Demo: Open CAN Bus Interfaces for BeamNG.tech: Enabling Affordable HIL and Software-in-the-Loop Automotive Security Research

Bhargab Acharya
University of Memphis

Christos Papadopoulos
University of Memphis

Spiros Thanasoulas
University of Memphis

Abstract

BeamNG.tech is a high-fidelity soft body vehicle simulator widely used in automotive research. We present three open BeamNG transport extensions: `CANBusSerial.lua` (slcan USB-CAN adapters), `CANBusUDP.lua` (UDP bridge between machines), and `CANBusVirtual.lua` (direct SocketCAN binding); plus an input consumer `CANBusInput.lua`, that expose BeamNG's vehicle CAN traffic to vcan, SocketCAN tools, real ECUs, or a physical instrument cluster, at a fraction of the hardware cost (roughly $25 for a CANable adapter versus ~$300 for a PEAK PCAN-USB, or zero with vcan). As a demonstration we drive a BeamNG vehicle with Comma.ai's Openpilot through this CAN layer, and will show a hardware-in-the-loop demo with a physical instrument cluster at the conference. The Lua modules will be released on GitHub and as a BeamNG mod. [1]

1 Introduction

Automotive security and ADAS research benefits from high-fidelity simulators that expose the same interfaces as real vehicles, in particular the CAN bus. BeamNG.tech [2] provides excellent vehicle physics and a rich Python API, and supports hardware-in-the-loop CAN, but only through PEAK System adapters. PEAK adapters are reliable but expensive and not what most students, hobbyists, or budget-constrained labs already own. Cheap USB-CAN adapters that implement the slcan protocol are widely available for a small fraction of the price, and Linux's built-in virtual CAN (`vcan`) costs nothing.

In our prior work [1] we built a direct integration between BeamNG and Openpilot [4] for telematics generation. That system used ad-hoc Python interfaces that did not resemble a real in-vehicle network, limiting its value for security research where the CAN bus is the primary attack surface.

This demo presents four open Lua modules that solve the underlying problem: how to get BeamNG's CAN traffic out to whatever hardware or software a researcher actually has, and how to let an external controller drive the simulator back through the same bus.

[1] https://github.com/bhrg3se/beamng_canbus

2 The Lua Modules

The toolkit consists of three transports (`CANBusSerial.lua`, `CANBusUDP.lua`, `CANBusVirtual.lua`) and one input consumer (`CANBusInput.lua`). All three transports are drop-in replacements for BeamNG's existing `CANBusPeak.lua`: they mirror its public API and error-code namespace exactly, so any vehicle Lua written against the PEAK extension works unchanged after switching the require; this includes `CANBusInput.lua` and BeamNG's stock `RacingDisplay.lua` instrument-cluster output. Researchers can develop against the UDP/vcan path with no hardware, then move to a real adapter and a real ECU without changing simulator-side code.

`CANBusSerial.lua`. Drives a serial-attached CAN adapter using the slcan ASCII protocol. Any USB-CAN dongle that speaks slcan (CANable being the most common) works without modification. This is the cheap-HIL path: plug in the adapter, wire it onto a real CAN bus, and BeamNG appears as another node alongside any physical ECUs.

`CANBusUDP.lua`. Encapsulates CAN frames in UDP packets and ships them to a configured endpoint. On the receiving end, a small bridge unpacks frames into a Linux `vcan` interface, so SocketCAN tools, Wireshark with the CAN dissector, intrusion detection systems, and full autonomous stacks see the bus as if it were physical. The receiver can be a separate computer, a Linux VM, or WSL on the same Windows host. This lets us run BeamNG on Windows and Openpilot in WSL while sharing a single CAN view.

`CANBusVirtual.lua`. Binds directly to a Linux SocketCAN interface (`vcan0` for pure-software setups or any `canX` device for real hardware) via raw CAN sockets. Zero-cost, zero-network-overhead path when BeamNG runs on Linux. Also the natural target on the receiving side of `CANBusUDP.lua`.

`CANBusInput.lua`. Listens for inbound CAN frames carrying control commands (throttle, brake, steering, and similar actuator signals) and applies them to the BeamNG vehicle. Transport-agnostic: sits on top of any of the three transports. This closes the loop when an external controller (Openpilot, a custom autonomy stack, or an attacker tool) needs to drive the simulated vehicle rather than just observe it.

3 Application: Driving Openpilot Through the Bus

We replaced the direct Python bridge from our prior framework with a CAN-mediated link. Vehicle telematics flow from BeamNG over CAN to Openpilot, and Openpilot's control outputs flow back over the same bus. We reuse Openpilot's existing simulator harness (originally written for MetaDrive) as the integration point, swapping in our BeamNG CAN link.

The current message set is custom and synthetic, covering steering angle, wheel speeds, throttle/brake position, and a subset of BeamNG's telematics.

4 Demonstration

The demo will show BeamNG integrated with OpenPilot, with the latter driving a car autonomously inside the simulator. We will show a single demo that combines software-in-the-loop and hardware-in-the-loop. BeamNG will run on a laptop; `CANBusUDP.lua` will carry vehicle CAN traffic into a vcan interface where Openpilot drives. In parallel, `CANBusSerial.lua` will stream the same state through an inexpensive CANable slcan adapter onto a real CAN bus wired to a real physical automotive instrument cluster. The cluster's dials, indicators, and warning lights will respond in real time to Openpilot's actions in BeamNG.

Attendees will watch live CAN traffic on a decoder, and see the same simulated vehicle simultaneously controlled by an autonomous stack and reflected on real automotive hardware. We will also demonstrate a basic proof-of-concept attack (a CAN injection or replay against a control or instrument signal) to show the kind of security experiments the toolkit enables.

5 Why This Matters for Security Research

Putting BeamNG on a real or virtual CAN bus exposes the same attack primitives a real vehicle does (injection, replay, fuzzing, flooding), and detection tools developed here port directly to physical vehicles. It also makes it safe to run attacks that would be dangerous or destructive on a real car: spoofing wheel-speed sensors to confuse ABS/ESC, injecting steering commands on a curve, replaying brake-release frames during emergency braking, or flooding the bus to induce ECU dropout. Entry cost drops from the expensive PEAK adapter (one per workstation) to either zero (vcan) or the price of an slcan dongle.

6 Limitations and Future Work

The toolkit does not achieve a guaranteed real-time behavior. CAN is a hard real-time bus, and our transports introduce jitter we have not quantified: the UDP path adds host networking latency, the slcan path is bounded by USB-serial throughput and adapter firmware, and frame emission is currently driven from BeamNG's vehicle Lua tick rather than a dedicated timer. For the demo applications, end-to-end behavior is visibly correct, but we do not yet claim suitability for experiments depending on tight cycle-time guarantees (e.g., ABS/ESC arbitration timing).

A natural extension is generalized DBC support so the toolkit works with any vehicle's CAN definitions by swapping the DBC file. The obstacle is that DBC signal names are not standardized across OEMs. We plan to address this via COVESA's Vehicle Signal Specification (VSS) [5] as a vendor-neutral intermediate layer: BeamNG signals emitted under stable VSS paths, with per-vehicle VSS-to-DBC mappings drawn where possible from opendbc [3].

We will release the Lua modules and UDP bridge on GitHub and as a BeamNG mod. Telematics datasets collected will be published on PIVOT project [6].

7 Conclusion

BeamNG ships with a PEAK-only CAN example, leaving researchers without PEAK hardware to roll their own transport. The modules presented here fill that gap by opening the simulator's CAN layer to any slcan adapter, any UDP-reachable consumer, or any SocketCAN interface. The Openpilot integration and instrument-cluster HIL demo show what becomes possible once the bus is open: cheap, realistic, and safe automotive security testbeds.

References

[1] ACHARYA, B., PAPADOPOULOS, C., AND THANASOULAS, S. Demo: Telematics data collection from BeamNG with Openpilot integration. In *Proceedings of the 3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec '25)* (2025), USENIX Association.

[2] BEAMNG GMBH. Beamng.tech: Vehicle simulation for research. https://beamng.tech, 2023.

[3] COMMA.AI. opendbc. https://github.com/commaai/opendbc. Accessed: 2026-05-05.

[4] COMMA.AI. openpilot: Open source driver assistance system. https://github.com/commaai/openpilot, 2024.

[5] COVESA. Vehicle signal specification. https://github.com/COVESA/vehicle_signal_specification. Accessed: 2026-05-05.

[6] PIVOT PROJECT. Platform for innovative use of vehicle open telematics (pivot) project. NSF-funded Research Project, 2025. Available at: https://pivot-auto.org/.