromagnetic Injection and Hardware Nonlinearity

Haoran Yan[1], Ziyu Shao[1], Shuhao Zhang[1], Qinhong Jiang[2], Yan Long[1]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]The Hong Kong Polytechnic University

A Artifact Appendix

A.1 Abstract

This artifact supports the audio-eavesdropping case studies and the speech-enhancement pipeline of the paper. It comprises: (i) released case-study audio for cross-wall eavesdropping, closed-loop manipulation, and long-distance eavesdropping; (ii) the speech-enhancement code; (iii) a released model checkpoint; (iv) preprocessing, training, and inference scripts; and (v) released noise samples. The goal of the artifact is to enable evaluators to inspect the key results and claims described in the paper, including the released case stud audio evidence and the speech-enhancement pipeline that reconstructs intelligible speech from recovered eavesdropped audio. Consistent with the paper's responsible-release policy, security-sensitive operational details that could directly enable misuse, including vulnerable injection frequencies and closed-loop control logic, are intentionally omitted.

A.2 Description & Requirements

The artifact contains the public open-science materials focusing on released case-study audio recordings and the speech-enhancement codebase described in the paper. The artifact is organized as follows.

Case study audio/ contains the recordings for cross-wall eavesdropping, closed-loop manipulation, and long-distance eavesdropping described in Section 5.

`Cross-wall eavesdropping/` contains the cross-wall case-study audio. `Clean/` holds the clean source utterances (male and female versions) covering privacy- and confidentiality-sensitive content: a personal meeting location, a bank-card PIN and transfer amount, and a business price quote. The two `Eavesdropped/` subdirectories contain the corresponding speech recovered through the injection-induced EM side channel in a home/hotel-style private setting and a meeting-room business setting, supporting validation of the paper's cross-wall eavesdropping results.

`Closed-loop manipulation/` contains the audio materials for the landline-phone closed-loop manipulation case study. `Clean audio/` contains original audio generated directly through TTS. `Eavesdropped audio/` contains Alice's speech recovered through the injection-induced side channel. `Enhanced eavesdropped audio/` contains the recovered speech after applying the released speech-enhancement pipeline. `Synthesized audio/` contains adversarial speech generated to alter the conversation, such as changing a quote, reversing an agreement, or changing navigation instructions. `Audio recorded from Alice/` contains victim-side microphone recordings of Alice's speech. `Audio recorded from injection/` contains victim-side microphone recordings of the injected malicious speech. Together, these files contain audio samples across the eavesdrop-synthesize-inject chain described in the paper.

`Long distance eavesdropping/` contains the original audio generated directly through TTS and eavesdropped audio samples at 12 m, 18 m, 24 m, and 30 m, corresponding to the extended-range eavesdropping experiment enabled by higher-power EM injection.

Enhance Code/ contains the SGMSE-based speech-enhancement pipeline used to enhance recovered audio described in Section 3.4. It includes preprocessing, training, and enhancement scripts; the model implementation; package requirements; the training configuration; released noise samples; and a released checkpoint with sensitive textual metadata.

The top-level `README.md` is the primary guide for artifact evaluation. It documents the artifact scope, directory layout, external dataset provenance, and commands for exercising the speech-enhancement pipeline.

A.2.1 Security, privacy, and ethical concerns

Consistent with the paper's responsible-release policy, this artifact is carefully scoped to avoid disclosing security-sensitive operational details that could directly facilitate misuse, including vulnerable injection frequencies and active injection control logic.

As with any software artifact, inspectors or users should exercise ordinary caution when installing dependencies and loading machine-learning checkpoints, as these actions involve the standard risks associated with executing third-party code. Overall, the released materials are intended to enable artifact evaluation and evidence inspection while preserving the responsible-disclosure boundary of the work.

A.2.2 How to access

The artifact is publicly archived on Zenodo at the concept DOI https://doi.org/10.5281/zenodo.18500378. The project website containing demo video is available at https://injecteave.github.io/.

A.2.3 Hardware dependencies

The authors successfully ran the released speech-enhancement pipeline on NVIDIA A800 and GeForce RTX 4090 GPUs. For practical inference and optional training, we recommend an NVIDIA CUDA-capable GPU with more than 16 GB of GPU memory.

A.2.4 Software dependencies

The pipeline targets Python 3.8+ with CUDA 11.8+ recommended. Python package requirements are listed in `Enhance Code/requirements.txt` (including `torch`, `torchaudio`, `librosa`, `soundfile`, `pytorch-lightning`, and related packages) and can be installed with a single `pip` command in A.3.1.

A.2.5 Benchmarks

The main evaluation workloads are the released case-study audio files under `Case study audio/`, including cross-wall eavesdropping audio, closed-loop manipulation audio, and long-distance eavesdropping audio. These files support evaluators to inspect the released evidence and compare clean reference audio with corresponding recovered, enhanced, synthesized, or victim-side recorded audio where applicable.

`Enhance Code/checkpoints/model.ckpt` contains the trained speech-enhancement model, allowing evaluators to run inference directly on released eavesdropped audio without retraining. The released noise samples (`Enhance Code/data/noise/`, 118 WAV files) support the documented preprocessing pipeline and future reuse of the data-generation workflow.

The clean-speech source for constructing paired enhancement training data is the LibriSpeech `train-clean-100` subset (OpenSLR resource SLR12, archive `train-clean-100.tar.gz`, CC BY 4.0), available from http://www.openslr.org/12/. LibriSpeech provides clean speech utterances used as ground-truth targets when synthesizing clean/noisy training pairs. It is needed for optional preprocessing, training, or reuse experiments.

A.3 Set-up

A.3.1 Installation

Download and extract the artifact from the Zenodo DOI https://doi.org/10.5281/zenodo.20432240. All commands are run in the `Enhance Code/` directory. Install the Python dependencies:

```
cd "Enhance Code"
python -m pip install -r requirements.txt
```

A.3.2 Basic Test

Run a single-file inference on one released eavesdropped recording inside `Enhance Code/` after installation:

```
cd "Enhance Code"
python enhancement.py \
  --test_file "../Case study audio/Closed-loop manipulation
    ↪ /Eavesdropped audio/Yes, turn right and you will
    ↪ see the hotel.wav" \
  --enhanced_dir enhanced_audio \
  --ckpt checkpoints/model.ckpt \
  --N 30 --device cuda
```

Each path is a single logical line; a hooked arrow (↪) marks only visual wrapping within the column, so no file name is broken across a line continuation.

Expected behavior: the script loads the checkpoint (`Model loaded`), writes one enhanced WAV file under `enhanced_audio/`, prints the saved output path (`Enhanced file saved to: ...`), and finishes with `Enhancement completed! Output: enhanced_audio`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *The artifact provides the released audio evidence supporting the paper's case studies in Section 5: cross-wall eavesdropping in Section 5.1, closed-loop landline-phone manipulation in Section 5.2, and long-distance eavesdropping in Section 5.3. The released files are organized so that clean reference audio can be directly inspected with corresponding eavesdropped, enhanced, synthesized, or victim-side recorded audio where applicable.* This is validated by experiment (E1).

(C2): *The artifact provides the SGMSE-based speech-enhancement pipeline described in Section 3.4. Using the released checkpoint, evaluators can run inference on a released eavesdropped audio sample and obtain an enhanced output, corresponding to the paper's claim that enhancement improves recovered audio intelligibility.* This is validated by experiment (E2).

A.4.2 Experiments

(E1): *Inspect released layout and audio* [15 human-minutes + negligible compute + <1 GB disk]: confirm the released audio evidence is present and well-formed.
How to: Inspect the directory tree and audio with any local audio tool.
Preparation: Extract the artifact.
Execution: verify the three case-study groups. Inspect one released audio file with any local audio tool.
Results: The directory structure matches the artifact README, and the audio plays successfully, supporting (C1).

(E2): *Speech-enhancement inference* [15 human-minutes + ∼minutes compute on GPU + <1 GB disk]: enhance a released eavesdropped recording with the released checkpoint.
How to: Run the Basic Test inference command.
Preparation: Complete Set-up (Installation). A CUDA GPU is recommended; CPU execution has not been validated by the authors and may be slow or fail depending on PyTorch/checkpoint loading.
Execution: From `Enhance Code/`, run `enhancement.py` with `--ckpt checkpoints/model.ckpt` on a file from `Case study audio/Closed-loop manipulation/Eavesdropped audio/`.
Results: One enhanced WAV file is written to `enhanced_audio/`, and the completion message is printed. The enhanced output can be compared against the corresponding file in `Closed-loop manipulation/Enhanced eavesdropped audio/`, supporting (C2).

A.5 Notes on Reusability

The speech-enhancement component is designed to be reusable beyond the paper's case studies. Because the model is trained on paired data built from public clean speech and released noise samples—rather than device-specific recordings—it learns general distortion and noise patterns instead of memorizing a particular device, speaker, or environment. As discussed in Section 3.4, this yields strong generalization across injection-induced side-channel settings: the same checkpoint applies to recovered audio from different devices and scenarios without device-specific retraining, and it may also have value as a general-purpose speech-enhancement and audio-denoising component outside this context.

The released code exposes separate preprocessing, training, and inference entry points. Users can apply the released checkpoint directly to new noisy audio with `enhancement.py`, build new paired data with `preprocess.py` from their own clean-speech and noise sources, or fine-tune the model on a small amount of domain-specific paired data to adapt it to a new acoustic environment or sensing modality.

The preprocessing and training scripts are provided as illustrative and reusable workflow components. They document how paired enhancement data can be constructed and how the model can be trained or fine-tuned for future use, but they are not intended to regenerate the exact released checkpoint. The training settings (model, STFT, SDE, and optimization) are documented in `configs/train_config.yaml` for traceable adaptation.

Optional data-construction and training checks

The following optional checks exercise the data-construction and training code paths and require downloading an external dataset. They are intended to illustrate and reuse the preprocessing/training workflow, not to reproduce the exact training set or regenerate the released checkpoint.

- *Preprocessing* [30 human-minutes + ∼tens of minutes compute + a few GB disk]: download `train-clean-100.tar.gz` from OpenSLR SLR12, then run `preprocess.py` on a small subset with the released `./data/noise` as `--noise_dir` and the SNR range documented in the artifact README (`--snr_min -15 --snr_max 15`). Paired `clean`/`noisy` files appear under the chosen `--output_dir`, exercising the nonlinear-simulation data pipeline.
- *Training*: `train.py`, configured through `configs/train_config.yaml`, is provided for completeness. It is not intended to reproduce the paper's training run or regenerate the released checkpoint.

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/usenixsec2026/.