OCTANE: An Extensible Open Source Car Security Testbed

Parnian Najafi Borazjani
George Mason University

Christopher E. Everett
George Mason University

Damon McCoy
George Mason University

Abstract

Security research and training using cyber-physical systems (e.g., automotive networks) is challenging because of the need to replicate the interactions between the hardware components and the control software of the systems. These interactions are challenging to replicate due to dynamic inputs in real-world environments that cause various interactions of hardware components and control software within the network. In particular, automotive networks are challenging for security research and training because although the protocols of the automotive networks are standardized (e.g., CAN, LIN), the implementation details by each automotive manufacturer are not standardized and are generally not publicly available.

In this paper, we present OCTANE, which reduces the barrier of entry into the security research and training of automotive networks by providing a software package and a hardware framework for the reverse engineering and testing of automotive networks. OCTANE provides a platform for security research and training by replicating the interactions between the hardware components and control software of the systems so that the user can focus on the security aspects of the automotive network instead of the tool configuration and setup. In addition, OCTANE includes security focused features, such as the ability to replay and fuzz test different automotive network protocols. OCTANE will be released as an open source tool, which will enable the automotive security community to extend this tool to support future automotive protocols.

1 Introduction

Modern automobiles are pervasively controlled by networked computers and recent studies have demonstrated that these system are vulnerable to attackers with access to internal automotive networks [28, 33] and remote external attackers [17]. Automotive cyber-physical security research is challenging because of steep barriers to entry due to the fact that there is no open source automotive testbed. In particular, automobile network research presents additional challenges due to the lack of documentation of manufacture specific proprietary protocols built on top of standardized protocols utilized by automotive manufacturers. This forces every research group to build their automotive testbed software from scratch and often times enter into restrictive agreements with automotive manufactures to obtain access to the documentation necessary to build such a testbed, thus preventing the release of their testbed software.

We presented our initial design and implementation of OCTANE [20], which is an automotive security testbed that will facilitate the analysis, understanding, and testing of automotive cyber-physical systems. OCTANE enables researchers and students to rapidly begin to explore automotive cyber-physical systems by providing a platform for reverse-engineering and testing through real-world experimentation of a lab network setup or an automobile.

This paper is both an update on more advanced functionality that has been integrated into OCTANE and an introduction of OCTANE to the automotive industry community. OCTANE is composed of both a software package and hardware framework. The software package enables monitoring and transmitting of CAN [5] messages that can be used to perform general purpose network diagnostic and debugging functionality and includes many features to facilitate reverse engineering of proprietary protocols. In addition, OCTANE has the ability to perform automated security “fuzz” testing and replay testing of Electronic Control Units (ECU). Also, OCTANE will be released under an open source license and is designed to be modular and easily extended. This gives it a distinct advantage over closed sourced solutions, such as Intrepid’s vehicle spy software that is not focused on security analysis. The free and open nature of OCTANE reduces the burden on those interested in exploring automotive cyber-physical security by allowing the them to concentrate on security research and not the minutiae of setting up an automotive testbed.

2 Background

Previously, in order to provide automotive security, we should have just used mechanical locks or car alarms but nowadays, the advent of on-board computers has changed the problem statement fully. Automotive networks are getting more complex with the ever growing number of electronic control units (ECUs) and subnetworks that are used to connect the ECUs [31, 41, 42]. The automotive networks usually include a controller area network (CAN) sub-network [5], a FlexRay sub-

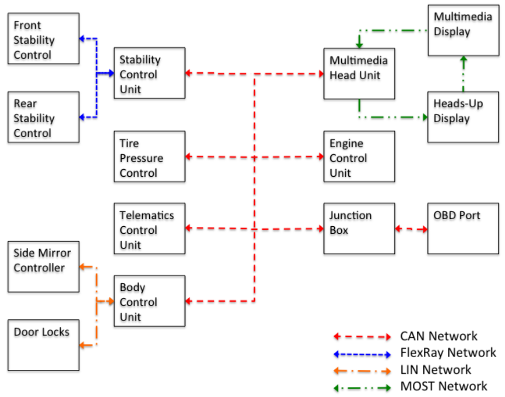


Figure 1: Exemplary automotive network.

network [7], a local interconnect network (LIN) sub-network [11], and a media oriented systems transport (MOST) sub-network [34]. Each sub-network has their own technical details such as bus speed that can be beneficial in different scenarios requiring fast multimedia transfer or low cost.

The CAN sub-network is the backbone of the automotive network that communicate between the other sub-networks. The CAN sub-network is a resilient network that is able to operate even if one or more of the ECUs are defective or with the increased electromagnetic noise in an automotive environment. The CAN sub-network also provides an interface for the standardized on-board diagnostic (OBD) port citeOBD-standard, which enables emissions testing and hardware testing. The FlexRay sub-network is used for safety critical automotive applications (e.g. stability control, back-up detector)and high-speed application. Although FlexRay hardware costs more comparing to CAN hardware, however, it provides higher bandwidth and fault tolerance and redundancy and a combination of fixed, time-triggered transmit windows and dynamic transmit windows for important messages. The LIN sub-network is for low-bandwidth, low-speed automotive applications(e.g.,door lock/unlock, side mirror controls). LIN was developed to replace CAN in these application for cost reduction purposes. The MOST sub-network is for high-speed, high-bandwidth multimedia automotive applications (e.g., video camera, video streaming). Generally, the cost for the hardware components and the complexity of the corresponding protocol also corresponds to the bandwidth. In other words, each sub-network has particular advantages and dis-advantages and can be utilized by automotive manufacturers to decrease the overall manufacturing cost while providing the best network performance.

Each sub-network has an industry accepted standard [5, 7, 11, 34]. On top of these standards, each automotive manufacturer has an application layer that extends the standard network protocol and implements the standard network protocol in different ways (e.g., GM-LAN [22]). Automotive manufacturers generally keep their particular implementations as trade secrets as set forth in the AUTOSAR working principle of “Cooperate on standards, compete on implementation” [3]. Therefore, these proprietary application layers are generally not publicly available. The proprietary application layers developed by the automotive manufacturers provide a wealth of knowledge about the network implementation but it should be reverse engineered in order to be able to use it. [1].

Not only automotive networks is used for safety and control of the vehicles but also, it enables automobile maintenance through the OBD port and telematics control unit (e.g., cellular network updates [19]). The maintenance can be as simple as reading the error codes of an ECU through the OBD port using a testing tool [12] or as complicated as re-programming an ECU [2]. Automobile maintenance through the OBD port and telematics control unit is important for the industry to reduce maintenance cost of automobiles.

3 Related Work

In the automotive security field, a few software packages [26, 28, 33, 35], lab network setups [24, 26, 28, 38] and real-world test setups [17, 28, 33, 38] have been utilized by prior researchers. These testbeds were limited to the specific security testing technique that was being evaluated and based on the publicly available information on these testbeds, these testbeds were not designed for easy setup and use by other users through a graphical user interface. OCTANE is designed to enable users to quickly move from a basic understanding of automotive networks [36] to testing security solutions on automotive networks [3].

The prior work provides an overview of the security issues within automobile networks [24, 26–28, 33, 37]

[1] Each of these proprietary application layers includes: (i) ECU access control protocols; (ii) ECU re-programming application program interfaces (APIs); (iii) ECU memory access APIs; (iv) diagnostic APIs; (v) ECU parameter modification APIs; and (vi) network control APIs.

[2] The re-programming of ECUs can be done using automotive manufacturer approved tools [1] or through third party tools (e.g., PCLink [13], TunerPro [15]). The re-programming can range from fine-tuning control parameters (e.g., higher engine idle RPMs) to an updated version of the control software (e.g., new version to fix braking issues).

[3] OCTANE is not currently designed for autonomously operated automobiles or vehicle-to-vehicle communication and as such, the extensive prior work in these fields are not discussed in section 3.

and proposed solutions to the security issues [26, 40]. The proposed solutions include honeypots [29, 39], firewalls [41], intrusion detection systems [25, 26, 29], encrypted communication [23, 32, 38, 41, 42], ECU authentication [41, 42], and secure communication techniques [16, 30]. Kleberger et al. [27] provide a comprehensive overview of the security research in the automotive security field and possible security solutions for automobiles. However, only a few of these overviews implemented security solutions for the security issues on lab network setups or real-world test setups [26]. OCTANE is designed to enable users to configure and test the proposed solutions to the security issues described in the prior work.

4 Testbed

Open Car Testbed And Network Experiments (OCTANE) consists of a Software package and a hardware platform that can be used for testing and reverse engineering of automotive networks. OCTANE monitor packets and enables reverse engineering by providing a packet monitor with customized packet identification (e.g., identify door lock packet, identify ECU reprogramming request packet) and a customized packet transmitter that enables us to verify or understand the packet functionalities by injecting the monitored packets to the network (e.g., the ECU re-programming request packet actually initiates the ECU re-programming process, the identified lights-on packet actually turns on the lights). Decoding of the monitored packets enables a researcher to quickly reverse engineer the proprietary protocols utilized by automotive networks. This operation is made even easier by the ability of naming the packets and recording the names in an XML document. The identify packet feature not only lets the researcher immediately see all the identified packets in the packet monitor but also allows other researchers to use the findings by distributing the XML files.

OCTANE paves the way for automatic network testing and security solution testing on a network. For example, a firewall could be configured to stop ECU reprogramming from an unauthorized device such as nontire pressure sensor packets from entering through the tire pressure controller. OCTANE provides the ability for a researcher to (i) monitor a CAN bus through an automobile's OBD port, (ii) replay parts of the monitored traffic, and (iii) monitor the CAN bus to determine how the automobile handles the replayed traffic. In this example, the replayed traffic could be changing a gas gauge command that the researcher is attempting to replicate. In another example, the researcher could send a sequence of messages in order to accomplish the task, which can be similar to a series of challenge response messages. The replayed traffic could be a ECU reprogramming command that the researcher is attempting to replicate to reprogram an ECU. In each example, after the researcher verifies that the replayed traffic correctly controls the respective part of the automobile, the researcher can save the replayed traffic in an XML file for future replay or sharing with other researchers. The software and the hardware of OCTANE are discussed below in turn.

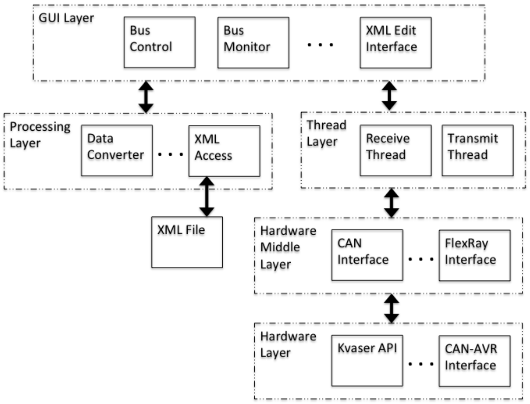


Figure 2: Architecture of software package. The architecture is designed to enable the software package to use diverse hardware interfaces by adding the application programming interface to the hardware middle layer without re-coding of the GUI layer or the thread layer.

4.1 Software

The architecture of the software package is depicted in Figure 2. The architecture includes a presentation layer (also referred to as GUI layer), a business logic layer that includes both processing layer, a thread layer, a hardware middle layer, and a hardware layer. The layered nature of the software package enables flexibility and adaptability in the types and quantities of network data being processed. First the software layers are explained briefly. Following the brief description of each layer, different components of the software package are discussed.

The presentation layer provides the GUIs for the software package. Screenshots of some of the GUIs are provided herein (i.e., Figures 3 and 5).

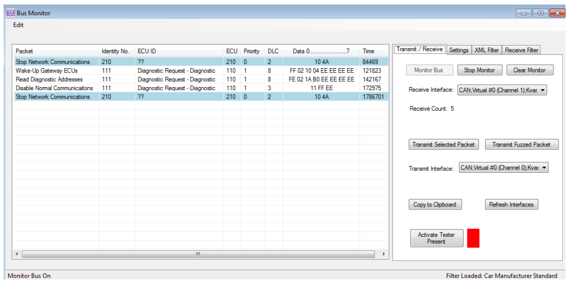
Additional GUIs can be quickly added to the software package for new features (e.g., ECU re-programming). The GUI layer has a direct connection to the business logic layer that incorporates the processing layer and the thread layer. The processing layer handles data manipulation for the software package (e.g., converts user input into CAN message format), logging of messages, and access to XML files for the XML automation described in section 4.1.3. The separation of the business logic layer and the presentation layer is done in order to off-load the data processing that would be beneficial considering the significant number of packets that are received from the car.

The thread layer provides threading mechanisms for the receiving and transmitting of data (e.g., CAN packet, LIN packet) through the appropriate hardware (e.g., Kvaser CAN adapter, Kvaser LIN adapter). The thread layer removes the delay in receiving and transmitting data from the GUI layer so that the GUIs are not stalled during the receiving and transmission process. The thread layer calls the receive and transmit interfaces of the appropriate hardware interfaces in the hardware middle layer (e.g., generic CAN interface, generic LIN interface).

The hardware middle layer is utilized to obscure and separate the implementation details of the actual hardware from the other layers of the software package. For example, a new hardware adapter (e.g., new CAN adapter) can be added to the software package by adding in the application programming interface (API) for the new hardware adapter in the hardware middle layer without having to re-code any part of the presentation layer, thread layer, or business logic layer. The hardware middle layer utilizes the appropriate hardware API in the hardware layer. The hardware layer is the API that the hardware device manufacturer provided or is coded for the particular hardware device (e.g., the CAN-AVR interface connects through a standard serial connection). The various layers work together to streamline the operation of the software package while allowing for extensions to the architecture.

4.1.1 Adapters

The advanced bus control interface of software package (not shown) controls the operation of the automotive network adapters (i.e., CAN [4, 6, 10], LIN [9], FlexRay [8]). The software package utilizes a variety of different hardware controllers. Moreover, other hardware controllers can quickly be added to the architecture as described above in section 4.1. The bus control interface enables a user to choose and configure the options of the various hardware controller [4]. The guided bus control, which is loaded by default, determines if any of the Kvaser, Intrepid or Ecom cables are connected to the computer and recommends sensible default configuration parameters along with the option for more advanced configuration of the adapters. If no hardware adapters are detected OCTANE implements a virtual interface that can be used for simulations.



Packet	Identity No.	ECU ID	ECU	Priority	DLC	Data 0.......................7	Time
Stop Network Communications	210	??	210	0	2	10 4A	84469
Wake-Up Gateway ECUs	111	Diagnostic Request - Diagnostic	110	1	8	FF 02 10 04 EE EE EE EE	121823
Read Diagnostic Addresses	111	Diagnostic Request - Diagnostic	110	1	8	FE 02 1A B0 EE EE EE EE	142167
Disable Normal Communicaitons	111	Diagnostic Request - Diagnostic	110	1	3	11 FF EE	172975
Stop Network Communications	210	??	210	0	2	10 4A	1786701

Receive Interface: CAN;Virtual #0 (Channel 1);Kvas

Receive Count: 5

Transmit Interface: CAN;Virtual #0 (Channel 0);Kvas

Monitor Bus On

Filter Loaded: Car Manufacturer Standard

Figure 3: Screenshot of the bus monitor interface of the software package. The bus monitor interface outputs the received packets and allows for transmission of packets back to a network.

<Packet>		
<Name>	Stop Network communications	<\Name>
<ID>	210	<\ID>
<DLC>	2	<\DLC>
<Message>	104A	<\Message>
<\Packet>		

Table 1: XML example of an imaginary packet for transmission and identification.

4.1.2 Monitor

Figure 3 illustrates bus monitor interface in the software package. The bus monitor interface outputs the received packets and allows for transmission of packets back to a network. The bus monitor displays the received packets from the selected *Receive Interface*. As illustrated in figure 3, the bus monitor enables a user to quickly and efficiently view the received packets in a human-readable form (i.e., English text readable form versus hexadecimal form) and identify packets upon reverse engineering of the proprietary application layer (e.g., Honda application layer can be defined as a CarType in the XML file). The combined efforts of a community of users, via the sharing of a XML file, could significantly reduce the time required to reverse engineer a proprietary application layer.

Filters. There are two types of filtering that is available in bus monitor. The first type of filtering is called *Receive Filter* that enables the highlighting and filtering of the packets based on the selected ID, Message or other properties. This feature helps in narrowing the scope of packets that may be beneficial in reverse engineering

[4] The configuration options include bit rate parameter, time between bits parameter, synchronization parameter, and other parameters associated with the various hardware controllers.

compared to the high volume of packets that are received from the car. The second type of filtering is the XML filter that help in identifying the known packets. The known packets are the packets that has already been identified and saved in the XML document. The filters are available using the XML file as illustrated in figure 2 and a user can select a particular car type (e.g., automobile brand, automobile make, automobile year and model). The XML file is shareable among users to facilitate community reverse engineering of the proprietary application layers for automotive manufacturers. The filter selection of a particular car type enables received packets to be identified in figure 3. Figure 3 illustrates an XML filter identification of a Stop Network Communications packet (the XML for the packet is depicted in table 1) and other packet and ECU identifications. As illustrated in figure 3, those specific packets are highlighted using the receive filter as described above.

Bit Priority Filter. Figure 3 also illustrates the package's bit priority filter. The bit priority filter can be utilized to view the quality of service (QoS) indicators for CAN networks. The quality of service for CAN networks is based on message priority so a packet with the highest priority identifier (also referred to as ECU identification although CAN does not require the identification to identify the transmitter or the receiver) obtains network access [18]. For CAN networks, the lowest identifier has the highest priority identifier and obtains network access. However, the CAN standard [5] does not allow for all zeros in the identifier (note that zeros are dominant in CAN), so the identifier would not be all zeros. To this end, ECUs on a CAN network utilize the most significant bits and the least significant bits to differentiate quality of service indicators. Also, some proprietary network application layers increase the least significant bit by one for responses to requests (e.g., request identification is 630 and response identification is 631). For these networks, the bit priority filter enables users to associate requests and responses together through the most significant bit field. The bit priority filter enables a user to change the number of bits in the *Settings* tab. Thus, the bit priority filter provides a user with the tools to reverse engineer proprietary network application layers through the identification and analysis of quality of service indicators in packets and the request and response sequence of packets.

Tester Present. The right corner of the Bus Monitor interface includes a tester present button. When activated, the tester present button works as a heart beat of the diagnostic tool. Since some services keep the controller in a diagnostic state, we have to do the same and send the tester present every 1 or 2.5 seconds so that the ECUs listen to OBD commands.

<Packet >		
<Name>	Gas Gauge	<\Name>
<ID>	07C0	<\ID>
<DLC>	8	<\DLC>
<Message>	04300300[2/Gas]000000	<\Message>
<\Packet >		

Table 2: XML example of a dynamic packet for Gas Gauge. The dynamic part ([2/Gas]) can be replaced by any value.

Transmit Selected/Fuzzed Packets. Moreover as you can see in Figure 3, the bus monitor also enables the transmission of any of the received packets via the *Transmit Selected Packet* button to a selected automotive network selected by the *Transmit Interface*. The *Transmit Selected Packet* button functionality enables the user to select a plurality of received packets for transmission to the selected automotive network. The bus monitor enables sending fuzzed packets (i.e., changing some part of the packets) using *Transmit Fuzzed Packets* over the network. This is useful in the sense that we can see the result of changing some values that would be beneficial in reverse engineering of some dynamic packets such as gas gauge and engine RPM. The transmission aspects of the bus monitor enable a user to test interactions with the automotive network (e.g., transmit suspected door unlock packets to the automotive network) and test security features of the network (e.g., transmit seed responses to the automotive network in response to a seed request for ECU re-programming). The bus monitor includes other functionality (e.g., copy to clipboard, log to file) that is not described herein due to space limitations.

Packet Response. One of the other features that facilitates reverse engineering of the vehicle is the IFTHEN feature (refer to table 4) that is included in the bus monitor GUI. The IFTHEN feature provides for the challenge response features described in some automobile protocols. This is helpful in situations such as unlocking the controller. In some protocols, in order to unlock the controller, a seed must be requested, a key is generated using the seed in conjunction with a proprietary algorithm, and the key is sent to the controller. If the controller gets the same result as the key that was sent, it will send a positive response showing that it is unlocked.

4.1.3 XML Automation

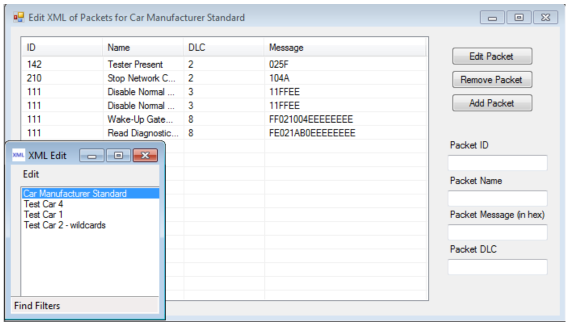
The packages's XML editing interface as shown in Figure 4 enables a user to efficiently add, delete, or modify a car or car type in the XML file. The XML editing interface also enables a user to efficiently add, delete,

<Sequence>		
<Name>	Request Seed from ECUs	<\Name>
<Packet>		
<Number>	1	<\Number>
<Name>	Standard Hello1	<\Name>
<ID>	622	<\ID>
<DLC>	2	<\DLC>
<Message>	104A	<\Message>
<\Packet >		
<Packet>		
<Number>	2	<\Number>
<Name>	Standard Hello2	<\Name>
<ID>	623	<\ID>
<DLC>	2	<\DLC>
<Message>	1F4A	<\Message>
<\Packet >		
<\Sequence>		

Table 3: XML example of a sequence of packets for Request Seed from ECUs for instance (Data values are not real). The sequence of packets is for different packets to be transmitted and identified by the software package.

<IFTHEN>		
<Name>	Challenge and Response to Seed Calculation	<\Name>
<IF>		
<Packet>		
<Name>	Standard Hello1	<\Name>
<ID>	104A	<\ID>
<DLC>	2	<\DLC>
<Message >	1010	<\Message>
<\Packet >		
<\IF >		
<THEN >		
<Packet>		
<Name>	Standard Hello2	<\Name>
<ID>	623	<\ID>
<DLC>	2	<\DLC>
<Message >	1F4A	<\Message>
<\Packet >		
<\THEN >		
<\IFTHEN>		

Table 4: XML example of a packet response for transmission upon identification of a specified packet or packets. The packet response enables one or more packets to be transmitted in response to the identification of a packet or packets.



Edit XML of Packets for Car Manufacturer Standard

ID	Name	DLC	Message
142	Tester Present	2	025F
210	Stop Network C...	2	104A
111	Disable Normal ...	3	11FFEE
111	Disable Normal ...	3	11FFEE
111	Wake-Up Gate...	8	FF021004EEEEEEEE
111	Read Diagnostic...	8	FE021AB0EEEEEEEE

XML Edit

Car Manufacturer Standard
Test Car 4
Test Car 1
Test Car 2 - wildcards

Figure 4: Screenshot of the XML edit interface for different filters and XML edit for packets. The XML edit interface enables the editing of a car or car type and the editing of packets for a car or car type.

or modify packets for the selected car type. The XML editing interface enables users to quickly and efficiently modify the XML file without having to learn the XML format (see, e.g., table 1) and to share the XML information with other users to facilitate reverse engineering of the proprietary application layer.

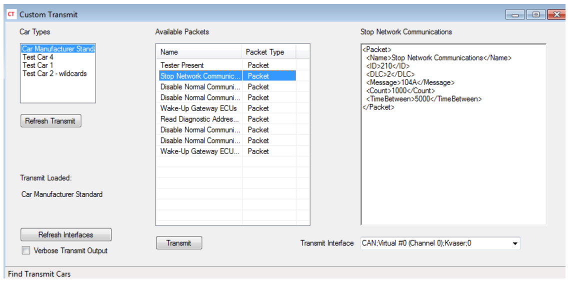
The XML automation provides the following XML extensions:

- *Dynamic Packets* (e.g., ID = [GAS/2]; Message = AE[7/VIN]); as illustrated in table 2;
- *Packet Sequences* (e.g., packet A followed by packet B); as illustrated in table 3;
- *Packet responses* (e.g., response with packet D upon receipt of packet C); as illustrated in table 4; and
- *Packet Subroutines* with sequences and responses (e.g., packets A and B and then response with packet D upon receipt of packet C).

These extensions provide the user with additional tools for the reverse engineering and testing of automotive networks. The extensions enable users to encode and share specific interactions of a proprietary application layer.

4.1.4 Custom Transmit

Figure 5 illustrates the custom transmit interface. The custom transmit interface lists available filters, such as a particular car or car type, and then the packets associated with the selected car or car type that are available for transmission. The filters and packets are stored in the XML file as illustrated in figure 2 for efficient editing and sharing. Table 1 illustrates XML schema of a packet for transmission using the custom transmit interface. The custom transmit interface enables the transmission of one or more of the selected packets on the selected *Transmit Interface*. Although the screenshot depicts CAN messages in the figure 5, any of the other network protocols (e.g., LIN, FlexRay) can be utilized by the custom trans-



Car Manufacturer Stand
Test Car 4
Test Car 1
Test Car 2 - wildcards

Name	Packet Type
Tester Present	Packet
Stop Network Communic...	Packet
Disable Normal Communi...	Packet
Disable Normal Communi...	Packet
Wake-Up Gateway ECUs	Packet
Read Diagnostic Addres...	Packet
Disable Normal Communi...	Packet
Disable Normal Communi...	Packet
Wake-Up Gateway ECU...	Packet

```
<Packet>
 <Name>Stop Network Communications</Name>
 <ID>210</ID>
 <DLC>2</DLC>
 <Message>104A</Message>
 <Count>1000</Count>
 <TimeBetween>5000</TimeBetween>
</Packet>
```

Transmit Loaded:

Car Manufacturer Standard

Transmit Interface: CAN;Virtual #0 (Channel 0);Kvaser;0

Figure 5: Screenshot of the custom transmit interface of the software package. The custom transmit interface facilitate the selection of a packet from the database of known packets in a particular car or car type and transmission of packets associated with the selected car or car type.

mission interface. The custom transmit interface enables a user to build transmission sequences for reverse engineering (e.g., how does the network respond to a certain packet?) and security testing (e.g., does the ECU let me control the engine with a certain packet?). The use of the XML file in the custom transmit interface also reduces reverse engineering and testing time by reducing manual typing of packets, configuration time (e.g., a user can load shared XML files from fellow researchers for testing), and sharing by fellow users.

4.1.5 Transmit

The software package's transmit interface (not shown) enables transmission of one or more packets. The transmit interface enables a user to specify a CAN packet for transmission to the selected *Transmit Interface*. The various features of the transmit interface (e.g., flags, number of messages, incremented identifiers for a plurality of packets) enable a user to test responses from an automobile network to various packets (e.g., how does the network respond to fuzzed packets?; how does the network respond to the same CAN message with a range of CAN identifiers?). The transmit interface decreases the user time in the setup and configuration of sending packets to the automotive network, thereby enabling the user to focus on reverse engineering and testing.

4.2 Hardware

The hardware framework for the testbed includes a lab network setup for testing of particular parts of an automobile network (e.g., window setting, door lock/unlock) and a description of how to setup tests for real-world automobiles. The lab network setup enables users to reverse engineer and test the security for isolated parts of an automobile network in a controlled environment (e.g., undergrad class project of capture the flag for door lock/unlock, research initial testing of firewall implementation on tire pressure control, re-programming of ECU). The real-world test setup enables users to extend the reverse engineering and security testing from the lab network setup to determine how the a complete automotive network operates and responds to different packets (e.g., an attempt to perform a denial of service on the CAN network using the software package was not successful in a real-work test because the CAN adapter was not able to saturate the network via the OBD port). For the lab network setup and the real-world test setup, we provide the process of setting up each setup instead of a list of actual parts so that the user can choose the optimal setup from a cost (e.g., undergraduate laboratory versus a research laboratory) and automotive network perspective (e.g., a single CAN network versus three different networks). Overall, the combination of both the lab network setup and the real-world test setup provides the foundation for users to reverse engineer and test security solutions on automobile networks.

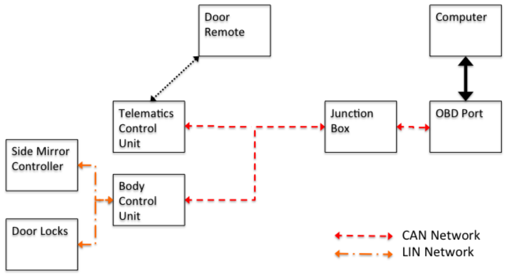


Figure 6: Exemplary automotive network for a lab network setup. A graduate student can utilize the automotive network to configure proposed security solutions for testing.

4.2.1 Lab Network Setup

Figure 6 illustrates an exemplary lab network setup for users to test automotive networks. The process for setting up the lab network includes the following steps:

(a) *Determine Research Types* that the lab network should facilitate (e.g., undergraduate laboratory, security testing);
(b) *Determine Automotive Networks Types* that need to be researched (e.g., CAN, LIN);
(c) *Identify Automobile Types* that include the automotive network types (e.g., BMW, Honda);
(d) *Identify Adapters* that include the automotive network types (e.g., Kvaser, AVR-CAN);
(e) *Determine Budget* for lab network; and
(f) *Match Automotive Parts* to meet the research types, the automotive network types, the automobile types, the adapters, and the budget.

We selected parts for a 2011 BMW X5. The selection was based on the following decision process: (a) facilitate an undergraduate laboratory, graduate research, and security testing; (b) include a CAN sub-network, a FlexRay sub-network, and a LIN sub-network; (c) certain BMW models included the three automotive network types (e.g., research through web searches [21] and review of electronic wiring diagrams); (d) Kvaser supported CAN [10] and LIN [9] network types and Intrepid Control Systems supported FlexRay network type [8]; and (e) budget was large enough to support a large network. Figure 6 illustrates part of the lab network that we assembled [5]. The computer in figure 6 is connected to the Kvaser CAN adapter, which is connected to the OBD port and to the CAN network through the OBD port, and the Kvaser LIN adapter, which can be connected to the LIN network.

4.2.2 Real-World Test Setup

Figure 1 illustrates part of an exemplary real-world network. The user can utilize a computer with a CAN adapter to access the CAN network via the OBD port. The process for selecting an automobile for real-world tests includes the following steps (similar to the process for a lab network):

(a) *Determine Automotive Networks Types* that need to be researched (e.g., CAN, LIN);
(b) *Identify Automobile Types* that include the automotive network types (e.g., BMW, Honda);
(c) *Identify Adapters* that include the automotive network types (e.g., Kvaser, AVR-CAN);
(d) *Determine Access* to different automobile types; and
(e) *Match Automobiles* to meet the automotive network types, the automobile types, the adapters, and the access.

We utilized the software package to test five real-world automobiles. We tested the following automobiles: (i) 2013 Chevrolet Cruze; (ii) 2012 Chevrolet Cruze; (iii) 2011 Chevrolet HHR; (iv) 2010 Toyota Matrix; and (v) 2006 Toyota Corolla. For each automobile, we obtained the electrical wiring diagrams [2, 14]. We utilized the OBD port to interface with the internal automobile network and utilized the electrical wiring diagram to determine which ECUs are visible via the OBD port.

5 Research and Training Opportunities

OCTANE can be utilized for research opportunities and training opportunities. Each of these opportunities are discussed below in turn.

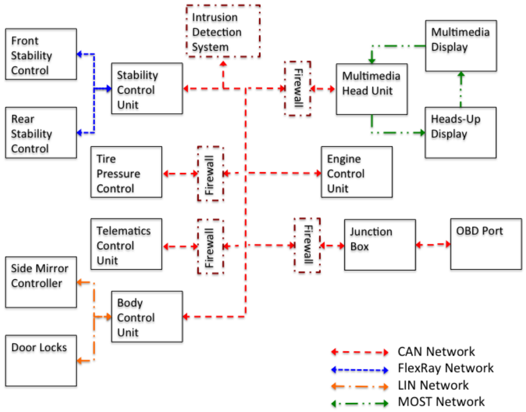


Figure 7: Exemplary firewalls and intrusion detection system on an automotive network. The firewalls are positioned at every entry point on the automotive network. The intrusion detection system is positioned on the core CAN network.

5.1 Research Opportunities

With regard to *research opportunities*, figure 7 depicts a model automotive network with model security devices. The security devices are meant to be used as security solutions for the automotive networks to prevent and stop unauthorized access to the automotive network (e.g., encryption techniques, authentication device).

Below are some of the security solutions that can be tested using the testbed:

- *Firewall* to prevent transmission of unauthorized packets into an automotive network [41];
- *Intrusion Detection System* to detect anomalies on an automotive network [25, 26, 29];
- *Packet Encryption* to protect the data on an automotive network from easy sniffing and packet insertion, besides, it will prevent side channel attacks [23, 32, 38, 41, 42];
- *ECU Authentication* to prevent unauthorized ECUs (e.g., ECU inserted by a malicious actor) from interacting with an automotive network [41, 42]; and
- *ECM Security* to detect tampering of ECUs along with authentication tampering [30].

For *training opportunities*, figure 6 illustrates an exemplary automotive network for laboratory experiments and testing of proposed security solutions. There are also many training uses of the lab network that are not related to automobile security testing (e.g., embedded operating system exercises, networking laboratory exercises). The following are some of the training uses of the testbed related to automobile security testing:

- *Automotive Network Laboratory Security Exercise* to sniff network activity and attempt to take-over the

[5] Figure 6 does not show the FlexRay sub-network for the dynamic stability control

network (e.g., control the side mirror, lock/unlock the doors);

- *Automotive Laboratory Embedded Programming Exercise* to program an AVR-CAN controller to receive and transmit packets on a CAN network [4]; and
- *Automotive Security Testing Exercise* to implement an intrusion detection system on an AVR-CAN controller for a CAN network [4].

6 Learning Points

During the development of OCTANE we encountered some pitfalls that automotive security researchers should know and take into consideration.

- *Finding Application Layer* information for a particular automobile manufacturer is challenging as described in section 2. Since automobile manufacturers usually preserve their particular implementations as trade secrets as described in the AUTOSAR working principle of ”Cooperate on standards, compete on implementation” [3]. Therefore, in order to gain access to network implementation information, we had to reversed engineer these implementation, which is time-consuming and susceptible to errors.
- *Finding Automotive Parts* is hard without having an actual vehicle. Repair facilities generally have the actual vehicle for repair and can obtain automotive parts due to the exact configuration of that vehicle. The other problem that we faced is that some of the parts require a VIN number in order to work properly and this makes the research even harder without support from the industry.
- *Electrical Diagrams* are necessary in order to identify the ECUs that can be accessed via the OBD port for testing of different vehicles [6].

7 Future Work and Conclusion

Building on our work on the software package, the next steps will be several extensions as well as security implementations for testing in automotive networks. The detailed extensions are explained below:

- *Monitoring of Actuator Inputs.* This would enable fully automated fuzz testing by being able to transmit a CAN message and monitor which actuators are triggered.
- *Wireless.* Cars are becoming increasingly wirelessly connected. Thus, it is important for OCTANE to be able to monitor and test the wireless components integrated into cars.
- *Improve Dynamic Packets and Sequence of Packets.* This will enable the user to transmit dynamic packets and automatically identify potential instances of dynamic packets and sequence of packets.

Regarding the security aspects of the project, our future plans include:

- *Firewalls for Incoming Traffic.* This would help to stop unexpected incoming traffic from accessing an automotive network as described in section 5.1; and
- *ECM Security.* This would prevent unauthorized re-programming and access by unapproved devices and persons as described in section 5.1.
- *Car2X Security.* This would enable OCTANE to monitor and test the car2X networks, which will enable increased understanding vulnerabilities and improve their security.

OCTANE provides a foundation for cyber-physical security research in the automotive space. There are many challenges during the entry into this research field and the testbed is designed to reduce the challenges and ease the entry. OCTANE eases the entry by providing the software tools to analyze automotive networks and a hardware framework for setting up the physical components of the automotive networks. Overall, OCTANE is designed to allow users to learn about automotive networks through the available resources [36] and then proceed to the design and implementation of security solutions for the automotive networks through the implementation of proposed security solutions [27, 32] or newly developed security solutions.

In this paper we present OCTANE, which is an intuitive and flexible software and hardware based testbed that reduces the barrier to entry for both researching automotive security and teaching courses on this topic. Our software tools incorporate easy to use GUI’s that allow for monitoring and transmitting of messages on many of the standardized automotive networking protocols along with a portable XML scheme for defining and sharing proprietary parts of the application layer APIs and protocols that require time consuming reverse-engineering efforts. It is our hope that OCTANE will be useful for implementing and evaluating existing theoretical automotive network solutions in a standardized environment and provide the industry, research and teaching communities with an open source software platform and hardware setup guidelines to enable sharing of information.

[6] In some automobiles, all of the ECUs are reachable from the OBD port (i.e., on the primary CAN network) however on other automobiles, the sub-systems (e.g. window controls, door lock) are in a sub-network such as a secondary sub-network or LIN sub-network which cannot be accessed from the OBD port. These subnetworks can be accessed via direct tapping into the sub-network as described in section 4.2.2

Acknowledgments

This work was support in part by National Science Foundation grant NSF CNS-1205453.

References

[1] Acdelco techconnect - tss - technical support. http://www.acdelcotechconnect.com.

[2] Auto repair reference center. http://www.ebscohost.com/public/auto-repair-reference-center.

[3] Automotive open system architecture. http://www.autosar.org.

[4] Avr-can. https://www.olimex.com/Products/AVR/Development/AVR-CAN/.

[5] Can specification, version 2.0. http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf.

[6] Ecom cable - controller area network (can) to usb adapter. http://www.cancapture.com/ecom.html.

[7] Flexray consortium. http://www.flexray.com/.

[8] Flexray evb. http://store.intrepidcs.com/FlexRay-EVB-p/flexray-evb.htm.

[9] kvaser.com - lin usb. http://www.kvaser.com/zh/products/lin/usb.html.

[10] kvaser.com - usb. http://www.kvaser.com/zh/products/can/usb.html.

[11] Lin subbus. http://www.lin-subbus.de/.

[12] Obdlink sx scan tool - obd interface. http://www.scantool.net/obdlink-sx.html.

[13] Pclink g4 - link engine management systems - plug-in and wire-in aftermarket ecu's. http://www.linkecu.com/support/downloads/pclink-download/PCLinkG4.

[14] Toyota technical information system. https://techinfo.toyota.com.

[15] Tunerpro and tunerpro rt - professional automobile tuning software. http://www.tunerpro.net.

[16] Alexandre Bouard, Benjamin Weyl, and Claudia Eckert. Practical information-flow aware middleware for in-car communication. In *Proceedings of the 2013 ACM workshop on Security, privacy & dependability for cyber vehicles*, pages 3–8. ACM, 2013.

[17] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, Berkeley, CA, USA, 2011. USENIX Association.

[18] S. Corrigan. Introduction to the controller area network (can). Application report, Texas Instruments, July 2008.

[19] U. Drolia, Zhenyan Wang, Y. Pant, and R. Mangharam. Autoplug: An automotive test-bed for electronic controller unit testing and verification. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1187–1192, oct. 2011.

[20] Christopher E Everett and Damon McCoy. Octane: Open car testbed and network experiments bringing cyber-physical security research to researchers and students. In *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*. USENIX.

[21] R. Frank. Under the hood of the bmw x5. *Auto Electronics*, page 18, May/June 2007.

[22] General Motors Company. Gmw gmw3110: General motors local area network enhanced diagnostic test mode specification. http://engineers.ihs.com/.

[23] B. Groza and S. Murvay. Secure broadcast with one-time signatures in controller area networks. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 371–376, aug. 2011.

[24] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks — practical examples and selected short-term countermeasures. In *Proceedings of the 27th international conference on Computer Safety, Reliability, and Security*, pages 235–248, Berlin, Heidelberg, 2008.

[25] T. Hoppe, S. Kiltz, and J. Dittmann. Applying intrusion detection to automotive it-early insights and remaining challenges. *Journal of Information Assurance and Security (JIAS)*, 4(6):226–235, 2009.

[26] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks — practical examples and selected short-term countermeasures. *Reliability Engineering and System Safety*, 96:11–25, 2011.

[27] P. Kleberger, T. Olovsson, and E. Jonsson. Security aspects of the in-vehicle network in the connected car. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 528–533, 2011.

[28] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462, may 2010.

[29] U.E. Larson and D.K. Nilsson. Securing vehicles against cyber attacks. 2008.

[30] G. Lee, H. Oguma, A. Yoshioka, R. Shigetomi, A. Otsuka, and H. Imai. Formally verifiable features in embedded vehicular security systems. In *Vehicular Networking Conference (VNC), 2009 IEEE*, pages 1–7, 2009.

[31] G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, 2002.

[32] K. Lemke, C. Paar, and M. Wolf, editors. *Embedded Security in Cars: Securing Current and Future Automotive IT Applications*. Springer Publishing Company, Incorporated, 1st edition, 2006.

[33] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. Defcon 21, 2013.

[34] MOST Cooperation. Most - home. http://www.mostcooperation.com/home/index.html.

[35] D.K. Nilsson, U.E. Larson, F. Picasso, and E. Jonsson. A first simulation of attacks in the automotive network communications protocol flexray. 53:84–91, 2009.

[36] D. Paret. *Multiplexed Networks for Embedded Systems*. John Wiley and Sons, Ltd, 2007.

[37] Jan Pelzl, Marko Wolf, and Thomas Wollinger. Automotive embedded systems applications and platform embedded security requirements. In *Secure Smart Embedded Devices, Platforms and Applications*, pages 287–309. Springer, 2014.

[38] H. Schweppe and Y. Roudier. Security and privacy for in-vehicle networks. In *Vehicular Communications, Sensing, and Computing (VCSC), 2012 IEEE 1st International Workshop on*, pages 12–17, june 2012.

[39] V. Verendel, D.K. Nilsson, U.E. Larson, and E. Jonsson. An approach to using honeypots in in-vehicle networks. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, 2008.

[40] M. Wolf. *Security Engineering for Vehicular IT Systems*. Vieweg+Teubner, 2009.

[41] M. Wolf, A. Weimerskirch, and C. Paar. Security in automotive bus systems. In *Proceedings of the Workshop on Embedded Security in Cars (escar)'04*, 2004.

[42] M. Wolf, A. Weimerskirch, and T. Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal of Embedded Systems*, 2007.