

Towards a Cyber Assurance Testbed for Heavy Vehicle Electronic Controls

Jeremy Daily, Rose Gamble, Stephen Moffitt, Connor Raines, Paul Harris, and Jannah Miran
University of Tulsa

Indrakshi Ray, Subhojeet Mukherjee, and Hossein Shirazi
Colorado State University

James Johnson
Synercon Technologies

ABSTRACT

Cyber assurance of heavy trucks is a major concern with new designs as well as with supporting legacy systems. Many cyber security experts and analysts are used to working with traditional information technology (IT) networks and are familiar with a set of technologies that may not be directly useful in the commercial vehicle sector. To help connect security researchers to heavy trucks, a remotely accessible testbed has been prototyped for experimentation with security methodologies and techniques to evaluate and improve on existing technologies, as well as developing domain-specific technologies. The testbed relies on embedded Linux-based node controllers that can simulate the sensor inputs to various heavy vehicle electronic control units (ECUs). The node controller also monitors and affects the flow of network information between the ECUs and the vehicle communications backbone. For example, a node controller acts as a clone that generates analog wheel speed sensor data while at the same time monitors or controls the network traffic on the J1939 and J1708 networks. The architecture and functions of the node controllers are detailed. Sample interaction with the testbed is illustrated, along with a discussion of the challenges of running remote experiments. Incorporating high fidelity hardware in the testbed enables security researchers to advance the state of the art in hardening heavy vehicle ECUs against cyber-attacks. How the testbed can be used for security research is presented along with an example of its use in evaluating seed/key exchange strength and in intrusion detection systems (IDSs).

CITATION: Daily, J., Gamble, R., Moffitt, S., Raines, C. et al., "Towards a Cyber Assurance Testbed for Heavy Vehicle Electronic Controls," *SAE Int. J. Commer. Veh.* 9(2):2016, doi:10.4271/2016-01-8142.

INTRODUCTION

Heavy vehicles (i.e., trucks and buses) are part of the US critical infrastructure with the mission to carry out a significant portion of commercial and private business operations. However, little effort has been invested in cyber security for these mobile assets. Recent research efforts demonstrated security vulnerabilities in passenger vehicle controller area network (CAN) [1]. These, and related reports of CAN bus hacks, have put the heavy vehicle sector on notice to more quickly address cyber security issues. As such, a cyber-physical testbed is well suited to address the rapid developments in cyber security research.

As cyber assurance testing ensues, any identified threats should be well-founded with tested mitigation strategies in place. To provide the needed accuracy to demonstrate potential exploits and subsequent trust in mitigation strategies, a scalable high-fidelity testbed using actual heavy vehicle electronic control units (ECUs) is needed. The purpose of this paper is to detail the design and implementation of a

testbed suitable for cyber assurance testing for commercial vehicles, with primary focus on Class 8 truck-tractors typically used in long-haul applications.

The concept of a testbed for networked embedded systems is already well-accepted in the Supervisory Control and Data Acquisition (SCADA) space. For example, Sandia National Laboratories has operated a SCADA security program since 1998, part of which is a SCADA testbed that allows researchers to test vulnerabilities and new security methods [2]. Similar testbeds are operated by other US national laboratories, as well as European countries [3]. As industrial process control/SCADA computers have a similar purpose as vehicle ECUs, namely to monitor and control physical phenomena, the proven usefulness of the testbed approach in the SCADA suggests the value of an automotive security testbed.

Differences Between Commercial Vehicles and Passenger Cars

Although security attacks and countermeasures for automobiles might be valid for heavy vehicles, there are differences that create a distinctive threat landscape and necessitate dedicated security experimentation. The first difference is that heavy vehicles deploy a J1939 specified communication network, with open documentation as to packet definition and transmission. Passenger vehicles use distinct, higher-level communication protocols that may be proprietary, and thus, less open to the creation of attack vectors. For example, some cars have a proprietary CAN bus running from the Airbag Control Module to the Electronic Stability Controller that is accessible through either an end node or physically locating the wires within the car [4].

Finding vulnerabilities and attacking J1939 protocols have very severe consequences as they are used by almost all commercial heavy vehicles. The second difference is the fact that J1939 introduces a newer set of protocols that must be run above the CAN (physical) layer. Because the J1939 layers are modelled on the network OSI stack, many of these protocols have underlying resemblance with communication protocols used in computer networks. Thus, the attacks prevalent on computer networks may be adapted to vehicular networks adhering to the J1939 standards. For example, spoofing source and destination addresses used as part of the extended CAN identifier in J1939 may allow attackers to cause denial-of-service attacks like amplification and reflection.

While the J1939 network is founded on the same base CAN protocol that passenger vehicles use, heavy trucks follow a more horizontal integration to allow for customization. This customization can occur in the choice of engine, brake controller, telematics unit, stereo system, and other ECUs, which necessitates a more homogeneous network architecture than what is seen in passenger vehicles.

The third difference between heavy trucks and passenger vehicles is with respect to the universal usage of telematics and 3rd party devices. Fleet managers often install third party telematics units to connect the truck and driver to a back office server and logistics management system. Currently, there is no on-line mechanism in place for policing the J1939 network for standards compliance. Thus, it is possible for a heavy vehicle to be subverted through a third party system, and it is unclear if vehicle original equipment manufacturers (OEMs) can prevent such an attack.

Scope and Objective

The intention of this paper is to describe the design and testing of a cyber assurance testbed for discrete modules communicating on various vehicle communications networks with special emphasis on J1939. The implementation will leverage tools from IT security research that are applicable to heavy vehicles. Engineers and developers who lack the resources for creating such a testbed will be

able to use it by virtue of remote access and user management interfaces. Finally, the testbed operation will be verified by conducting a set of cyber-security related experiments.

TESTBED HARDWARE ARCHITECTURE

Design Requirements

The design of the testbed is driven by customer requirements. The customer, in this case, is a cyber security researcher who has adequate background in computer science. The customer may be familiar with computer tools like disassemblers and decompilers. They look for patterns and vulnerabilities in network communications. Many are well versed in using command line tools and Linux. Often the researcher is familiar with TCP/IP protocols. They understand basic cryptography. However, the customer does not have access to a vehicle.

Providing access to heavy vehicle electronic parts to a security researcher should minimize cost and maximize utility. Effective cyber security research would likely take an actual vehicle out of service, which is an expensive proposition. This expense has likely reduced the exposure of commercial vehicles to the security research community; they have focused mainly on cars [5, 6]. The cost of an engine control module can rival that of a high performance laptop computer, so the cost target for the testbed will have to account for the actual cost of the modules and materials, but will be much less than the cost of an actual truck.

The testbed needs to be able to interact with both the cyber and the physical aspects of heavy vehicle systems. As such, sensor simulation is needed to provide the modules in the testbed with some simulated physical environments.

Limitations

While the testbed uses actual ECUs, it is not intended to replace traditional hardware-in-the-loop (HIL) testing. The primary focus for a cybersecurity based testbed is to affect network communications. As such, less emphasis is placed on sensor simulation and determinism.

The authors took some liberties with the construction of the testbed and did not always follow the manufacturer's recommendation. Notably, many splices were made by crimping two wires into the same connector cavity. While electrical continuity is satisfied, the connections may not endure in an operational environment.

Topology

The structure of the testbed is designed to enable a researcher to log into a node and observe or affect network traffic on two ends of their controller. As shown in [Figure 1](#), the common backbone connects different ECU nodes. Each node has, at a minimum, a node controller and an ECU. Some nodes have a sensor simulator to mimic physical sensor values. Some ECUs, like a telematics unit, do not need physical sensors connected, which is why a sensor simulator is optional.

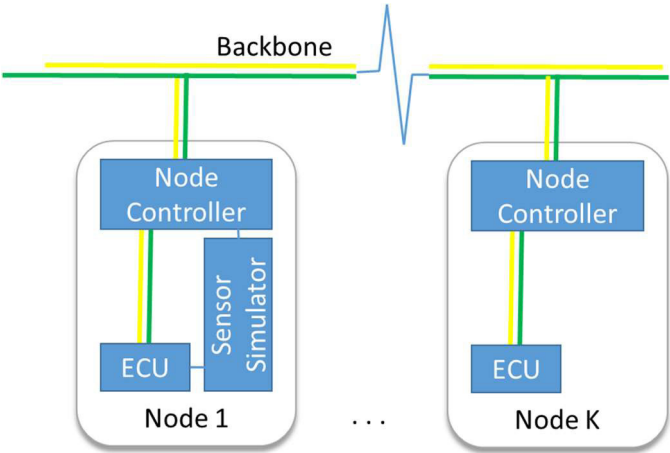


Figure 1. Testbed concept arrangement with node controllers separating each module from the backbone.

The testbed design starts with a communications backbone, which is shown in Figure 2. The backbone comprises the SAE J1939-13 9-pin connector and its signals [7]. The backbone has power, ground, J1939, J1708 and a second CAN network. Since the high current drawing functions, like firing injectors, are not part of the testbed, the backbone supplies power to each module through the J1939 connector. The backbone is modular with each section having a receptacle housing, pin housing, and a stub with a consistent connector, which is a 10-pin Molex Mini-Fit-Jr connector in this case.



Figure 2. Backbone stub that gives each node access to the communication bus through the Molex 10-pin connector.

The unique feature of this testbed is that every engine control module (ECM) has its own remotely accessible node controller. The node controller is a Beagle Bone Black connected to a communications circuit board. This hardware is connected to both the backbone and the ECM that it is controlling. The circuits in the node controller can enable the device to be a filter and separate the networks, or it can connect the communications lines directly from the backbone to the ECM. Each node controller has the Beagle Bone Black device (denoted as the Node Controller in Figure 3). Since some devices need sensors and actuators to function, an additional sensor simulator, controlled by USB from the Beagle Bone, can be used as part of the node controller.

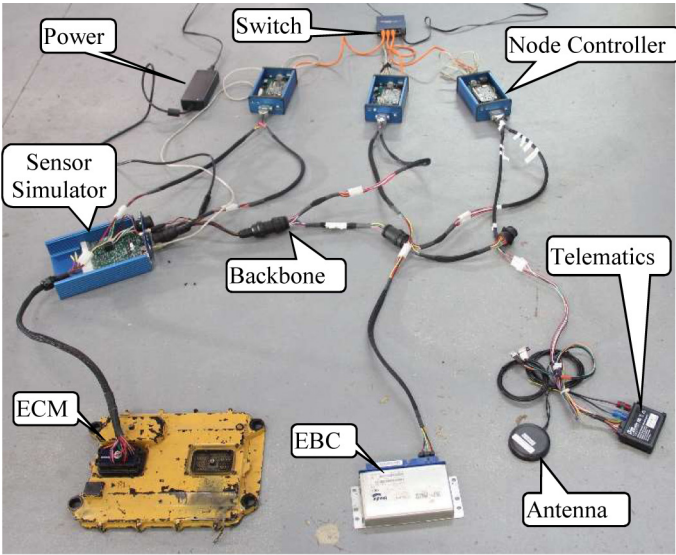


Figure 3. Physical layout of a basic testbed.

Each node controller is physically connected to an Ethernet switch, which is connected to the remote user management system. Through this network connection, remote users can log in and affect or observe network traffic to and from the backbone and the ECM in the node. They can also issue commands to affect the sensor simulation system. An additional serial debugging port from each Beagle Bone Black is made available for local access.

Table 1. Description of the items in the testbed shown in Figure 3.

Callout	Description and Notes
ECM	Engine Control Module. This is the primary control unit for the power plant.
EBC	Electronic Brake Controller. Under some circumstances, the EBC may issue torque commands to the ECM over J1939.
Telematics and Antenna	A third-party add-on connected to the J1708 and J1939 networks.
Backbone	A modular connection for power, J1939, J1708, and CAN2
Sensor Simulator	A microprocessor based system with switches, digital potentiometers, digital to analog outputs, CAN, and USB connections. This system simulates common engine signals.
Power	A 12V, 11-amp power supply is use to apply power to the backbone.
Switch	All node controllers are connected to an Ethernet switch. The switch also connects to the remote user interface and historian computer.
Node Controller	A Beagle Bone Black based system that operates between the node ECM and the backbone.

The node controller runs ARM Linux and has firewall-like capabilities. It can run code written by the security researcher. The node controller also interfaces with the sensor simulator.

USB. It can send commands and messages over the USB system to the microprocessor and peripherals that are programmed to emulate sensors and provide the ECM sensor values.

Node Controller Implementation

The node controller is built around a Beagle Bone Black single-board computer. The Beagle Bone Black sports the Texas Instruments 1GHz AM335x ARM® Cortex-A8 processor [4], which has two built-in CAN controllers, USB, and Ethernet. The operating system runs on a built-in eMMC chip with 4GB of capacity. The inexpensive computer interfaces with a custom built circuit card assembly that provides the following additional circuitry:

- Power input, protection, and regulation
- Two CAN Transceivers
- Two J1708 Transceivers
- CAN Bypass Relay
- J1708 Bypass Relay
- Serial debugging
- Ethernet breakout
- USB Host

The bypass relays are switched using GPIO pins from the Beagle Bone. When activated, the relays electrically connect the node ECU communication to the backbone, so they are always connected. When the relays are closed, the node controller cannot block network traffic. Instead, any network communications from either side will be transmitted or received on both the backbone and the node communication. In other words, the relays change the node controller from a man-in-the-middle to just another device connected to the communications bus.

Linux on the Beagle Bone

The node controller runs Ubuntu Linux with version 3.8 of the Linux kernel. The kernel contains modules for SocketCAN, which provides a socket interface to CAN hardware [8]. Kernel extensions exist for J1939 as well [9]. Command line open-source CAN utilities, such as candump and cangen, enable users to quickly set up meaningful experiments and logging functions. In fact, the default setting for the cangen command is to continuously send random messages on the network, which is effectively a simple fuzzing attack.

Of particular interest in the Beagle Bone and the AM335x processor is the availability of two on-board 200MHz programmable real time units (PRUs) [10]. These on chip peripherals are necessary to maintain the timing requirements for J1708 communications.

Linux-based J1708 Communications

Implementing J1939 [11] and ISO15765 [12] communication on a Linux computer was relatively simple, thanks to the existing driver ecosystem; all kernel-level operations had already been implemented in open-source software, and all physical and data-link layer operations for CANs are handled in hardware.

J1708, however, presents a design challenge due to the precise timing requirements. Since the hardware is abstracted in a Linux based system, the user space does not have access to low level timing functions and deterministic timing. As such, attempts to implement the J1708 communication as a program running in user space resulted in mishandling message frames frequently enough, even with a real-time kernel, to force an alternative, more deterministic, solution.

While the J1708 protocol is based on 9600 baud serial communication, and a UART can be configured in a fairly straightforward way to act as a J1708 transceiver, this only handles the physical layer of the protocol. While hardware J1708 and microprocessor code implementations exist, the goal was to reduce the amount of auxiliary processors on the printed circuit card and implement the protocol natively within the Beagle Bone Black, which has the TI AM335x processor.

As the TI AM335x System on a Chip (SoC) contains several on-board UART and GPIO modules, the chosen solution was to use those on-board modules to communicate on the J1708 bus. As messages on J1708 buses are delimited by quiet periods on the order of one millisecond in length, and message's priority is determined by the number of bit times a sender waits after "bus quiet" to send it, determinism in the time that bytes are sent and received by the controlling program is essential to a properly functioning implementation.

The solution leveraged the real-time capabilities of AM335x's Programmable Realtime Unit and Industrial Communication Subsystem (PRU-ICSS or PRU) to provide the necessary determinism. The PRU system includes two 200MHz RISC co-processors on the SoC that can communicate with the main CPU core, as well as the on-board peripherals. The PRUs are specifically designed for deterministic execution: instructions take one cycle to complete, and there is no pipelining of instructions.

Operations where determinism is critical, i.e. bus arbitration and sending & receiving of messages, are carried out by the PRU.

Communication between the PRU and userspace programs is carried out by a privileged userspace process; a functional diagram outlining the implementation is given in Figure 4. It is possible to implement this in the Linux kernel.

The algorithm used by the driver is described in the J1708 standard [13]. Briefly, upon initialization the driver waits for 10 consecutive bit times of bus quiet to indicate that any subsequent characters received begin a new message. When a message is received, it is placed in a circular buffer in shared RAM and an interrupt is triggered to alert the userspace process to read it. Likewise, when a message is to be sent, the userspace process places the message in a different circular buffer and sends an interrupt to signal the PRU process that the message is to be sent when the next bus quiet condition exists. Loading of the PRU code and communication of interrupts between the PRU and userspace are handled using the PRU Userspace IO (UIO) driver provided by the kernel.