HotWire — AEC Reviewer Guide

WOOT '26 Artifact Evaluation

Kuan Yu Chen, Wen Wei Li, Shi Cho Cha (National Taiwan University of Science and Technology)
Md Hasan Shahriar, Wenjing Lou (Virginia Tech)

2026-05-02

This is a one-page guide for the WOOT '26 Artifact Evaluation Committee.

Artifact location

- **Public source repository:** https://github.com/sickcell6000/HotWire
- **License:** GPL-3.0
- **Zenodo DOI:** 10.5281/zenodo.19986377 (snapshot of the `woot26-artifact-rc1` tag; a separate post-AE version will be archived for the camera-ready paper per the WOOT '26 CFA two-stage process).

Badges requested

We request evaluation for two badges:

- **Artifact Available.** *The artifact is publicly available.* Public GitHub repository under GPL-3.0, archived under a release tag, and mirrored to Zenodo DOI 10.5281/zenodo.19986377.
- **Artifact Functional.** *The core functionality of the artifact can be confirmed by the AEC.* One command — `bash verify_artifact.sh` — runs eight checks (F0–F6) in approximately five minutes on a clean Linux / macOS / Windows machine and exits with `8/8 ✓ ALL PASSED`.

The paper's §6 field-test results were collected on 4 production EVs and 7 commercial DC fast-charging stations and rely on physical access to that fleet. The frozen capture bundles in `datasets/real_hw_traces/` preserve the wire-level evidence from those field tests, but reviewers are not expected to re-run them.

What you need

- A Linux, macOS, or Windows machine (any of those work).
- Python **3.11** or newer.
- ~500 MB free disk, ~5 minutes for the headline check.
- **No hardware required.** No internet required after `pip install`.
- **Docker is *optional*.** `verify_artifact.sh` F1 prefers Docker CI if a daemon is running (278 tests in-container) and transparently falls back to host pytest if not (179 tests; 99 Docker-only ones auto-skip). Either way F1 = PASS and the run reaches `8 / 8 ✓ ALL PASSED`.

Three verification paths

You only need to run **one** of these for the Functional badge. Do all three if you want to also see the attacks live and inspect the real-hardware traces.

Path 1 — One-shot functional check (5 minutes)

Step 0 (all platforms): download the artifact zip from Zenodo The "Available" snapshot for this AE submission lives at the Zenodo record below — please download the zip from there rather than `git clone`-ing the repo, so reviewers and authors look at exactly the same bytes:

> **Download:** https://zenodo.org/records/19986377 (DOI 10.5281/zenodo.19986377)

The download is a single file, `HotWire-woot26-artifact-rc1.zip` (~5 MB). Save it somewhere convenient (e.g. `~/Downloads/`), then follow the platform-specific block below — each starts with `unzip` + `cd` into the unpacked directory.

Linux / Raspberry Pi

```
# One-time system prerequisites (Pi/aarch64 has no PyQt5 wheel,
# so we use the apt-packaged Qt5 binary and let venv see it):
sudo apt update
sudo apt install -y python3-venv python3-dev python3-pip python3-pyqt5 python3-pyqt5.qtsvg unzip

unzip HotWire-woot26-artifact-rc1.zip
cd HotWire-woot26-artifact-rc1
python3 -m venv --system-site-packages hotwire-venv
source hotwire-venv/bin/activate
pip install -r requirements.txt
bash verify_artifact.sh
```

macOS

```
unzip HotWire-woot26-artifact-rc1.zip
cd HotWire-woot26-artifact-rc1
python3 -m venv hotwire-venv
source hotwire-venv/bin/activate
pip install -r requirements.txt
bash verify_artifact.sh
```

Windows `verify_artifact.sh` is bash-only by design (matches the convention used by every WOOT 2025 accepted artifact). Confirm you have a bash environment first — Start menu → search `Git Bash`:

- **Git Bash present** (Git for Windows installed): open Git Bash and follow the *Git Bash* block below.
- **Only WSL**: WSL is Linux — follow the **Linux / Raspberry Pi** block above, run inside `wsl`.
- **Neither**: install Git for Windows (~50 MB; bundles Git Bash). Or install WSL.

Windows — Git Bash / MSYS2

```
# Native Windows Python; venv layout is `Scripts/`, not `bin/`.
# (Git Bash includes `unzip`. Alternatively right-click the zip in
# Explorer → "Extract All…" and skip the unzip line.)
unzip HotWire-woot26-artifact-rc1.zip
cd HotWire-woot26-artifact-rc1
python -m venv hotwire-venv
source hotwire-venv/Scripts/activate
pip install -r requirements.txt
bash verify_artifact.sh
```

PowerShell users: swap the activate line for `.\hotwire-venv\Scripts\Activate.ps1`. `verify_artifact.sh` itself still needs Git Bash or WSL because the script is bash-only.

For real-hardware (PLC modem) work, additionally install `libpcap-dev` and `pip install -r requirements-hw.txt`.

Expected last line:

```
8/8 ✓ ALL PASSED
```

The script runs eight checks (F0 through F6) covering:

- **F0** Package import and Python compatibility.
- **F1** 278 unit + integration tests via `pytest` (8 skipped on hosts without real hardware).
- **F2** Headless V2G smoke session (sim mode).
- **F3** Parametric matrix: 3 voltages × 3 durations = 9 successful sim-mode sessions.
- **F4** A1 attack script (EVCCID impersonation).
- **F4** A2 attack script (forced-discharge / PreCharge voltage forgery).
- **F5** GUI test suite (PyQt5 widget tree, signal wiring).
- **F6** Frozen-evidence integrity (SHA-256 verification of the 11 curated real-hardware test bundles in `datasets/real_hw_traces/`).

If any check fails, the script prints the failing command and exits with a non-zero code — please paste that output into an AEC issue.

Path 2 — Interactive GUI demo (10 minutes)

In two terminals on the same machine:

```
# Terminal 1 — rogue charging station
python scripts/run_gui.py --mode evse --sim

# Terminal 2 — rogue vehicle
python scripts/run_gui.py --mode pev --sim
```

Both processes loop back over a virtual PLC link (no Ethernet required). The GUI shows every DIN 70121 message in a four-tab tree (Combined / Rx / Tx / Trace log) and lets you pause-and-override fields before they go on the wire.

To reproduce the two attacks:

1. Open `config/attack_presets.json` to see the 16 bundled presets.

2. In the EVSE GUI, click **Launch attack → A1** (EVCCID impersonation) or **A2** (forced-discharge). The PEV side will honor the forged values and the trace log will show the abuse path the paper describes.

Path 3 — Inspect frozen real-hardware evidence (15 minutes)

```
datasets/real_hw_traces/comprehensive_matrix/
├── test1_a1_evccid/          ← A1 attack on real charging stack
├── test2_a2_voltage/         ← A2 attack on real charging stack
├── test3_multi_pause/        ← multi-pause field manipulation
├── test4_abort/ … test11_long_running/
├── FINDINGS.md               ← what each bundle proves
└── README.md                 ← per-bundle rationale
```

Each bundle has:

- `*.pcap` — raw HomePlug AV / V2GTP capture (open in Wireshark with the HomePlug AV dissector).
- `session.jsonl` — one decoded DIN 70121 message per line, including a `values:` field with the actual EVCCID, voltages, SoC, etc. **No EXI decoder is needed** — the recorded fields are plain JSON.
- `config.json` — exact run parameters (mode, target voltage, attack preset).

The mapping from each bundle to a specific paper claim is in `docs/PAPER_VALIDATION.md` (§ "Suite 0 through Suite 6").

What maps to what in the paper

Paper claim	Where to verify
§3 DIN 70121 / ISO 15118-2 FSM	F1 + F2 + F3 / `hotwire/fsm/fsm_evse.py,fsm_pev.py`
§4 A1 EVCCID-impersonation attack	F4 / GUI demo / `test1_a1_evccid/`
§4 A2 forced-discharge attack	F4 / GUI demo / `test2_a2_voltage/`
§5 Tooling (FSM, GUI, pause/override)	F0–F2 / GUI demo
§6 Field results on real fleet	Frozen captures only — physical fleet not in scope

Known-good environments

We have run `verify_artifact.sh` to **8/8 PASS** on all of:

- Raspberry Pi OS Bookworm (aarch64) + Python 3.11 + Docker.
- Ubuntu 24.04 LTS (amd64) + Python 3.12 (host pytest fallback).
- Ubuntu 26 "resolute" (amd64) + Python 3.14 (host pytest fallback).
- Windows 11 22H2 + Python 3.12 + Git Bash (MINGW64) + Docker Desktop.