OCTANE: Open Car Testbed And Network Experiments
Bringing Cyber-Physical Security Research to Researchers and Students

Christopher E. Everett
Department of Computer Science
George Mason University

Damon McCoy
Department of Computer Science
George Mason University

Abstract

Security research and teaching 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 because of the dynamic inputs in real-world environments that cause various interactions of the hardware components and control software within the network. In particular, automotive networks are challenging for security research and teaching 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 Open Car Testbed And Network Experiments (OCTANE), which reduces the barrier of entry into the security research and teaching 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 teaching 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.

1 Introduction

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 introduce the Open Car Testbed And Network Experiments (OCTANE) frame-work to 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.

In particular, OCTANE is composed of both a software package and hardware framework. The software package supports monitoring and transmitting many of the protocols used by automotive manufacturers (e.g., CAN [6], LIN [12]) and includes many features to enable researchers to more easily reverse-engineer proprietary protocols and create shareable XML schemes that encode these proprietary protocols. Also, OCTANE is designed to be modular and easily extended which allows teachers to share class room experiments they create and researchers to add in support for new automotive network protocols and hardware monitoring devices. As part of the hardware framework, we provide recommendations and guidelines on how to setup different hardware configurations to enable complex security research and more simple hardware configurations for courses. The hardware configurations enable the users to test real-world interactions between parts of an automobile network in the lab network setup and test proposed security operations in a real-world automobile environment. OCTANE will be released under an open source license and made freely available. 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

Automotive networks have become more complex in recent years as the number of electronic control units (ECUs) and the number of sub-networks (also referred to as network buses and networks in general) connecting the ECUs have increased [32, 40, 41]. Figure 1 illustrates a simple exemplary automotive network. The automotive network includes a controller area network (CAN) sub-network [6], a FlexRay sub-network [8], a local interconnect network (LIN) sub-network [12], and a media
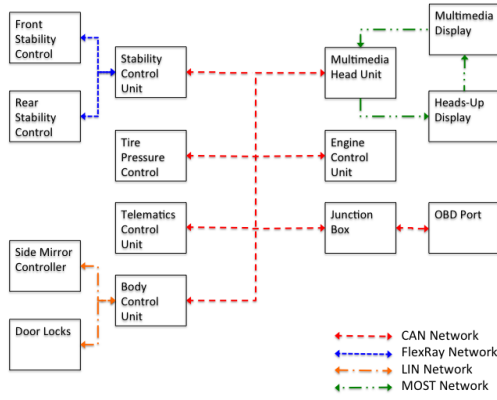
Figure 1: Exemplary automotive network.

oriented systems transport (MOST) sub-network [34] [1]. Each sub-network has certain technical details (e.g., bus speed, trigger type) and advantages with respect to the use in different car models (e.g., fast multimedia transfer, low cost, high reliability for safety functions). These technical details and advantages are not described in this paper but are described in numerous other publications (see, e.g., [36, 40]). However, this paper provides a brief overview of the sub-networks with respect to figure 1.

Turning to figure 1, each sub-network, CAN, FlexRay, LIN, and MOST, is utilized for technical and cost reasons within the automotive network. The CAN sub-network is the backbone of the automotive network and provides an interface for the standardized on-board diagnostic (OBD) port [14], which enables emissions testing and hardware testing. The FlexRay sub-network is for high-speed, safety-critical automotive applications (e.g., stability control, back-up detector). The LIN sub-network is for low-speed, low-bandwidth automotive applications (e.g., side mirror control, door lock/unlock). 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 disadvantages and can be utilized by automotive manufacturers to decrease the overall manufacturing cost while providing the best network performance.

In addition to the control and safety of the automo-bile, the automotive network enables maintenance of the automobile 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 [13] 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.

Turning back to the various sub-networks, each sub-network has an industry accepted standard [6, 8, 12, 34]. In addition to 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., GMLAN [21]). However, these proprietary application layers are generally not publicly available. Instead, 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]. The proprietary application layers developed by the automotive manufacturers provide a wealth of knowledge about the network implementation but generally this knowledge has to be reverse-engineered [3].

3 Related Work

In the automotive security field, a few software packages [27, 29, 35], lab network setups [25, 27, 29, 37] and real-world test setups [18, 29, 37] 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 setup and use by other users. OCTANE is designed to enable users to quickly move from a basic understanding of automotive networks [36] to testing security solutions on automotive networks [4].

The prior work provides an overview of the security issues within automobile networks [25, 27–29] and proposed solutions to the security issues [27, 39]. The proposed solutions include honeypots [30, 38], fire-

---

[1]There are many other automotive networks that are currently implemented in production vehicles (e.g., body electronics area network (BEAN) [24], audio visual communication - local area network (AVC-LAN) [4]). There are also many planned automotive networks that are expected to be implemented in vehicles in the next 5-15 years (e.g., ethernet [23], time-triggered CAN [32]).

[2]The re-programming of ECUs can be done using automotive manufacturer approved tools [1] or through third party tools (e.g., PCLink [15], TunerPro [17]). The re-programming can range from tweaking 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]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.

[4]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.

walls [40], intrusion detection systems [26, 27, 30], encrypted communication [22, 33, 37, 40, 41], ECU authentication [40, 41], and secure communication techniques [31]. Kleberger et al. [28] 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 [27]. 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) provides a software package and a hardware framework for the reverse engineering and testing of automotive networks. OCTANE enables reverse engineering by providing a packet monitor with customized packet identification (e.g., identify door lock packet, identify ECU re-programming request packet) and a customized packet transmitter to verify that the monitored packets operate on 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). The verification that the monitored packets operate on the network enables a researcher to quickly and accurately reverse engineer the proprietary protocols utilized by automotive networks.

OCTANE enables testing of the automotive network by providing the tools to test proposed security solutions on the network (e.g., firewall to stop ECU re-programming from an unauthorized device, firewall to stop non-tire pressure sensor packets from entering through the tire pressure control). For example, OCTANE enables 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 a door lock command that the researcher is attempting to replicate. In another example, the replayed traffic could be a ECU re-programming command that the researcher is attempting to replicate to re-program 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.
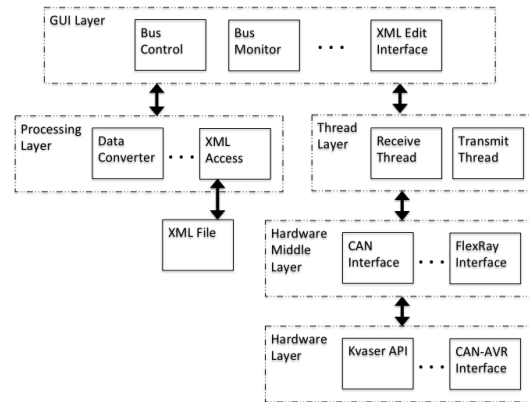

Figure 2: Architecture of software package. The architecture enables the software package to be utilized with additional hardware interfaces by adding the application programming interface to the hardware middle layer without re-coding of the GUI layer or the thread layer.

4.1 Software

Figure 2 illustrates the architecture of the software package. The architecture includes a GUI layer, a processing layer, a thread layer, a hardware middle layer, and a hardware layer. The software package is divided into the layers to enable flexibility and adaptability in the types and quantities of network data being processed. Each layer is briefly discussed below. Following the brief description of each layer, the different components of the software package, adapters, monitor, custom transmit, XML automation, and transmit, are discussed in turn.

The GUI layer provides the GUIs for the software package. Screenshots of some of the GUIs are provided herein (i.e., Figures 3 and 4). Additional GUIs can be quickly and efficiently added to the software package for new features (e.g., ECU re-programming). The GUI layer connects to the processing layer and the thread layer. The processing layer performs the processing of the data for the software package (e.g., convert user input into CAN message format), logging of messages, and access to XML files for the XML automation described in section 4.1.4. The processing layer off-loads the data processing from the GUI layer to avoid stalling the GUIs.

The thread layer provides threading mechanisms for the receiving and transmitting of data through the appropriate hardware. 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.
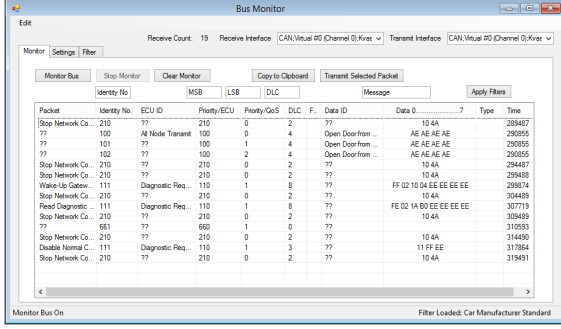
3

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 a packet for transmission and identification.

The hardware middle layer is utilized to obfuscate 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 GUI layer, thread layer, or processing layer. The hardware middle layer calls the appropriate hardware API in the hardware layer. The hardware layer is the API provided for by the hardware device manufacturer or coded for the particular hardware device. 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 software package's bus control interface (not shown) controls the operation of the automotive network adapters (i.e., CAN [5, 7, 11], LIN [10], FlexRay [9]). The software package enables a variety of different hardware controllers to be utilized. Other hardware controllers can quickly and efficiently be added to the architecture as described above in section 4.1. The bus control interface enables a user to turn on or off the various hardware controllers for use by the software package and control configuration options for each hardware controller [5]. Overall, the control interface provides user control over the various hardware controllers.

4.1.2 Monitor

Figure 3 illustrates the software package's bus monitor interface. The bus monitor interface outputs the received packets and allows for transmission of packets back to

---

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

a network. The bus monitor displays the received packets from the selected *Receive Interface*. The bus monitor also enables the selection of a filter . 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 includes packets, ECUs, and messages. 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 a 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, 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.

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 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, bit priority, filter) that is not described herein due to space limitations.
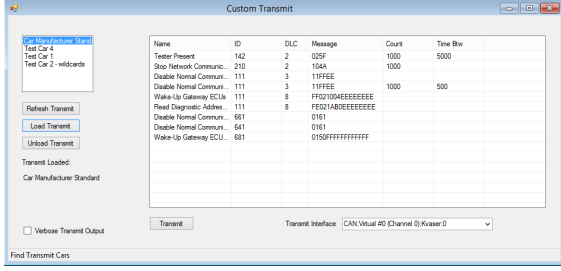
Figure 4: Screenshot of the custom transmit interface of the software package. The custom transmit interface enables the selection of a filter for a particular car or car type and transmission of packets associated with the selected car or car type.

4.1.3 Custom Transmit

Figure 4 illustrates the package's 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 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 CAN messages are illustrated in the screenshot of figure 4, any of the other network protocols (e.g., LIN, FlexRay) can be utilized by the custom transmission 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.

4.1.4 XML Automation

The packages's XML editing interface (not shown) 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, or modify packets, messages, and ECU IDs 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.

As a future improvement of the software, the XML automation will be extended to accept the following XML extensions:

- *Wildcards* (e.g., ID = 64?; ID = 1*; Message = AE1[0-5]);
- *Packet Sequences* (e.g., packet A followed by packet B);
- *Packet responses* (e.g., response with packet D upon receipt of packet C);
- *Packet Subroutines* with sequences and responses (e.g., packets A and B and then response with packet D upon receipt of packet C); and
- *Calculated Packet Responses* based on received packet (e.g., response with packet F upon receipt of packet E, packet F includes $ID_F = ID_E + 1$ and $message_F$ calculated with a pre-defined function based on the $message_E$).

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

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 automo-
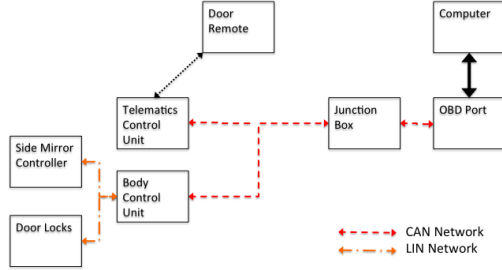
Figure 5: Exemplary automotive network for a lab network setup. A graduate student can utilize the automotive network to configure proposed security solutions for testing.

tive 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.

4.2.1 Lab Network Setup

Figure 5 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) cer-

tain BMW models included the three automotive network types (e.g., research through web searches [20] and review of electronic wiring diagrams); (d) Kvaser supported CAN [11] and LIN [10] network types and Intrepid Control Systems supported FlexRay network type [9]; and (e) budget was large enough to support a large network. Figure 5 illustrates part of the lab network that we assembled [6]. The computer in figure 5 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, 16]. 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 Teaching Opportunities

For *research opportunities*, figure 6 illustrates an exemplary automotive network with exemplary security devices. The security devices could be utilized as security solutions for the automotive networks to prevent and stop unauthorized access to the automotive network. The following are some of the security solutions that could be tested using the testbed:

---

[6]Figure 5 does not show the FlexRay sub-network for the dynamic stability control
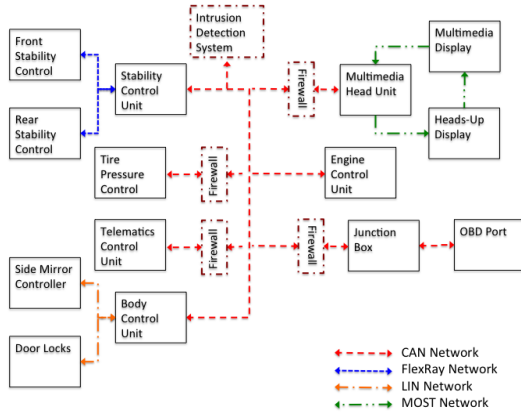
Figure 6: Exemplary firewalls and intrusion detection system on an automotive network. The firewalls are positioned at every entry point on the automotive network. The intrusion detection system is positioned on the core CAN network.

- *Firewall* to prevent transmission of unauthorized packets into an automotive network [40];
- *Intrusion Detection System* to detect anomalies on an automotive network [26, 27, 30];
- *Packet Encryption* to protect the data on an automotive network from easy sniffing and packet insertion [22, 33, 37, 40, 41];
- *ECU Authentication* to prevent unauthorized ECUs (e.g., ECU inserted by a malicious actor) from interacting with an automotive network [40, 41]; and
- *ECM Security* to detect tampering of ECUs along with authentication tampering [31].

For *teaching opportunities*, figure 5 illustrates an exemplary automotive network for laboratory experiments and testing of proposed security solutions. There are also many teaching 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 teaching uses of the testbed related to automobile security testing:

- *Undergraduate Laboratory Security Exercise* to sniff network activity and attempt to take-over the network (e.g., control the side mirror, lock/unlock the doors);
- *Undergraduate Laboratory Embedded Programming Exercise* to program an AVR-CAN controller to receive and transmit packets on a CAN network [5]; and
- *Graduate Security Testing Exercise* to implement an intrusion detection system on an AVR-CAN controller for a CAN network [5].

6 Learning Points

Automotive security researchers should be aware of the following issues to avoid pitfalls that we encountered during the development of OCTANE.

- *Finding Application Layer* information for a particular automobile manufacturer is challenging as described in section 2;
- *Finding Automotive Parts* is challenging without an actual vehicle. The automotive parts infrastructure is designed for repair facilities with access to the actual vehicle being repaired; and
- *Electrical Diagrams* are important for identifying the ECUs that are accessible via the OBD port for the particular automobile under testing [7] as described in section 4.2.2.

7 Future Work and Conclusion

Currently, we have plans for several extensions to the software package along with security implementations for testing for automotive networks. The following software package extensions are planned:

- *XML Automation Extensions* to enable robust reverse engineering and testing through interactive packet responses and manipulation as described in section 4.1.4; and
- *Remote Access* to enable researchers and students to access OCTANE from locations remote from the lab network setup and the real-world automobiles.

The following security implementations and testing of the implementations are planned:

- *Firewalls for Incoming Traffic* to prevent unexpected incoming traffic from obtaining access to an automotive network as described in section 5; and
- *ECM Security* to prevent re-programming and access by unauthorized devices and persons without the owner's permission as described in section 5.

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 pro-

---

[7]For some automobiles, the entire spectrum of ECUs are accessible through the OBD port (i.e., on the primary CAN network) and for other automobiles, the sub-systems (e.g., door lock, window controls) are in a sub-network (e.g., LIN sub-network, a secondary CAN sub-network) and are not visible via the OBD port (i.e., direct tapping into the sub-network is required).

tools 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 research and teaching community with an open source software platform and hardware setup guidelines to enable sharing of information.

Acknowledgments

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] Avc-lan - elinux.org. http://elinux.org/AVC-LAN.

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

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

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

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

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

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

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

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

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

[14] On-board diagnostics (obd) - us epa. http://www.epa.gov/obd/.

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

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

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

[18] 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.

[19] U. Drolia, Zhenyan Wang, Y. Pant, and R. Mangharam. Auto-plug: 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] R. Frank. Under the hood of the bmw x5. *Auto Electronics*, page 18, May/June 2007.

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

[22] 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.

[23] C. Hammerschmidt. Beyond flexray: Bmw airs ethernet plans. *EE Times*, June 2010.

[24] H. Honda, S. Uehara, K. Sakai, T. Akatsuka, and S. Akiyama. Body electronics area network (bean). Technical report, SAE Technical Paper 970297, 1997.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

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

[33] 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.

[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] 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.

[38] 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.

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

[40] 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.

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