USENIX Security '26 Artifact Appendix: HAMLOCK: HArdware-Model LOgically Combined attacK

Sanskar Amgain[1*], Daniel Lobo[2*], Atri Chatterjee[2*], Swarup Bhunia[2], Fnu Suya[1]

[1]University of Tennessee [2]University of Florida

A Artifact Appendix

A.1 Abstract

This artifact implements HAMLOCK, a hardware–software co-design backdoor in which a small software-side trigger pathway is paired with a hardware Trojan payload, so that the backdoored model behaves benignly in software-only inspection yet misclassifies triggered inputs once deployed on the Trojaned hardware. It provides the three attack variants studied in the paper—single-neuron trigger optimization, single-neuron weight optimization, and the multi-neuron attack—together with a functional emulation of the hardware Trojan (trigger detection via sign-/exponent-bit comparison and payload bias injection), so no physical FPGA or ASIC is required to reproduce the model-level results. The artifact further includes evaluation harnesses for six representative defenses spanning black-box and white-box backdoor-sample detection (STRIP, BBCaL, IBD-PSC, TED) and lightweight backdoor mitigation (fine-tuning, fine-pruning, and CLP). Running the artifact trains the clean models, injects the backdoors, and produces the backdoored checkpoints and per-defense logs that reproduce the paper's major claims (C1–C4) on MNIST, CIFAR-10, GTSRB, and ImageNet with LeNet, VGG-16, and ResNet-18. Reproducing claims C1–C4 requires only a single CUDA-capable GPU and the open-source software in `requirements.txt`.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Executing this artifact poses no risk to the evaluator's machine. The artifact performs no destructive operations, requires no elevated privileges, and does not disable, weaken, or interfere with any security mechanism on the host system. It does not establish network connections beyond downloading the standard public benchmark datasets and pretrained models, and it neither collects nor transmits any user or system data. All computation is confined to the project directory and standard framework caches.

All experiments use only publicly available, non-sensitive benchmark datasets (MNIST, CIFAR-10, GTSRB, and ImageNet), and no personally identifying or private information is involved at any stage.

The artifact implements a backdoor attack and therefore has inherent dual-use characteristics. However, it produces only backdoored model checkpoints and an emulated hardware-Trojan inference pipeline within an isolated research environment; it does not generate malware, exploit any external system, or target deployed hardware. Consistent with the Ethical Considerations in our paper, the artifact is released solely to enable defensive research into the hardware–software attack surface, and we explicitly discourage any malicious use.

A.2.2 How to access

The artifact is available at https://doi.org/10.5281/zenodo.17980434. The repository contains the attack implementations and entry points, the six defense evaluation harnesses under `defenses/` (each with its own `README.md`), the RTL under `rtl/`. Datasets and the ImageNet-pretrained backbones are downloaded automatically as described in the Benchmarks section (ImageNet itself must be obtained separately). We will provide stable updated reference for the camera-ready version once all the reviewer's feedback has been addressed.

A.2.3 Hardware dependencies

Reproducing our major claims (C1–C4) does *not* require any specialized hardware accelerator. The hardware Trojan logic (trigger detection via sign-/exponent-bit comparison and payload bias injection into the output logits) is functionally emulated within the software inference pipeline, so no physical FPGA or ASIC is needed to obtain the reported attack-success and clean-accuracy results.

The model-level experiments require a single machine with one CUDA-capable NVIDIA GPU. The MNIST (LeNet), CIFAR-10, and GTSRB (VGG-16/ResNet-18) experiments are lightweight and run comfortably on a GPU with at least 8 GB of memory; a CPU-only run is possible for these but considerably slower. The ImageNet experiments are the most

*These authors contributed equally to this work.

demanding and we recommend a GPU with at least 16–24 GB of memory for these.

Our experiments were conducted on a server equipped with four NVIDIA L40S GPUs (48 GB each; only one is required to reproduce our results), 128 logical CPU cores, and approximately 512 GB of system RAM. No specific microarchitecture, interconnect, or hardware performance counters are required.

A.2.4 Software dependencies

The artifact runs on a Linux system (tested on Ubuntu) and requires *Python 3.9.25*. All Python dependencies, including PyTorch with CUDA support, are listed in *requirements.txt* and can be installed into an isolated virtual environment as described in the Installation section. A working NVIDIA CUDA driver compatible with the installed PyTorch build is required for GPU execution. Reproducing the model-level experiments (claims C1–C4) requires only the above open-source software.

A.2.5 Benchmarks

We evaluate on four standard, publicly available image-classification benchmarks: MNIST, CIFAR-10, GTSRB, and ImageNet (ILSVRC-2012). MNIST, CIFAR-10, and GTSRB are downloaded automatically into the directory specified by `-dataset_dir` on first use (the directory is created if it does not exist). ImageNet must be obtained manually from its official source (https://www.image-net.org/challenges/LSVRC/2012/2012-downloads.php) and placed in the expected directory, as it requires registration and is not auto-downloaded.

No backdoored or task-specific model checkpoints need to be supplied. The convolutional backbones (VGG-16 and ResNet-18) are initialized from torchvision's standard ImageNet-pretrained weights, which are downloaded automatically on first use; LeNet (for MNIST) is trained from scratch. The artifact then trains/fine-tunes the clean models locally during the trigger-optimization stage, and the resulting checkpoints are reused by the weight-optimization and multi-neuron attacks.

A.3 Set-up

A.3.1 Installation

Our experiments were conducted using *Python 3.9.25*. We recommend creating an isolated virtual environment before installing the dependencies. The required packages are listed in *requirements.txt* and can be installed with the following commands:

```
python3 -m venv .env
source .env/bin/activate
python3 -m pip install -r requirements.txt
```

A.3.2 Basic Test

The following one-epoch run exercises the full software path—dataset download, model construction, GPU training, backdoor injection, the emulated trigger pathway, and checkpoint serialization—in a few minutes on a single GPU. It is a scaled-down (`-epochs 1`) invocation of the trigger-optimization attack and writes its checkpoints to a throwaway directory.

```
python3 main.py \
    --dataset_dir ./data/ --dataset cifar10 \
    --epochs 1 --model resnet --device cuda:0 \
    --target_label 0 --inject 1 \
    --train_model 1 --batch_size 256 \
    --model_path ./checkpoints_smoketest \
    --dump_model 1 --lam 0.1 --threshold 0.0 \
    --use_normalization 1 --seed 1
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): HAMLOCK achieves a near-100% attack success rate (ASR) while incurring only a minimal clean-accuracy drop (at most $\sim$2.6%). This is demonstrated by experiment (E1) described in Section 5.1, with results reported in Table 1.

(C2): HAMLOCK evades state-of-the-art white-box backdoor-sample detection defenses (IBD-PSC and TED). This is demonstrated by experiment (E2) described in Section 5.2 (subsection *Backdoor sample detection*), with results reported in Table 3.

(C3): HAMLOCK evades black-box backdoor-sample detection defenses (STRIP and BBCAL), both before deployment (software-only model) and after deployment on the Trojaned hardware. This is demonstrated by experiment (E3) described in Section 5.3, with results reported in Table 4 and Table 5.

(C4): HAMLOCK remains effective against backdoor-mitigation defenses based on lightweight retraining (fine-tuning and fine-pruning). This is demonstrated by experiment (E4) described in Section 5.2 (subsection *Effectiveness under lightweight retraining*), with results reported in Table 6.

A.4.2 Experiments

(E1): *Attack effectiveness without defenses [30 human-minutes + 4–6 compute-hours (CIFAR-10 and GTSRB) + $\sim$10 GB disk]*: This experiment supports claim C1. It runs the three attack variants of HAMLOCK—the single-neuron trigger-optimization attack (`1N-trigger`), the single-neuron weight-optimization attack (`1N-weight`), and the three-neuron multi-neuron attack (`3N-weight`)—on a given dataset/architecture and reports, for the resulting backdoored model, the clean accuracy and the attack

success rate (ASR) both *without* the Trojaned hardware (software-only model) and *with* the emulated hardware Trojan active. The expected outcome is that the software-only model is functionally indistinguishable from a clean model (clean-accuracy drop of at most $\sim$2.6% and an ASR of effectively 0, i.e. at most $\sim$0.6%), while the same model deployed on the emulated Trojaned hardware reaches an ASR of $\sim$100% (slightly lower for the `3N-weight` variant). These values populate Table 1.

How to: The three attacks are run, *in this order*, using the prebuilt scripts in `scripts/` and documented in the repository `README.md` (section "Steps to run the attack"): `run_trigger_optimization_attack.sh` (`1N-trigger`), `run_weight_optimization_attack.sh` (`1N-weight`), and `run_3N_attack.sh` (`3N-weight`). The order is mandatory: the trigger-optimization step trains and saves the clean checkpoint (under `checkpoints/clean_models_1/`) that the other two attacks load. Select the (dataset, architecture) cell by editing the `dataset` and `model` variables (and `-device`) at the top of each script; the full argument list is described in the `README.md` "Arguments" table.

Preparation: Activate the virtual environment from the Set-up section. Datasets are handled as described under Benchmarks (MNIST, CIFAR-10, GTSRB auto-download; ImageNet placed manually). No pretrained checkpoints are required—they are produced by the first script. We recommend reproducing the CIFAR-10 and GTSRB rows (lightweight, single GPU).

Execution: From the repository root:

```
bash ./scripts/run_trigger_optimization_attack.sh
bash ./scripts/run_weight_optimization_attack.sh
bash ./scripts/run_3N_attack.sh
```

Repeat (after editing `dataset`/`model`) for each Table 1 row to reproduce.

Results: Each script prints the two Table 1 metrics to standard output. The mapping is the only non-obvious part:

- **Clean accuracy (CA), w/o hardware:** `Accuracy AFTER injection` (trigger/weight scripts) or `Clean acc after` (3N). The clean-accuracy *drop* is relative to `Accuracy BEFORE injection` (for 3N, relative to the clean checkpoint's accuracy from the first script).
- **ASR with hardware Trojan active ($\sim$100%):** `ASR` (trigger script) or `ASR on triggered images` (weight/3N) — the fraction of *triggered* inputs misclassified to the target label once the emulated payload fires (the monitored neuron exceeds the trigger threshold). This is the ASR reported in Table 1.
- **Clean-input false-trigger rate (software-only, $\leq$0.6%):** printed as `ASR on clean images` (weight/3N) — despite that label it is *not* an ASR but a false-positive rate: the fraction of *clean* inputs on which the monitored neuron spuriously exceeds the threshold. A near-zero value confirms the dormant software-only model does not fire the backdoor on benign inputs.

Expect, for every cell, with-hardware ASR $\approx$100% (somewhat lower for `3N-weight`), without-hardware ASR near 0, and a small clean-accuracy drop (within the per-dataset bounds above). This validates claim C1.

(E2): *Evading white-box backdoor-sample detection (IBD-PSC and TED) [30 human-minutes + 2–4 compute-hours + negligible extra disk (reuses E1 checkpoints)]*: This experiment supports claim C2. It runs the two white-box, input-level detectors—IBD-PSC and TED—against the backdoored *software-only* models from E1 (these detectors require access to model internals, so they are applied to the dormant pre-deployment model). The expected outcome is that neither detector can separate backdoored from clean inputs, yielding an AUROC of $\approx$0.5 (no better than random). These results populate Table 3.

How to: Both defenses reuse the checkpoints produced in E1 (no retraining). The detectors and their exact commands are documented in `defenses/IBD-PSC/README.md` and `defenses/TED/README.md`, and are driven by the prebuilt sweep scripts in each directory. Each run prints an AUROC that is compared against Table 3.

Preparation: Activate the environment from the Set-up section and ensure E1 has been run so the backdoored checkpoints exist under `checkpoints/` (see the per-defense `README.md` for the exact path layout). IBD-PSC is evaluated on the single-neuron attacks (`hamock`, `hamock_weights`); TED additionally covers the multi-neuron attack (`hamock_sep`). We recommend the CIFAR-10 and GTSRB configurations (ResNet-18 / VGG-16).

Execution: From the repository root:

```
bash defenses/IBD-PSC/run.sh
bash defenses/TED/run.sh
```

Each script writes one log per (attack, dataset, model) into the respective `logs/` directory.

Results: Read the AUROC from each run's output (IBD-PSC prints a summary line plus explicit `AUROC:`/`TPR:`/`FPR:` lines; TED prints `AUC:` with TP/FP/TN/FN). For every configuration the AUROC should be $\approx$0.5 (observed range $\sim$0.43–0.52), with low TPR. TED's `Accuracy on bd_loader` is also low, confirming the software model does not act on the trigger. This shows HAMLOCK's backdoored inputs are indistinguishable from clean inputs to both white-box detectors, validating claim C2.

(E3): *Evading black-box backdoor-sample detection (STRIP and BBCaL) [30 human-minutes + 2–4 compute-hours + negligible extra disk (reuses E1 checkpoints)]*: This experiment supports claim C3. It runs the two black-box,

input-level detectors—STRIP and BBCaL—against the backdoored models from E1, in both the *software-only* scenario (the dormant model, before hardware deployment) and the *hardware-deployed* scenario (with the Trojan functionally emulated in the inference pipeline). The expected outcome is that both detectors fail to separate backdoored from clean inputs in both scenarios, yielding an AUROC of ≈0.5 (no better than random). These results populate Table 4 (MNIST) and Table 5 (GTSRB and CIFAR-10).

How to: Both defenses reuse the checkpoints produced in E1 (no retraining). The detectors and their exact commands are documented in `defenses/strip/README.md` and `defenses/bbcal/README.md`, and are driven by the prebuilt sweep scripts in each directory. Each run prints a summary line whose AUROC field is the metric compared against Table 4/5.

Preparation: Activate the environment from the Set-up section and ensure E1 has been run so the backdoored checkpoints for `<attack>` ∈ {`hamock`, `hamock_weights`, `hamock_sep`} exist under `checkpoints/` (see the per-defense `README.md`). As in the READMEs, we recommend the CIFAR-10 and GTSRB configurations (ResNet-18 / VGG-16).

Execution: From the repository root, run each defense's script; each one covers all scenarios (software-only, hardware single-neuron, and hardware multi-neuron):

```
bash defenses/strip/run.sh
bash defenses/bbcal/run.sh
```

Each script writes one log per (attack, dataset, model) into the respective `logs/` directory.

Results: Read the AUROC from each run's final summary line (`tn fp fn tp f1 precision recall AUROC clean_acc` for STRIP; `... AUROC` plus explicit `AUROC:`/`TPR:`/`FPR:` lines for BBCaL). For every configuration the AUROC should be ≈ 0.55 (observed range ∼0.45–0.65), with recall/TPR either low or—for BBCaL—offset by an equally high FPR. This confirms that HAMLOCK's backdoored inputs are statistically indistinguishable from clean inputs to both detectors, before and after hardware deployment, validating claim C3.

(E4): *Robustness to lightweight backdoor mitigation (fine-tuning, fine-pruning, and CLP) [45 human-minutes + 4–8 compute-hours + negligible extra disk (reuses E1 checkpoints)]*: This experiment supports claim C4. It applies three model-hardening mitigations to the backdoored models from E1—fine-tuning (FT), fine-pruning (FP), and Channel Lipschitzness-based Pruning (CLP)—and measures the attack success rate after mitigation. The expected outcome is that HAMLOCK *survives* all three: the trigger neuron and the hardware payload are decoupled from the normal forward pass, so retraining/pruning on clean data leaves the trigger pathway intact. The ASR therefore stays high (≈100% for the single-neuron attacks, ≈90–98% for the multi-neuron attack) while clean accuracy is maintained. These results populate Table 6.

How to: All three mitigations reuse the E1 checkpoints (no attack re-run). The commands are documented in `defenses/finetuning_finepruning/README.md` (FT and FP) and `defenses/CLP/README.md` (CLP), and are driven by the prebuilt scripts in each directory. Each run prints the post-mitigation clean accuracy and ASR, transcribed into Table 6.

Preparation: Activate the environment from the Set-up section and ensure E1 has produced the backdoored checkpoints for `<attack>` ∈ {`hamock`, `hamock_weights`, `hamock_sep`} under `checkpoints/` (see the per-defense `README.md`). We recommend the CIFAR-10 and GTSRB configurations (ResNet-18 / VGG-16).

Execution: From the repository root. Fine-tuning and fine-pruning (single-neuron via `expt_defense.py`, multi-neuron via `expt_defense_sep.py`; select the method with `-exp finetuning` or `-exp finepruning`):

```
bash defenses/finetuning_finepruning/run.sh
```

CLP (single-neuron and multi-neuron in one script):

```
bash defenses/CLP/run.sh
```

Each script writes one log per (attack, dataset, model) into the respective `logs/` directory.

Results: For every run, read the post-mitigation clean accuracy and ASR from the log (FT/FP print `Accuracy: ..., ASR: ...`; CLP prints `Test clean accuracy` together with `Test attack success rate`, before and after pruning). The ASR should remain ≈100% for the single-neuron attacks and ≈90–98% for the multi-neuron attack, essentially unchanged from the pre-mitigation value, with clean accuracy preserved. This confirms that lightweight retraining and pruning do not remove the backdoor, validating claim C4.

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/.