Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

2016-01-8142
Published 09/27/2016
Copyright © 2016 SAE International
doi:10.4271/2016-01-8142
saecomveh.saejournals.org

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]. This, 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.

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

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.

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

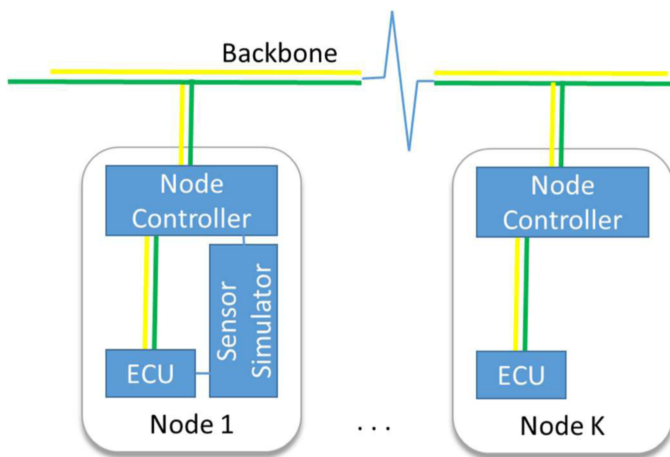


Figure 1. Testbed concept arrangement with node controllers separating each module from the backbone.

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.

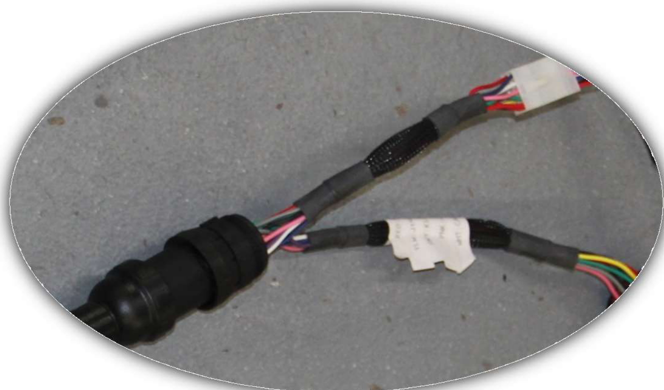


Figure 2. Backbone stub that gives each node access to the communication bus through the Molex 10-pin connector.

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.

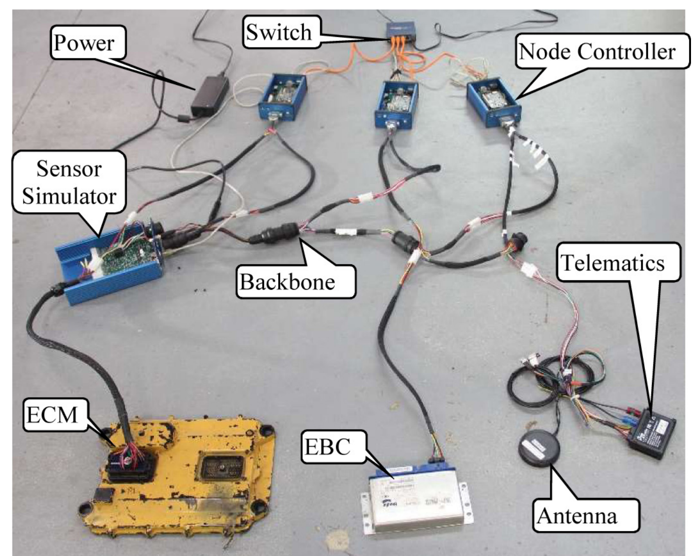


Figure 3. Physical layout of a basic testbed.

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 ECU 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 simulation system by

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

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 is handled using the PRU Userspace IO (UIO) driver provided by the kernel.

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

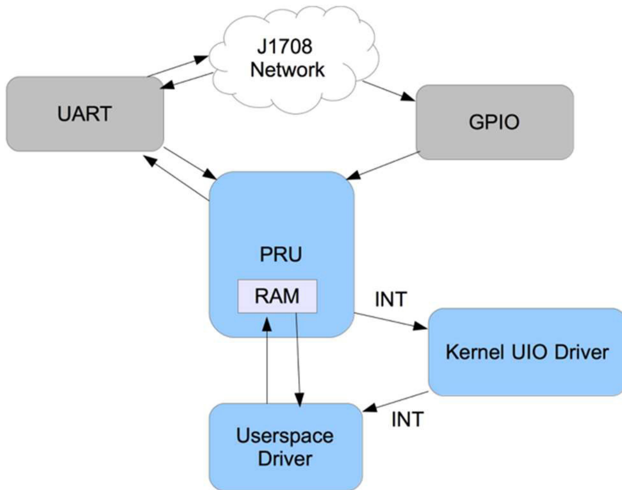


Figure 4. Functional Diagram of a J1708 implementation on the Beagle Bone Black.

The benefit of this software-based approach is that it leverages widely-available commodity hardware, making production of new nodes easier and less costly. Furthermore, the testbed node controllers have two of these drivers implemented, which enables it to act as a man-in-the-middle and examine or alter each message frame as it passes through the controller.

Sensor Simulator Implementation

Following the successes in using an Arduino based system in [14], the authors designed a universal sensor simulator module that can emulate many different sensors that an ECU is looking for. A simple potentiometer and switch network can be built that can emulate many resistive based and voltage based sensors, as shown in Figure 5.

An example implementation uses a Microchip MCP41HV51 High Voltage Digital Potentiometer [15] . This device has internal switches to function as SW2-SW5 in Figure 5. The switch at the top of Figure 5 provides a connection to the positive voltage. If a high voltage tolerant switch, like the Analog Devices ADG1401 [16], is used, then +12V can be used to generate voltages according to the taps on the digital potentiometer from 0 to +12V when SW1 is closed. If SW1, SW4, and SW5 are open, then an adjustable resistance can be simulated between Port 1 and Port 2. If SW1, SW2, and SW3 are open with SW4 and SW5 closed, then the Ports provide a resistance to ground, which can simulate temperature sensors and level sensors.

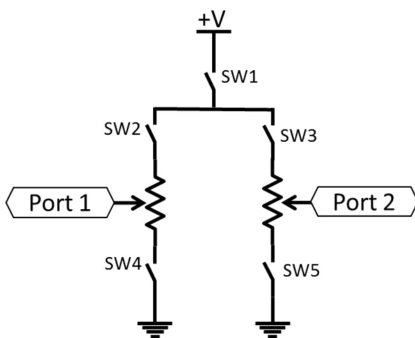


Figure 5. Potentiometer network to emulate many sensors.

Even a low-end microprocessor interfacing with the digital potentiometers and digital to analog converters (e.g. MCP4728) can receive commands over a USB to serial connection from the node controller and adjust settings. This functionality of the testbed gives the experimenter an opportunity to explore some scenarios involving cyber and physical aspects of these machines.

USER INTERACTION

One goal of the heavy vehicle testbed is to allow researchers to perform remote experiments and export the outcome data for analysis, without having to physically access a heavy vehicle. The testbed should have a full J1939 implementation so that all J1939 network messages are available.

In addition, it should allow researchers to

- Fully interact with the available ECUs
- Input experimental or historical data to determine ECU responses
- View the results rendered in real time using live-updating charts and tables that are parameterized according to the researcher's standards
- Inject their own J1939 messages onto the CAN network which may simulate attacks
- Record any traffic generated by an experiment for analysis

The remote interface should provide an intuitive platform that is accessible even to those without a strong technical background. Since the testbed is based on J1939 standard, it can be expanded to function with any other hardware or simulation devices that utilizes the same standard. The interface also allows testbed administrators control over the node controllers. Thus, the remote interface is expandable to meet future growth and expansion.

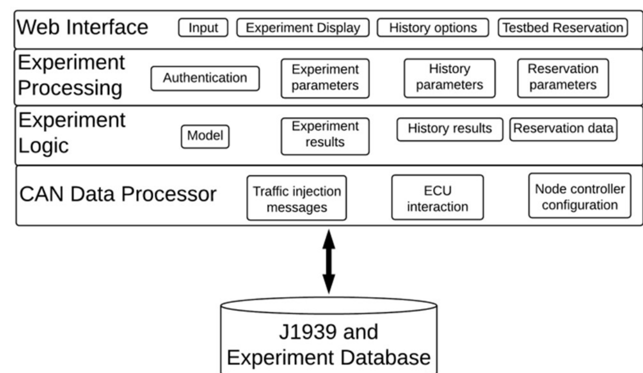


Figure 6. Software Architecture of the Remote Interface to the Testbed

The testbed architecture shown in Figure 6 consists of five layers: the Web Interface, the Experiment Processing framework, the Experiment Logic framework, the CAN Data Processor, and the Database. The Web Interface frontend allows a client to login via a web browser where they are given options to create experiments, view their history, and make reservations. When creating an experiment, a user can select experiment parameters and the relevant display for the results. When the experiment is started, the setup parameters are passed to the Experiment Processing layer, which

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

periodically retrieves newly generated data and renders the results for display by the Web Interface. Experiment Processing allows novice users to understand CAN traffic by converting its messages into a human readable format with units and text descriptions. At the same time, advanced users have the ability to download the actual CAN messages for more in-depth analysis. Experiment Processing is also used for access to experiment history and the reservation system to access the testbed.

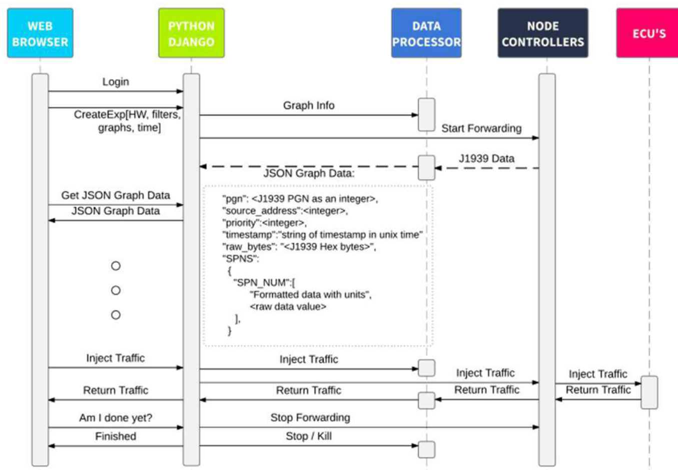


Figure 7. Sequence Diagram Illustrating the Processing Performed by the Remote Interface

The CAN Data Processor layer forms the backend. With the node controllers bridging the ECUs to the CAN network, the network can be logically segmented or combined in any arrangement desired by the user. The node controller allows a user to selectively forward and inject messages either into the backbone or the truck components on the testbed. The CAN Data Processor framework saves the traffic in the experiment database. This experiment data can be downloaded by the researcher at any time for offline use. To package the J1939 traffic for use by both the database and the web browser, the setup distributes the processing across the node controllers to increase concurrency and prevent overwhelming the server. As the Node Controllers send messages to the backend, it aggregates the messages and waits until the frontend calls for an update. Lastly, any injected messages are handled by the CAN Data Processor just like the rest of the J1939 messages.

The sequence diagram in Figure 7 depicts the flow of functionality of the remote interface. Upon logging in, the user's credentials are sent through Django's user authentication. Once authorized, the user can navigate to the "Experiment" tab and create an experiment to run during their reserved time. Creating an experiment involves selecting the desired hardware (e.g. ECU or Brake Controller), filters (e.g. number of data points or smooth curve), graphs (e.g. Speed vs. Time) and experiment duration. This is accomplished by creating a thread that passes from the Web Interface through the layered architecture to the CAN Data Processor, which translates the node controller's data. This thread is what allows the user's experiment settings and graph data to be passed to the node controllers found in the testbed. Once the node controllers receive their instructions, they respond by sending J1939 data using the data processing thread. Once processed, the data is converted to JSON so the information can be represented graphically on the researcher's display. The experiment duration specified by the user determines how long the browser requests information from the node controllers. Throughout the experiment, the researcher has the liberty to inject their own traffic into the experiment. This traffic is sent to the data processor and to the node controllers. Eventually the injected traffic arrives at the desired ECUs and their responses follow the path back to the browser according to the standard J1939 protocol. When the experiment completes, a signal is passed to the node controllers, which causes the data processor thread to terminate and the experiment to be saved one final time. This flow of functionality allows the user to witness the direct results of injecting their traffic into a running simulation and promotes efficient and secure means of sending experiment data through the network.

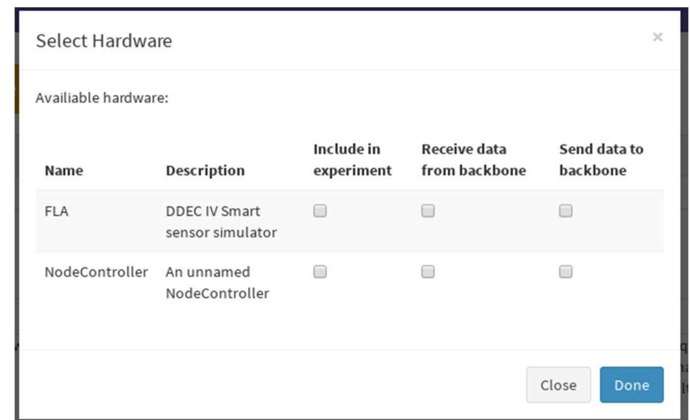


Figure 8. The Select Hardware Interface

Figure 8 shows a sample of how the user selects the available hardware in the testbed to work with. Figure 9 shows how the user can add charts to display the data. One design challenge for the remote interface is determining the best way to present the data generated by experiments by updating charts and tables in real time. A JavaScript library called Chart.js handles data rendering for graphs. The charts can display any SPN that has a numerical value.

Another type of display that can be added is a data table, as shown in Figure 10. Tables are also updated in real-time and can display any PGN.

After the user adds the desired charts and tables, empty displays are shown on the experiment page awaiting experimental data. Once the experiment is started, data will begin to populate the specified charts and tables. Figure 11 shows the entire interface with a table and a graph being populated by the experiment.

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

Figure 9. Adding a Chart Display

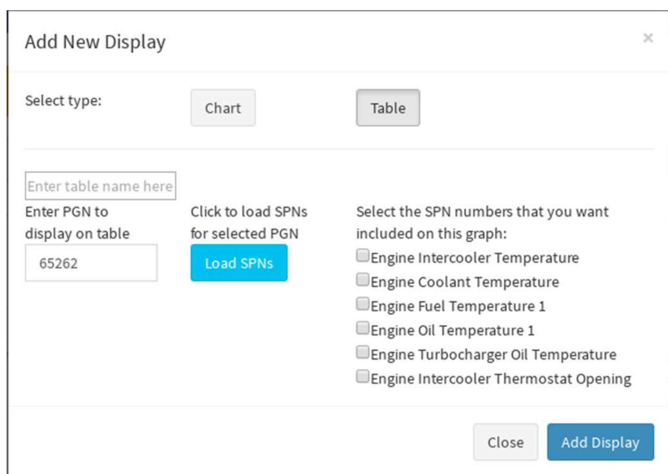


Figure 10. Adding a Table

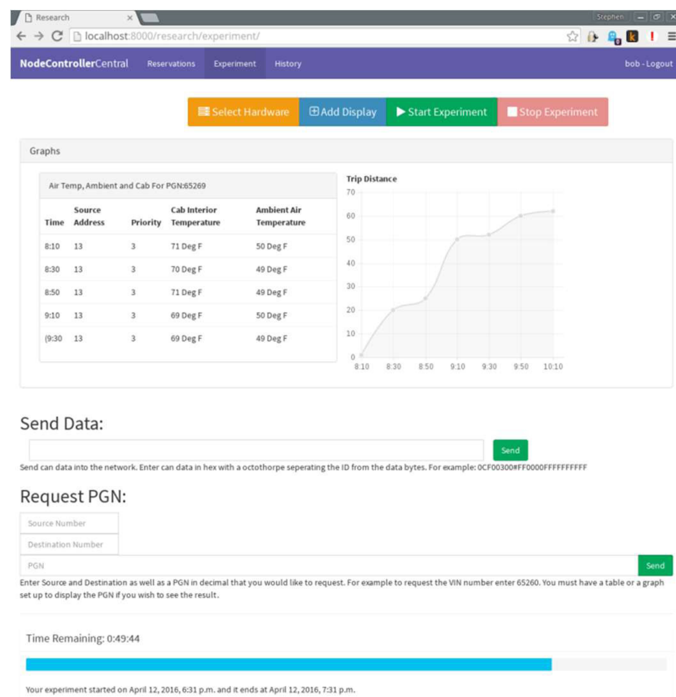


Figure 11. Populating a Data Table and a Graph

TESTBED EXPERIMENTS

To demonstrate the utility of the testbed, sample experiments are described with some results in this section.

Seed/Key Exchange Experiment

When an ECM is communicating with diagnostic software, there may be routines that escalate privilege. As such, an ECM may request a key from the diagnostic software. To standardize this transaction, ISO15765 describes a security setup session. When the diagnostic software begins a security transaction, it requests a seed from the ECM. The ECM will respond with a seed value of 16 bits. The diagnostic software performs some math on the seed and transmits a key to the ECM. If the key from the diagnostic software matches the key that the ECM was anticipating, then the ECM will proceed as if it has been authenticated.

Of interest is the feasibility of cracking the seed/key exchange using a brute/force attack. With only 16 bits for a key, there are 65536 combinations. The secret of the key is coded within the diagnostics software. Therefore, a PC with the diagnostics software is connected to the testbed through an RP1210 adapter. At the start of the seed key exchange, the message traffic as observed on the J1939 bus is shown in Table 1.

To harvest valid keys, the node controller connected to the ECU would emulate the ECU's response to the PC's request for a seed. A loop was constructed to give the PC every seed from 1 to 65536 and

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

record the key that is sent by the PC. In this fashion, all seed key pairs were obtained. The results, shown purposely with less detail, appear in Figure 12.

Table 1. Example of a Seed/Key using the testbed.

#	CAN ID	B1	B2	B3	B4	B5	Note
1	0x18DA00F1	0x02	0x27	0x05	0	0	PC requests the ECM for a Seed.
2	0x18DAF100	0x04	0x67	0x05	0x81	0xB7	ECM responds with a Seed
3	0x18DA00F1	0x04	0x27	0x06	0x16	0x98	PC sends a key
4	0x18DAF100	0x02	0x67	0x06			ECM acknowledges key

The PC software was programmed to run in an infinite loop that has an exit criterion of the ECM acknowledging the exchange. As such, there was no effort needed to manipulate the PC software; all data was obtained by simply asking for it over and over again. Each seed - key pair took 11 seconds to obtain, which is about 200 hours total. Once finished, the seed/key pairing turns into a 192kbyte lookup table. No knowledge of the algorithm is needed.

If the security seed and key space were expanded to fill the remaining 4 bytes of the CAN frame, there would be 48 bits to work with. This creates a space that has 2.81475E+14 pairs, which at 11 seconds per pair would take 98,180,642 years.

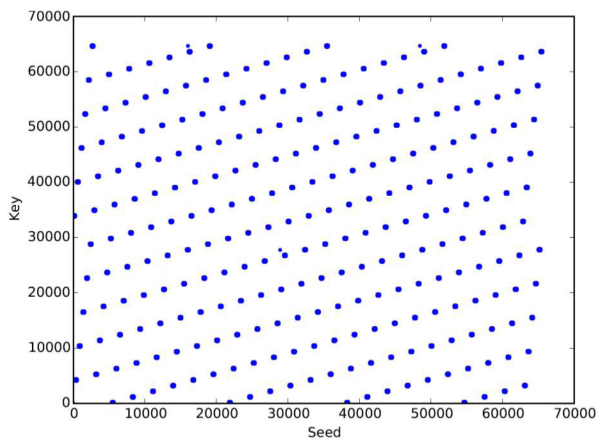


Figure 12. Result of querying the PC diagnostic software for seed/key pairs.

Intrusion Detection

Liao et al. [17] classify intrusion detection systems into three broad categories: signature-based, anomaly-based, and stateful protocol analysis. Signature-based systems look deep into the traffic like data to find patterns which match intrusion patterns. Anomaly-based systems match statistics and other information about network traffic to identify behavior which differs from normal flow. Stateful protocol analysis compares state transitions of regular protocol to that of intrusions; thus, some may argue that this falls under the broad spectrum of anomaly detection.

The following techniques are typically used for intrusion detection systems: statistics-based, pattern-based, rule-based, state-based, and heuristic-based. Statistics-based approaches identify intrusions based on probabilities and statistics and compares them to predefined thresholds. Pattern-based approaches match known attack patterns to incoming network flow patterns. Rule-based approaches validate incoming traffic against rules to detect intrusion. State-based approaches define finite state machines for modeling correct behavior and validate incoming traffic against probable state transitions. Heuristics-based approaches make use of heuristics to identify malicious traffic.

In the following, we enumerate some challenges specific to the CAN network which makes it hard to adapt the existing work on IDS. IDS must operate in real-time - the delay between the occurrence of the attack and its detection must be minimized. It should also be automated to the extent possible and require minimum human intervention. IDS must be placed such that the normal message control flow should not be disrupted. IDS may have to operate in resource constrained devices and perhaps use real-time training data. IDS must be upgradable as new attacks are discovered.

We first investigate what types of IDSs are suitable for heavy vehicles. Signature-based IDS require a pattern matching based on known attacks. Since attacks on heavy vehicles are relatively infrequent compared with the Internet, we may not have sufficient data to develop patterns of attack. Anomaly-based and stateful approaches are more likely to work. However, such techniques are resource consuming and so must be adapted to work for low powered embedded CAN controllers. CAN is a broadcast protocol and also there are a large number of variations which make it harder to model normal behavior. False positive rate is typically high in anomaly detection network. However, a high false positive rate may be fatal for heavy trucks. In spite of these challenges, researchers tend to use anomaly-based and stateful approaches because of the lack of availability of CAN/J1939 based attack signatures.

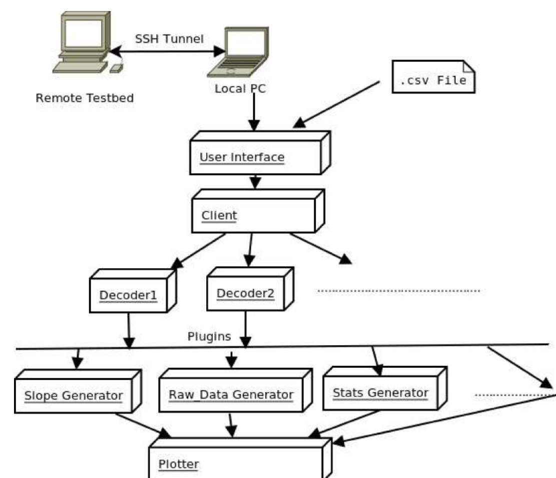


Figure 13. Remote intrusion detection system

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

Location of the IDS is also an important issue. In IP networks, IDS can be deployed in the host, network, or can be distributed over the host and network. In the context of CAN, the location of the IDS may give rise to a set of architectures which we plan to explore in our testbed and identify their merits and demerits. Host-based IDSs may not work well because the ECUs may lack the resources needed to do the complex computations. In addition, the ECUs must be modified to incorporate the new IDS. Moreover, IDSs are subject to change and such evolution is complex if their behavior is intertwined with the ECU functionality. Introducing a new embedded device that hosts the IDS is another alternative. However, this requires the heavy vehicles manufacturer to modify the bus architecture. A third alternative is to use a remote IDS which is a high-powered computationally intensive processor external to the CAN bus which performs the functionality of the IDS and sends alerts to the driver.

Such a remote IDS architecture is shown in Figure 13. The architecture consists of the following components.

User Interface

The user can choose the data sources (remote testbed or local sources), the plug-ins options, the number of records to process at a go, sleep time between processing data packets, and the PGNs and SPNs which will be analyzed.

Client

The client is responsible for reading data from a local file or remote connection. Remote data is read securely using an SSH tunnel, which forwards data from the remote testbed to a local server.

Decoders

The set of decoders for the PGNs and SPNs which helps in decoding the J1939 messages in real-time for analysis by the user.

Plug-Ins

A set of user developed plug-ins which can receive decoded SAE J1939 messages and use them to perform various tasks. For example, we can have a plug-in that compares related data generated by two ECUs to detect anomalies. We can have a plug-in that computes the slope of the data over time and detects abnormal behavior.

Plotter

A single instance of a plotter which can plot various user data as supplied to it by the plug-ins.

The IDS framework introduced in this paper adopts the train-classify-report-research approach as established by Cain et al. [19]. However, instead of using archived and processed training data, we opt for time-series data, as accumulated during the most recent runtime phases of the vehicle (from when the vehicle started). This allows us to mitigate the effect of two critical extraneous factors namely, environmental influences like road and weather conditions and proprietary characteristics of vehicles. The assumption behind this approach is that the vehicle will be in a stable state at the time of start. If at all unstable at that time, in-vehicle mechanisms are expected to detect those anomalies. The IDS will train on the streaming data as received over time and attempt to classify two broad categories of anomalies: data flow based anomalies and traffic statistics based anomalies. Data flow based anomalies refer to abnormal changes in data received from different ECUs, whereas traffic statistics based anomalies refer to abnormal traffic features like volume of traffic and abnormal protocol state changes.

We identified 3 features, namely, abnormal rate of change of data values, variant information received from different ECUs about the same PGN-SPN pair, and abnormal message sequences.

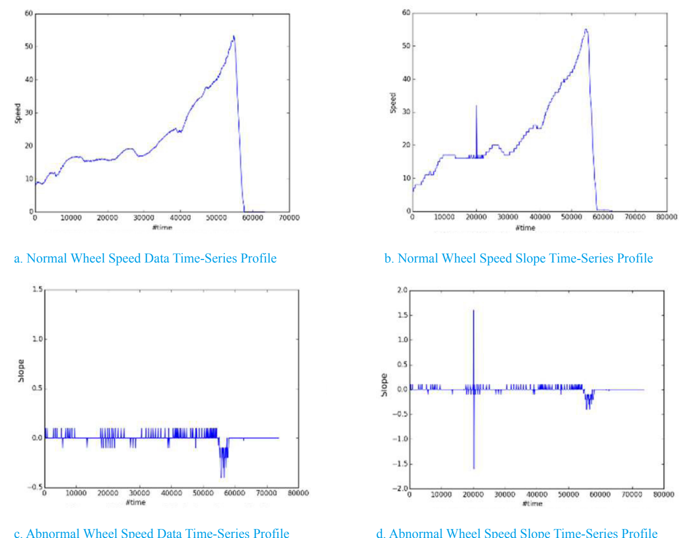
The first of these features make use of the slope on streaming CAN data. One of the plugins which receives the raw PGN-SPN data along with the timestamps, calculate the slope as shown in Eq. 1.

$$slope = \frac{current_value - pre_value}{current_timestamp - pre_timestamp} \tag{1}$$

The second feature compares raw data from two ECUs. In this case, we can use a plugin which takes raw data from the two inputs and performs a non-parametric statistical significance test on a moving window of certain width, to determine whether the two datasets received from the two ECUs are different.

Intrusion Detection Preliminary Experimental Results

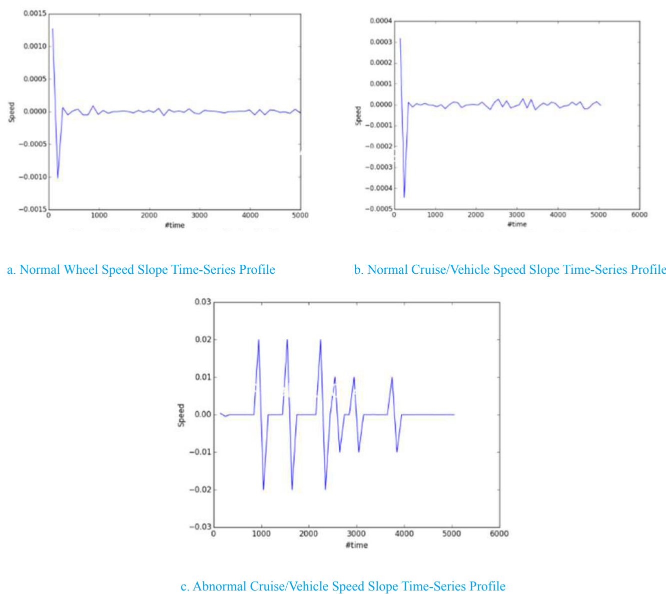
In this section we will specify three features we identified we used to detect subtle anomalies in J1939 networks.



a. Normal Wheel Speed Data Time-Series Profile
b. Normal Wheel Speed Slope Time-Series Profile
c. Abnormal Wheel Speed Data Time-Series Profile
d. Abnormal Wheel Speed Slope Time-Series Profile

Figure 14. Data flow based anomaly: Abnormal Rate of Change of Wheel Speed Data (PGN: 65215, SPN: 904) (x-axis: absolute timestamp, y-axis: speed/slope)

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017



a. Normal Wheel Speed Slope Time-Series Profile

b. Normal Cruise/Vehicle Speed Slope Time-Series Profile

c. Abnormal Cruise/Vehicle Speed Slope Time-Series Profile

Figure 15. Data flow based anomaly: Abnormal Difference in Similar Information from Wheel Speed Data (PGN: 65215, SPN: 904) and Cruise/ Vehicle Speed Data (PGN: 65265, SPN: 84) (x-axis: absolute timestamp, y-axis: speed/slope)

Data Flow-based features

Experiments are performed on the Paccar PX-8 Engine data as obtained from a network recording of a Peterbilt service truck with a Paccar PX-8 engine.

Abnormal Rate of Change of Data

Figure 14a depicts the normal engine speed behavior of Paccar PX-8 Engine data. This data shows an increase and then starts decreasing after reaching a peak value. Training an anomaly classifier on this data may be hard since it is ever changing and does not abide by any static parameter which can be used to model this information. However, a better perspective can be obtained from Figure 14b, where we plot the rate of change of engine-speed instead of the absolute data values. An interesting feature is that the positive edges on the y-axis have almost the same magnitude. This may be a potential predictor of anomalies as shown in Figure 14c and Figure 14d, where we manually compromised the integrity of one PGN 65215 message, and obtained the corresponding slopes. The slope in Figure 14d shows considerable change in positive y-axis peak.

The challenge is to find an efficient approach to predict this anomaly in such time series data and this approach can be applied on other SPNs. Moreover, since some SPNs generate categorical resolutions, calculation of slope poses its own challenges.

Abnormal Difference in Similar Information

Given SAE standards, multiple ECUs can provide the same or related information. For example, Wheel Speed Information (PGN: 65125), Front Axle Speed (SPN: 904) and Vehicle Speed 1 (PGN: 65265), Wheel-based Vehicle Speed (SPN: 84) provide almost similar information. This can be seen in Figure 15a and Figure 15b. We performed a non-parametric statistical significance test (Mann-Whitney-Wilcoxon Test) and the two slope datasets did not show significant difference (p-value = 1.17) at .05 confidence level. However, once we modified the data in Figure 15c, the same test revealed a p-value of 0.04, showing significant difference at 0.05 confidence level. Thus anomalies may be observed by detecting significant differences between data received from two separate ECUs producing similar information.

However, a lot of research needs to be done. First, we still need to perform this in real time and this requires identifying a suitable time window over which we can measure these differences. Second, we need to fix a correct sample size over which the statistical significance tests need to be performed. This may increase the number of false positives if the sample size or the window size is chosen incorrectly.

Traffic Statistics-based features

Multiple traffic statistics-based profiles can be created from CAN messages. However, this should not be much different from previously proposed research. Profiling the normal behavior of the trucker needs to done in terms of the traffic statistics (such as, volume) and patterns (such as, message timing correlations with respect to J1939 protocol flow).

Evaluation

In future, we plan to investigate whether real-time constraints can be satisfied by our IDS mechanism. Otherwise, we need to perform domain specific sampling. Our future work will investigate the usefulness of the various sampling techniques for real-time IDS.

We also need to evaluate our IDS against the standard metrics proposed by Tavallaaee et al. [18] for IP networks. This includes detection rates (fraction of alerts compared to the number of intrusions), false positive rates, accuracy, precision, detection and response times, and cost. In addition to these metrics, the resources consumed by the security mechanisms and timeliness measures will play a key role. If the resource consumption, response times, or costs are high, then the IDS must operate on sampled data. Here again, we need to define how to perform sampling so that accurate results are obtained.

Our eventual goal is to provide a repository of IDSs for J1939 traffic and provide a database of types of attacks detected, and requirements needed to deploy the specific IDS, and how they perform under the given metrics.

SUMMARY/CONCLUSIONS

According to the American Trucking Association (ATA), nearly 70% of America's freighted goods are transported by heavy trucks, which represents more than 9.2 billion tons of freight. America thrives because of the trucking industry. Research in cyber security for heavy vehicles is a means to improving and safeguarding this industry. The remote testbed can be used to aid in the process of improving security standards for heavy trucks as well as in discovery of potential vulnerabilities. This allows researchers to be more productive,

Downloaded from SAE International by Colorado State Univ, Wednesday, June 21, 2017

achieve results faster, and explore additional experimental possibilities that may not have been feasible when tethered to a physical truck.

REFERENCES

1. Kantor B., Anderson, D, Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., Checkoway, S., and McCoy, D., "Comprehensive experimental analyses of automotive attack surfaces," in USENIX Security, 2011.
2. Sandia National Laboratories, SCADA Testbed, http://energy.sandia.gov/energy/ssrei/gridmod/cyber-security-for-electric-infrastructure/scada-systems/testbeds/, accessed 4-8-16.
3. Luiijif E. and Christiansson H., "Creating a European SCADA Security Testbed," in *Critical Infrastructure Protection*, Goetz E. and Shenoi S., Eds., 2007, pp. 237-247.
4. Ruth, R. and Daily, J., "Accuracy and Timing of 2013 Ford Flex Event Data Recorders," SAE Technical Paper 2014-01-0504, 2014, doi:10.4271/2014-01-0504.
5. Checkoway S., McCoy D., Kantor B., Anderson D., Shacham H., Savage S., and Koscher K., "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in Usenix Security 1, 2011.
6. Koscher K., Czeskis A., Roesner F., Patel S., Kohno T., Checkoway S., and McCoy D., "Experimental Security Analysis of a Modern Automobile," in IEEE Symposium on Security and Privacy, Oakland, CA, 2010.
7. Society of Automotive Engineers, "J1939-11: Physical Layer, 250k Bits/S, Twisted Shielded Pair," 2006.
8. Kleine-Budde M., "SocketCAN--The official CAN API of the Linux kernel," in Proceedings of the 13th International CAN Conference (iCC'12), Hambach Castle, Germany, 2012, pp. 5-17.
9. J1939 on Linux Project, 2014. Last accessed from http://elinux.org/J1939 on July 30, 2016.
10. Texas Instruments, Programmable Realtime Unit Subsystem, http://processors.wiki.ti.com/index.php/Programmable_Realtime_Unit_Subsystem, accessed 4-9-16.
11. SAE Surface Vehicle Recommended Practice, "Data Link Layer," SAE Standard J1939-21, Rev. Dec. 2010.
12. International Standards Organization, "ISO 15765-1: Road vehicles -- Diagnostic communication Over Controler Area Network," 2015.
13. SAE Surface Vehicle Recommended Practice, "Serial Data Communications Between Microcomputer Systems in Heavy Duty Vehicle Applications," SAE Standard J1708, Rev. Dec. 2010.
14. Daily, J., Johnson, J., and Perera, A., "Recovery of Partial Caterpillar Snapshot Event Data Resulting from Power Loss," SAE Technical Paper 2016-01-1493, 2016, doi:10.4271/2016-01-1493.
15. Microchip Technology Inc., "Mcp41hv51 Data Sheet, " http://ww1.microchip.com/downloads/en/DeviceDoc/20005207B.pdf, accessed 4-10-16.
16. Analog Devices, Inc., "Adg1401 Datasheet, " http://www.analog.com/media/en/technical-documentation/data-sheets/ADG1401_1402.pdf, accessed 4-7-16.
17. Liao H.-J., Lin C.-H. R., Lin Y.-C., and Tung K.-Y., "Intrusion detection system: A comprehensive review " *Journal of Network and Computer Applications*, vol. 36, pp. 16-24, 2013.
18. Tavallee M., Stakhanova N., and Ghorbani A. A., "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 40, pp. 516-524, 2010.
19. Cain Hareil, "CAN Traffic modelling, Applying Machine Learning for Anomaly Detection in CAN bus networks," 13th ESCAR Europe, Cologne, November 2015.

CONTACT INFORMATION

Jeremy Daily can be reached at
jeremy-daily@utulsa.edu

Rose Gamble can be reached at
gamble@utulsa.edu

Indrakshi Ray can be reached at
indrakshi.ray@colostate.edu

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1619641 and Grant No. 1619690. This work was partially sponsored by National Motor Freight Traffic Association, Inc. (NMFTA). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of NMFTA or the U.S. Government.

DEFINITIONS/ABBREVIATIONS

CAN - Controller Area Network

GPIO - General Purpose Input Output

IDS - Intrusion Detection System

PGN - Parameter Group Number (from J1939)

PRU - Programmable Realtime Unit

SoC - System on a Chip

SPN - Suspect Parameter Number (from J1939)

UART - Universal Asynchronous Receiver/Transmitter

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE International.

Positions and opinions advanced in this paper are those of the author(s) and not necessarily those of SAE International. The author is solely responsible for the content of the paper.