DOCTOR OF PHILOSOPHY

A Fuzz Testing Methodology for Cyber-security Assurance of the Automotive CAN Bus

Fowler, Daniel S.

Award date:
2019

Awarding institution:
Coventry University

A Fuzz Testing Methodology for Cyber-security Assurance of the Automotive CAN Bus

by

Daniel S. Fowler

May 2019

HORIBA MIRA Limited

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

Content removed on data protection grounds.

Certificate of Ethical Approval

Applicant:

Daniel Fowler

Project Title:

Automotive cyber-security - Testing simulated and experimental vehicle CAN systems for resilience using a fuzz test methodology

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

10 November 2017

Project Reference Number:

P63333

To my wife, Julie, and my sons, Joel and Luke, for their support and continuous giving to all.
To all my family and friends for their love.

Acknowledgements

This research programme and thesis was possible due to the support of many individuals at Coventry University and HORIBA MIRA Limited. I need to give special thanks to Dr. Jeremy Bryans, my Director of Studies, for his excellent doctoral supervision. His guidance, suggestions, and insights were very valuable. Likewise, Professor Siraj Shaikh, as the second supervisor, provided extremely constructive comments and knowledge.

At HORIBA MIRA professionalism and helpfulness was provided throughout the programme. Paul Wooderson, Dr. Madeline Cheah and Dr. Anthony Baxendale were especially supportive. All HORIBA MIRA staff were approachable, including Dr. Alastair Ruddle, Chris Mellors and Pierre Papon, who provided help at various times.

The Coventry University staff at the Faculty of Engineering, Environment and Computing, the Doctoral College and the Institute for Future Transport and Cities aided in the completion of the programme. This includes Kevin Vincent, Prof. Mike Blundell, Dr. Hoang Nga Nguyen and Prof. Andrew Parkes.

The need for security testing of vehicles is part of an ongoing research collaboration between HORIBA MIRA, who are a transportation and vehicle test, design and consultancy company, and Coventry University. They both engage in research within many areas of automotive engineering and are in central England, at the heart of the United Kingdom motor manufacturing industry. Vehicle cyber-security is one of many joint areas of interest and is one of the topics for development within the joint Centre for Connected and Autonomous Automotive Research (CCAAR).

Abstract

The cyber-physical vehicle is one of the underpinnings of modern society, however, if a vehicle's design is faulty it carries a risk of injury to the occupants and the public. It has been demonstrated that intelligent agents can penetrate connected cars via cyber attacks and cause an unsafe state. The possibility of a cyber attack means that cyber-security testing should be performed to maintain assurance in vehicle systems. However, vehicle cyber-security testing methods are immature.

Fuzz testing is a dynamic testing method for software-based systems. Automotive industry guidelines regard it as a component in the security testing process of cyber-physical systems. The hypothesis is that fuzz testing can be used over a system's lifecycle as part of the design and maintenance process for cyber-security. However, there are few evidential results on the application of fuzz testing to the automotive field. This applied research provides one of, if not the first, detailed contribution on fuzz testing automotive systems.

A tool to performing vehicle fuzz testing, called a fuzzer, was constructed using an iterative methodology to enable experimental observations on automotive systems and components. Using the dedicated fuzzer empirical results were gathered. The target for the fuzz testing was a lab vehicle's Electronic Control Units, accessed via a common intra-vehicular communications bus, the Controller Area Network. The results demonstrate that fuzz testing is indeed beneficial to the design of vehicle systems and can contribute to system assurance. Furthermore, the construction of the fuzzer and its application to vehicle systems has contributed a method for the development of additional security tests for the automotive field. However, the technology within a vehicle system is a challenge for cyber-security testing, this includes the cyber-physical aspects of a car and the symbiotic interaction of a vehicle's computational elements. There remains significant research work required before fuzz testing becomes commonly integrated into test procedures for all the systems within connected cars.

Table of contents

List of figures

Unless otherwise stated, all diagrams, photographs and figures are the work of the author. Where a diagram has been reinterpreted from its original source, that source is referenced.

List of tables

Unless otherwise stated, all tabular content is the work of the author.

Nomenclature

Acronyms / Abbreviations

ACL	Access Control Lists, these are used by firewalls and other computer security systems to provide security to various types of resources. A request to access a resource is checked against a list of users and groups who have permission for the requested access level.
ADAS	Advanced Driver Assistance Systems, vehicles that provide advanced driving aids, such as collision avoidance and lane-keeping.
API	Application Programming Interface, a method for software-based systems to provide functionality to other programs. The functionality is provided by a library of documented routines which make up the API.
AT	Attention, a serial communications protocol called the Hayes command set that originated in dial-up modems. AT commands are used by ELM based OBD devices
CAN	Controller Area Network, a common data transmission network between computational units (ECUs) within vehicles.
CAV	Connected and autonomous vehicle, a term applied to the development and deployment of highly computerised and sensor-enabled vehicles. Such vehicles are wirelessly connected to remote services via the Internet and may provide a degree of autonomous driving, from basic lane keeping to full driverless functionality.
CCAAR	The Centre for Connected and Autonomous Automotive Research is a collaboration between HORIBA MIRA Limited and Coventry University to investigate the technologies required for future vehicles.
COMM	a serial communications port on a PC, these were originally RS-232 ports but are now often virtual ports using USB interfaces.
CPS	Cyber-Physical System, a combined physical, computational and networked system. The physical components are monitored or controlled by computational elements, often inter-connected, a CPS can be connected to the Internet.

CPU Central Processing Unit, the brain of a computer system which executes program code read from the computer's memory.

CRC Cyclic Redundancy Check, an algorithm used to enable detection of bit errors in a data stream, often used in computer systems when data is moved between locations. For example, over a network or from computer disk to memory. CAN uses CRC as part of its error handling.

DCE Data Communications Equipment, in serial communications via RS-232 (includes emulated RS-232 via USB) the term DCE to refers to the equipment at the end the connection, i.e. the device sending the commands and data from a computer or terminal.

DLC Data Length Code, Part of the CAN protocol and a field in a CAN packet. Sets the number of bytes transmitted in the packet.

DLL Dynamic Link Library, a code library that contains specific functionality that is loaded by the main Windows program when that functionality is required.

DoS Denial of Service, an attack against a system that is performed by flooding the system with data at a rate that the system is unable to handle, causing the system to slow down and possibly halt.

DSRM Design Science Research Methodology, a research method that provides a framework to enable the considered design of an artefact in information systems domains, see Chapter 4.

DTC Diagnostic Trouble Code, these are read from a vehicle, normally via the OBD port, and provide a code that can be translated into a known fault or vehicle parameter. Often used for fault finding in vehicles.

DTE Data Terminal Equipment, serial communications via RS-232 (included emulated RS-232 via USB) the term DTE to refers to the computer side of the connection, i.e. the device acting as a serial terminal.

DUT Device Under Test, in engineering fields this is what a component is called when being subjected to a testing regime. See also SUT and ToE.

ECU Electronic Control Unit, an embedded computing device used to control an aspect of vehicular functionality.

ELM The ELM company produces a CAN interface MCU that is popular in OBD dongles that plug into cars. The ELM protocol is commonly used in apps that read vehicle data from the OBD port. The ELM chips have been widely counterfeited.

escar Embedded Security in Cars, one of the first (2003) regular conferences to address the automotive cyber-security field. Although a commercial conference, it is well established and attracts interesting presentations.

EVITA E-safety Vehicle Intrusion proTected Applications, this European project ran from 2008 to 2011. It looked at vehicle security issues. EVITA outputs are often cited in other works in the automotive cyber-security field.

GM General Motors, one of the world's oldest vehicle companies, and the World's largest until 2007. GM was one of the first companies to experiment with computers to control vehicle functions.

HARA Hazard Analysis and Risk Assessment, the method defined in ISO 26262 to assess the safety goals of a system and determine areas that require mitigation to avoid unreasonable risk

HEAVENS HEAling Vulnerabilities to ENhance Software Security and Safety was a project involving Volvo and Chalmers University, investigating methods and tools for security testing vehicle systems.

hex hexadecimal, the base 16 number system is used in software to simplify the display of binary and byte-sized data, four binary bits (half a byte) is displayed as a single character, 0 to F.

HIL Hardware-in-the-Loop, a way to test the hardware sub-components of a system by connecting the component to a test rig that partially or fully represents the final system.

HSM Hardware Security Module, implements cryptographic processes into silicon chips to increase the performance of encryption and decryption algorithms. Similar chips can now be found in cell phones, PCs and newer automotive MCUs.

id Identifier, an identifier is a unique sequence of characters given to an entity in a system to distinguish it from other entities. For example, in this research, each CAN packet has an id to allow the software to determine the meaning of the data in the CAN packet. In CAN the id is called the arbitration field.

IDE Integrated Development Environment, a software package that is used for part, or all, of the work cycle in the production of new software. It commonly consists of a source code editor, one or more compilers and code linkers, and it usually has code debugging capabilities. Other tools may be present, for example, code packaging for installation.

IDS Intrusion Detection System, performs analyse of network and system behaviour looking for anomalous activity that may signal a system compromise.

IEC International Electrotechnical Commission, an international body that publishes global electrical and electronic standards. They work with ISO on security standards, for example ISO/IEC 15408.

IO Input/Output, the connections to and from a computer or microcontroller to read and write data from and to the physical world.

IoT Internet of Things, widespread data gathering and analysis, using miniature sensor-enabled and Internet-connected computing devices. This provides a high volume of granular information (big data). The resultant information is used to improve the provision of services.

IPR Intellectual Property Rights, the legal rights that an individual, group, or organisation holds over an object, artefact, or original output created by their work or business.

ISO International Organization for Standardization, a global body that publishes standards in many domains, including automotive hardware and software standards and security and cryptography standards.

IT Information Technology, the umbrella term that covers the computer hardware and software used by organisations to automate their office, production and service functions. Whilst the computerised office was the first application of information technology it now encompasses all business, industrial and consumer areas.

JTAG Joint Test Action Group, an interface standard for on-board debugger access to CPU and other circuit silicon. Direct access to the CPU can circumvent security mechanisms.

MAC Message Authentication Code, a mac is an additional element added to a message or data packet that can be used to help ensure the validity of the message or data. MACs use cryptography to help preserve data integrity and authenticity. Several MAC schemes have been proposed for CAN.

MCU Microcontroller, a CPU that incorporates memory and IO for use in embedded computing applications. Small and powerful MCUs are behind the growth in IoT.

MIRA Motor Industry Research Association, originally the UK's vehicle research centre it became a limited company in 2001 as MIRA Ltd., and was subsequently purchased by the Japanese company HORIBA in 2015 to become HORIBA MIRA Ltd., a transportation test and design consultancy company.

MITM Man-in-the-Middle, a security issue where an adversary inserts themselves in between communicating parties. They eavesdrop and/or intercept and modify the communications for their gain, usually without the victims being aware.

MOST Media Oriented Systems Transport, a vehicle communications ring bus used for multimedia data transmission within a vehicle.

OBU On-board Unit, another term for an embedded computing device fitted to a vehicle, see ECU.

ODB On-Board Diagnostics, it is a requirement in many countries for a vehicle to have a port that can be used to read vehicle emissions data, several SAE specifications detail the OBD technical design and operation.

OS Operating System, the software that runs a computer, handling all the hardware, task scheduling and loading of programs, examples include Windows, Android, Linux and QNX.

PC Personal Computer, usually a normal desktop computer or workstation running a popular OS (Linux, Windows, macOS), can also refer to a laptop computer

PID Parameter Identifier, this is a number sent to an ECU during vehicles diagnostics to tell the ECU to transmit certain vehicle parameters, usually related to engine emissions data or ECU error codes. There are standard ids specified in SAE J1979 and manufacturers will implement custom ids for their diagnostics.

QNX Quantum UNIX, a UNIX like microkernel OS widely used in embedded applications, including vehicles.

R&D Research and Development, the process of producing new products or services, usually involving innovation and new design.

RS-232 A long-standing serial data communications, or COM, interface, RS-232 devices linked to a computer are often converted to USB ports which then emulate the older RS-232 standard.

SAE Society of Automotive Engineers, started as an organisation of automotive engineers, SAE International covers many forms of transportation and publishes many global standards in their field of expertise.

SBC Single Board Computer, a small computer with all the components on a single PCB, requiring only a power supply for operation. A well-known example is the Raspberry Pi. Arduino SBCs are another popular format.

SDL Secure Development Lifecycle, a software development process originating from Microsoft. It promotes a structured methodology for software development that includes designing in security from the outset.

SIL Software-in-the-Loop, a way to test a software component of a system by executing the component in a simulator that partially or fully represents the final system.

SMMT Society of Motor Manufacturers and Traders, the trade body for the UK car industry.

SPP Serial Port Profile, a fundamental Bluetooth protocol to emulate serial data connections between devices.

SUT System Under Test, in engineering fields this is what a system (consisting of multiple interconnected components) is called when being subjected to a testing regime. See also DUT and ToE.

ToE Target of Evaluation, in ISO/IEC 15408 (Common Criteria) this is the system or device being security assessed. See also SUT and DUT.

UDS Unified Diagnostics Services, a protocol (ISO 14229-1) to communicate with ECUs for fault diagnostics and firmware updates.

UI User Interface, this is the screens and input controls of a program, used for interaction with the software. The term Graphical User Interface, or GUI, is also used. A Windows or Apple Mac computer has a GUI.

USB Universal Serial Bus, a high speed serial communications bus that provides a common digital data IO and power connection to most modern computers, cell phones and devices. This ubiquitous interface, present in modern vehicles, has been shown to have security vulnerabilities.

V2D Vehicle-to-device, a vehicles wireless connection to a driver or passenger device to provide enhanced services, example include telephone integration via Bluetooth and wireless hotspots

V2I Vehicle-to-infrastructure, a vehicles wireless communication to city and road infrastructure for value-added services, examples include traffic information and speed warnings

V2V Vehicle-to-vehicle, wireless communications between vehicles to provide enhanced services, examples include collision pre-warning and platooning.

V2X Vehicle-to-X, a term to describe a vehicle's multiple connections to multiple services, covering V2V, V2I and V2D.

XCP Universal Measurement and Calibration Protocol, a protocol used to connect calibration equipment with ECUs, allowing access to measurements and calibration data, supports flash memory programming.

Chapter 1

Introduction

Hello, I'm Johnnycab. Where can I take you tonight?

Johnnycab, Total Recall (Film, 1990)

Human imagination often considers machines that can aid us before the technology is available to build them. Today's pocket supercomputers (smartphones), voice-controlled digital assistants (e.g. Alexa and Siri), and self-driving pods (e.g. Heathrow ULTra), were once science fiction. However, as computers, sensors and batteries become more powerful, smaller and efficient, more science fiction ideas enter the marketplace. For example, fully autonomous vehicles (our very own Johnnycabs) are under development. Advances in mechanical, materials, battery, electronics, computer and other technologies make possible these high impact machines. However, two technologies are of particular importance. Firstly, software, a computer cannot do anything useful without its programming. Secondly, wireless connectivity, the new technologies use untethered communications for command-and-control and interaction with remote services. Software and connectivity not only allow systems to provide their useful services, but they can also be a security issue. This is due to the possible presence of undiscovered software vulnerabilities hidden in all the code running in a machine.

1.1 The threat from cyber crime

New technology is not only beneficial to its users, but it is also another target for criminal intentions. Cyber crime has been in existence for almost as long as computers themselves [1]. Our hyper-connected and computer-based world operates under a duality. On the one hand, billions of devices successfully perform their correct functions every day. On the other hand news stories of successful high profile, computer hacking events occur with regularity[1]. Organisations and governments are engaged in the fight against computer-based crime performed via the Internet. This now includes car manufacturers, as their products are now part of the hyper-connected world and have been shown to

[1] https://www.hackmageddon.com/

be hackable [2]. Car manufacturers are having to adapt their product design and testing regimes to address the need for cyber-security resilience. (In this research, resilience is the ability for a vehicle's systems to resist a cyber attack and continue to function safely. Cyber resilience is discussed further in the literature review in Chapter 2.)

Cyber crime is an arms race between scientists and engineers, who build and need to defend their new technologies, and the criminals that attack the new technology for illicit gain. This cyber arms race will always continue, therefore, any new tools and techniques that can aid in producing a more secure system design are beneficial. Fortunately, vehicle manufacturers and their suppliers are not starting afresh. The knowledge gained from the many decades defending and testing traditional Information Technology (IT) systems can be adapted to vehicular systems. There is also the existing experience in performing functional testing on vehicles, where processes and procedures are rigorously applied. Such rigour can be invoked for cyber-security testing regimes. However, automotive cyber-security is only a recent field of study, and although it is being actively researched across the globe, the practical knowledge that can be deployed by manufacturers and testing companies is currently limited.

This research investigates how the systems in the many millions of connected vehicles being produced can be tested to improve their security resilience. The research achieves this with a new application of an existing technique known as fuzz testing. Fuzz testing is a dynamic analysis test method using randomised data. Fuzz testing is investigated for its application as a security test for vehicle systems.

1.2 Establishing the research aim

This research began with a high-level objective, which was to address the need to perform cyber-security testing on vehicles. To establish the main topic of this research, i.e. to establish the specific research question, that high-level objective needed to be understood by assessing the question:

What does it mean to cyber-security test vehicles?

Answering this question requires an understanding of the nature of cyber-security and its application to vehicles, and then understanding the problem of testing. These aspects were considered during the literature review, which discusses the complexity and challenges faced by vehicle manufacturers in testing their products for cyber-security resilience. Engineering a vehicle is a complex and challenging task, in [3] a vehicle's development stages are given as:

- Vehicle Concept
- Styling
- Engineering Start
- Integration
- Validation
- Pilot

- Production

Furthermore, when engineering the vehicle components, the hardware and software development occurs simultaneously [3]. The development process follows what is known as the *V-model* [3], [4] (or Vee-model), see Figure 1.1.

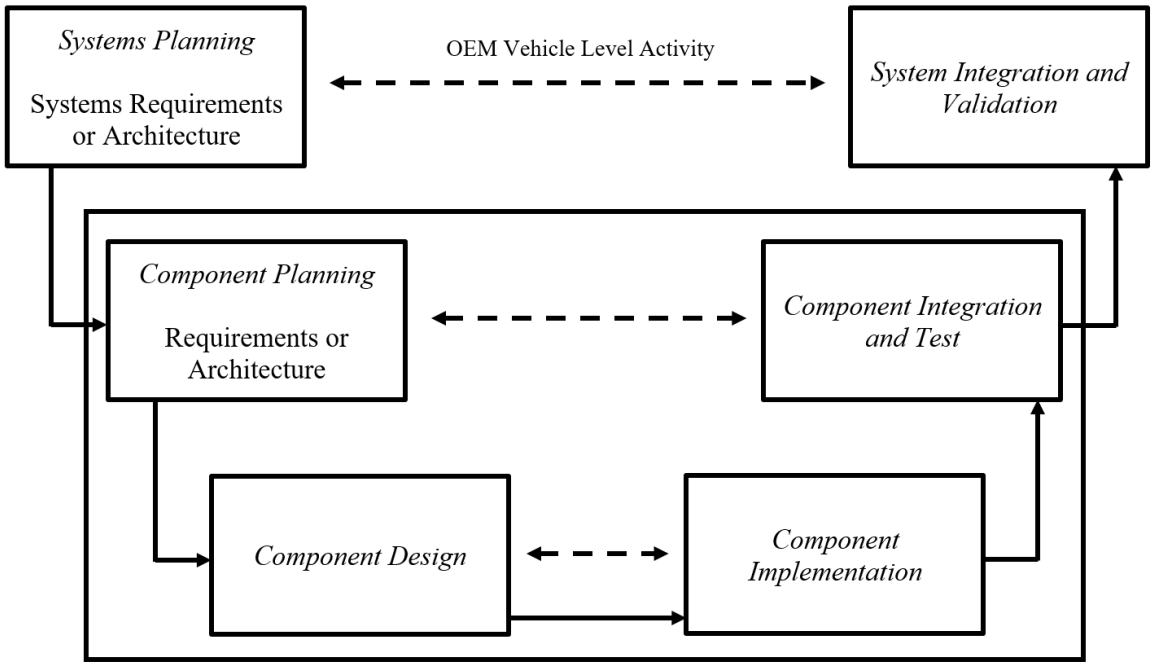


Fig. 1.1 The V-model for vehicle systems development, drawn from [3], the dashed lines represent verification and validation of phases

The nature of the V-model process, and complexity of vehicle systems engineering, has long required the provision of specialised equipment for pre-production and developmental testing. One research possibility was to use such test equipment for security testing early in the vehicle systems design process.

1.2.1 Hardware-in-the-loop and software-in-the-loop

Commercial test equipment companies provide systems to enable unit and integration testing of vehicular electronic systems [5]. This mimics the testing and development of systems and subsystems performed in the space and aerospace industries. These test and design tools are referred to as hardware-in-the-loop (HIL) and/or software-in-the-loop (SIL). A HIL takes the place of a physical component in a system design to allow for detailed debugging. A SIL can emulate several components, providing a full or partial virtual system to accelerate overall system development. Figure 1.2 shows the software for a Vector[2] HIL/SIL system emulating a vehicle's network system. The Vector HIL/SIL system was used in this research.

[2]https://www.vector.com/

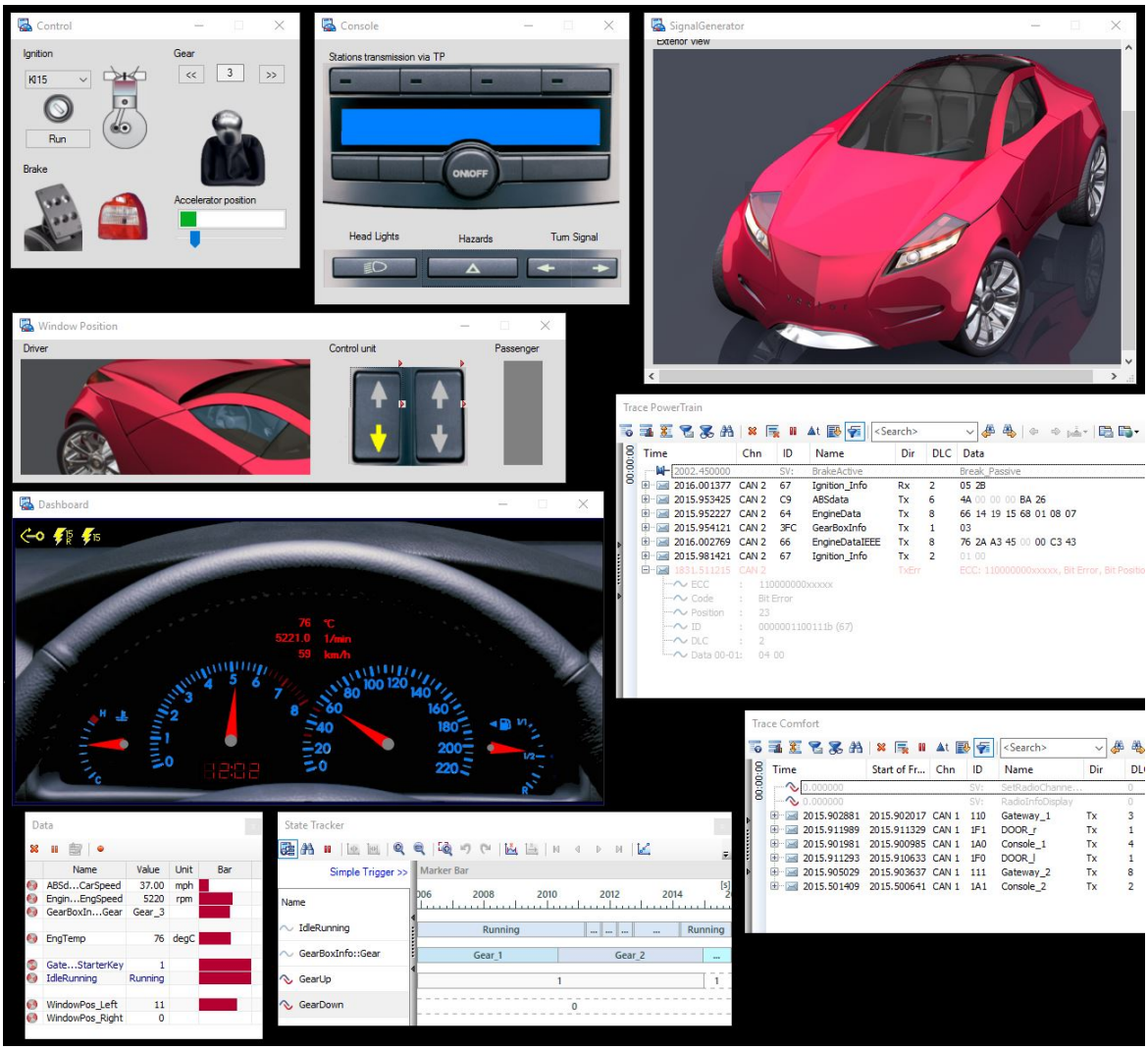


Fig. 1.2 A HIL/SIL system from Vector Informatik GmbH was used during this research, here their CANoe software is executing a vehicle simulation

1.2.2 CAN is a common vehicle component interface and network

The technology behind the computational networked elements in a modern vehicle are discussed in the literature review, with further expansion in Appendix B. A computer in a modern vehicle is referred to as an Electronic Control Unit (ECU). An ECU is an embedded computer designed to perform a specific computational task. ECUs are often interconnected with a data bus called the Controller Area Network (CAN). The ubiquity of the CAN bus and its presence on vehicle ECUs made it a suitable target for this research. The Vector HIL/SIL system is able to simulate multiple ECUs and CAN busses.

1.2.3 Testing for secure design

To support the concept of designing security into a system, a HIL/SIL system can be used for cyber-security testing early in the research and development (R&D) process of a new component or vehicle. This would mean that automotive engineers will have to consider security from the outset. For this research a HIL/SIL system was used to establish a cyber-security testbed, enabling an experiment to validate the idea of using such testbeds for security testing [6], see Chapter 4. Whilst the experiment demonstrated that cyber-security testing can be performed with a HIL/SIL development system, the contribution to knowledge was limited. However, the presentation of the results at a conference did encourage the consideration of dynamic test methods for vehicular cyber-security testing [7]. Examination of the literature showed a lack of publications on a dynamic test method, fuzz testing, for security testing automotive systems.

1.2.4 The research motivation

Fuzz testing is a dynamic test procedure for systems. A dynamic test is one that is performed whilst the system is in operation. Fuzz testing uses a large amount of randomly generated or mutated input data to evoke a system failure. Causing a system failure may reveal an exploitable vulnerability, breaking system security. The literature review revealed that there is no substantial detail available in applying fuzz testing to automotive systems, providing an opportunity to formulate a research question.

A lack of knowledge in automotive fuzz testing motivated this research, alongside the general automotive cyber-security issues discussed in the literature review. Fuzz testing has been successfully used in general computer systems testing [8] and is included as part of the Microsoft Secure Development Lifecycle (SDL) [3]. Thus, the initial high-level objective to address the need to perform cyber-security testing on vehicles became focused on the provision of a practical fuzz testing method for automotive systems. This new method is intended for use as an aid for testing the resilience of a vehicle system to cyber attack. The results from performing the fuzz testing may then be used to improve system design. The overall aim of fuzz testing is to raise the level of cyber-security assurance the manufacturers have in the software running in their vehicles. The issue of cyber-security assurance

is discussed in the literature review, however, in the context of this research, it can be summarised as the confidence that one has in a system's ability to respond to a cyber attack. With the above motivation in mind, the research question can be stated.

1.2.5 The research question

The research question examines fuzz testing as applied to automotive systems. Whilst fuzz testing has been used successfully in non-automotive domains, is that success transferable to automotive engineering? In particular, the use of fuzz testing is to test the security resilience of the automotive system, therefore:

> **RQ** - How can the results from fuzz testing contribute to vehicle system cyber-security assurance?

The hypothesis is that fuzz testing will reveal system weaknesses, and thus indicate that the system design needs to be changed to remove or reduce those weaknesses. This will be examined using physical experiments. However, manufacturers will require a usefully deployable test method. This means the test method should be cost-effective, in time as well as money. The initial and ongoing overheads to set-up and execute the fuzz testing should not be onerous.

Factors for a practical automotive fuzzer tool

The literature review discusses the limitations of using existing tools for fuzz testing automotive systems. To aid research into automotive fuzz testing the supporting tools should be practical to deploy, a point made by HORIBA MIRA engineers. The factors for practicability include:

- The technology required to use the fuzz testing tool.
- The ease of deployment of the tool and setting up a fuzz testing session.
- The ability to fuzz the vehicle technology that is targeted (in this research it is the CAN bus).
- The performance characteristics of the tool (the rate of the data generation and data processing capabilities).

Furthermore, any fuzz testing performed should provide test results that can be acted upon, otherwise, the testing is a waste of resources. This means that the fuzz testing must add value to the testing processes. These factors were in mind when considering this research program's aims and objectives.

The aims to address the research question

The initial identification of a knowledge gap in the testing of vehicular cyber-security was the first high-level aim of this research. The knowledge gap is in the application of fuzz testing to automotive systems development (see the literature review), and allowed for the formulation of further research aims:

1. The first aim of this research was to assess the potential to use fuzz testing as a component in automotive cyber-security testing.
2. The second aim of this research is to provide a practical, based on the list of factors above, fuzz testing method for use in automotive engineering.
3. The final aim is to use this applied research to contribute practical automotive fuzz testing knowledge.

These research aims provide a focus for the objectives of this research.

The objectives of this research

The above aims are achieved via this research program's objectives:

1. The first objective is to test the hypothesis that fuzz testing will aid vehicle system resilience, this will be achieved via fuzz testing experiments on vehicle systems and components.
2. To achieve the first objective, the second objective is to develop and evaluate, to meet the factors previously listed, a practical prototype automotive fuzzer (a fuzz testing tool).
3. The final objective is to evaluate the experience of using the fuzzer. This is to contribute to the automotive cyber-security field and provide future researchers and automotive engineers access to automotive fuzz testing knowledge.

For the first two objectives, the experiments and the tool development, the methodology used during this research is iterative. The methodology is discussed in Chapter 3.

1.3 This research program's contributions

In performing the applied research to achieve the objectives and meet the aims several contributions are provided:

1. The first contribution is to provide a literature review of the newly emerged field of automotive cyber-security, and to show that automotive fuzz testing has not been examined in detail. The found gap provides the basis for the research question.

2. The second contribution is to design, develop and deploy a prototype automotive fuzz testing tool targeted at the CAN bus. The new CAN fuzzer tool allowed for automotive fuzz testing experiments to be performed. This allowed for the collection of empirical evidence to aid in answering the research question.

3. The third contribution is to demonstrate that fuzz testing can be used to test the resilience of a vehicle system and vehicle components. The experimental results were evaluated to answer the research question.

4. In answering the research question other issues on fuzz testing automotive systems were raised. The fourth contribution is to provide a list of challenges to overcome by further research.

5. In performing this research program, a method was derived that can be used to develop other automotive security tests, that method is the fifth contribution.

6. A sixth contribution is a method to address the problem of combinatorial explosion in fuzz testing CAN. A method is provided to aid the reduction in the time involved in fuzz testing the CAN bus.

7. An additional seventh contribution was made, whilst not related to the main topic of fuzz testing, it was observed during this research. In Appendix C a potential vehicle systems cyber attack method is provided via variation of a CAN device's configured bit rate.

The experimental results will show that the prototype CAN fuzzer functions as required. It is a useful tool for fuzz testing automotive systems. It answers the research question by showing that the assurance of vehicle software can be improved with the aid of fuzz testing. This thesis provides, probably the first, detailed study on the problems and methods in fuzz testing automotive CAN-based systems and components.

1.4 Overview of the thesis

The literature review follows this introduction in Chapter 2. It begins by summarising the literature gathering method. Then the review discusses the following areas:

- The meaning of software assurance and security properties in relation to computational systems. Not only does this help understand the requirements of systems security, but it is also intended to familiarise the reader, especially those who are new to cyber-security, with the security concepts and terminology used in this research.

- There is a discussion on the emergence of the automotive cyber-security field (with some further history in Appendix A).

- A description of the underlying technology of the vehicle systems that needs to be protected is provided.
- There is a discussion on the problem of testing vehicle systems for cyber-security resilience.
- Any existing work on the application of fuzz testing to the automotive field is covered.

To provide enough technical detail on the vehicle technology used in this research, but not overburden the main review chapter, some of the technology descriptions are moved to Appendix B. The review chapter then finishes with the inferences drawn from the literature.

Following the literature review, Chapter 3 has the methodologies used in this research. Firstly, the framework to guide the overall research; the Design Science Research Methodology (DSRM) [9] applies to an information systems research programme. Secondly, a research method was developed from the practical work performed; this six-step method is suitable for developing automotive cyber-security testing tooling, methods and experiments.

The derived research method was developed from a set of experiments. Each experiment's specific applicable method is described at the beginning of the chapter covering the experiment. The individual methods are framed in terms of the DSRM and provide additional information on the specific tools and techniques used.

Following on from methodology chapter (Chapter 3), there is a series of chapters covering the applied research that was performed:

- In Chapter 4 a vehicle systems design tool was used as cyber-security testbed.
- The application of fuzz testing to networked automotive systems begins with Chapter 5 which describes the design of the software for an automotive CAN fuzzer.
- There is an evaluation of the developed fuzzer against physical targets in Chapter 6.
- In Chapter 7 a vehicle gateway component was a more complex target for the fuzzer. The target's hardware design is an obstacle, but the results are interesting.
- A car's media interface component is the target in Chapter 8. The isolated bench testing of CAN components again proved problematic but provided additional results.
- In Chapter 9 a vehicle's display ECU is the target of fuzz testing. Using a component with a visible interface overcomes a problem with the cyber-physical nature of a car's systems. The resilience of the component is found to be poor and there are indications of bugs present within the component's software.

The final chapter, Chapter 10, is the concluding discussion, covering:

- The conclusions drawn from the experiments.

- The knowledge and challenges uncovered.
- A discussion on the developed methods.
- Suggestions for further research work.

Some of the information derived for this research program is not core to the main thesis. However, the knowledge does provide useful supporting information. Due to limitations on thesis space that information is provided in Appendices.

- Appendix A provides some additional historical background on the computerisation of vehicles and computer-controlled vehicle functionality.
- Further technical details on vehicle networking technology and the CAN bus is provided in Appendix B.
- Appendix C discusses a weakness of the CAN bus. The weakness is considered for use as a possible cyber attack against a vehicle. This was derived from observations made during the development and validation of the CAN fuzzer. It was noticed that connecting a node at the wrong bit rate could cause the network to fail. This could be used as a cyber attack. An experiment was performed to validate this observation.

There are three Appendices related to housekeeping:

- In Appendix D the permissions for any sourced images are provided.
- Safety considerations for any readers who wish to reproduce the experiments performed in this thesis, or undertake similar experiments, are provided in Appendix E.
- Following the Coventry University's thesis requirements, the ethics documentation is provided in Appendix G.

Finally, two Appendices provide some additional technical information.

- Details on the CAN packet log file search method, used in Chapter 9, are in Appendix F.
- Appendix H contains additional technical information on the CAN fuzzer software development, supplementing to the content of Chapter 5.

1.5 Publications

The following publications were drawn from, or influenced by, this research program and thus, contain content that appears in this thesis.

- **Daniel S. Fowler**, Madeline Cheah, Siraj Shaikh, Jeremy Bryans, "Towards a Testbed for Automotive Cybersecurity", 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST), March 2017 (**Chapter 4**)
- Madeline Cheah, Jeremy Bryans, **Daniel S. Fowler** and Siraj Ahmed Shaikh, "Threat Intelligence for Bluetooth-enabled Systems with Automotive Applications: An Empirical Study", 3rd Workshop on Safety and Security of Intelligent Vehicles (SSIV 2017), June 2017 (**Chapter 4**)
- **Daniel S. Fowler**, Jeremy Bryans, Siraj Shaikh, "Automating fuzz test generation to improve the security of the Controller Area Network", ACM Chapters Computer Science in Cars Symposium: CSCS 2017, July 2017 (**Chapter 5**)
- **Daniel S. Fowler**, Jeremy Bryans, Siraj Ahmed Shaikh, Paul Wooderson, "Fuzz Testing for Automotive Cyber-security", 4th Workshop on Safety and Security of Intelligent Vehicles (SSIV 2018), June 2018 (**Chapters 5 and 6**)
- **Daniel S. Fowler**, Jeremy Bryans, Madeline Cheah, Siraj A. Shaikh, Paul Wooderson, "A Method for Constructing Automotive Cybersecurity Tests, a CAN Fuzz Testing Example", A3S 2019: IEEE International Workshop on Automobile Software Security and Safety (**Chapters 3 and 9**)

1.6 Summarising the introduction

This chapter began with an overview on how advances in technology have exposed the modern vehicle to the threat of cyber attacks. In the age of the increasingly connected and software-driven car cyber-security testing of vehicle systems needs to be addressed. This high-level need was formulated into a research question on using fuzz testing in the automotive field. It is motivated by the success of fuzz testing in the general IT domain and by a gap in the existing published work, where a lack of detail is evident.

To enable practical fuzz testing the common CAN bus is targeted due to its ubiquity and common interface to vehicle ECUs. To meet the aims and objectives an easy to deploy and use CAN fuzzer was developed to aid with gathering empirical evidence. The CAN fuzzer was used to experiment on physical and simulated vehicle CAN systems and components. The experimental results were used to assess the suitability of using fuzz testing in the automotive field.

As a result of this research seven contributions are provided to the research and engineering community. Furthermore, the discovered methodology can be used to further develop automotive security tests and tooling. Applying security from early in the automotive system's R&D process will enable the consideration of secure-by-design from the outset and, thus, increase the resilience of the software executing in the customer delivered vehicles. Improving resilience maintains assurance levels in vehicular systems.

Chapter 2

Literature review

> *. . . there are known knowns; there are things we know we know. We also know, there are known unknowns, that is to say, we know there are some things we do not know. But there are also unknown unknowns, the ones we don't know we don't know.*
>
> Donald Rumsfeld

The Introduction (Chapter 1) briefly discussed how advances in technology have led to vehicles becoming part of our connected world. This means cars will be exposed to cyber attacks and, therefore, the considerations of cyber-security become applicable. This literature review examines those issues in detail, investigating the new field of automotive cyber-security and the problem of testing vehicles for cyber resilience. The review shows that this new field is relatively young (a couple of decades-old), multidisciplinary, complex and the focus of both commercial and academic attention.

2.1 Aims of the review

The aims of the literature review were to:

1. Review the concepts and definitions of systems security.
2. Gain an understanding of the newly emerged field of automotive cyber-security, and thus the landscape under which this research was being performed.
3. Understand how topics in the new field apply to the high-level objective of this research (how to test a vehicle for cyber-security assurance).
4. Determine a gap in the automotive cyber-security field.
5. Use the gap in the knowledge to determine the topic of this research.

2.2 The literature discovery method

Automotive cyber-security is multidisciplinary in nature and contributions to the field can be found under these topics:

- Computer science - systems design, networking, cryptography, programming.
- Electronic engineering - embedded computers, electronics, wired and wireless data networks, testing.
- Operations management - processes and procedures, policies, training, engineering teams.
- Systems security - investigating how security applies to all of the above items.

Therefore, papers that can be regarded as being relevant to the automotive security field can be found in many different types of journals and conference proceedings covering the above topic areas, as well as specialised publications in the field of transportation. Thus, despite automotive cyber-security being a niche field, relevant articles are published over a broad range of journals and proceedings.

In terms of the taxonomy, keywords that apply to the field are fairly generic. For example keywords such as automotive, vehicle, security, systems, networking, encryption and testing. Keywords such as those, used in different combinations, generally return many thousands of search results. Therefore, it would have been inefficient to rely entirely on searches of databases for publications. Thus, the methods to find relevant papers included:

- Paper recommendations from colleagues.
- Searches of databases (Google Scholar, IEEE Xplore, ACM Digital Library, SAE Mobilus), concentrating on high citation peer-reviewed papers.
- Using references in relevant papers to find additional papers.
- Revisiting literature sources during this research to examine recent work.

This review chapter begins by looking at the concepts behind system security, starting with a discussion on software assurance, security properties, and established security terminology. A discussion is provided on the growth of the relatively new field (when compared to other science and engineering fields) of automotive cyber-security. Then the underlying digital technology of vehicles is discussed, before examining how such systems should be security tested. The review finishes with a look at how a test method called fuzz testing applies to automotive systems.

2.3 Software assurance and security properties

The modern vehicle is a Cyber-physical System (CPS), a kinetic machine unable to function without the computer systems operating under its metal skin. The kinetic nature of a car makes it different from traditional information systems, however, it is still software that is providing the command and control. Software security has been studied for several decades, and for users of computer systems, which now includes cars, software security is related to the concept of software assurance. To provide assurance is to provide a degree of confidence. Quoting from the United States (US) Department of Defence [10]:

> *The subversion and sabotage of software always results in the violation of the software's security, as well as some if not all of the software's other required properties. These include such properties as correctness, predictable operation, usability, interoperability, performance, dependability, and safety.*
> ***Software assurance** has as its goal the ability to provide to software acquirers and users the justifiable confidence that software will consistently exhibit its required properties. Among these properties, security is what enables the software to exhibit those properties even when the software comes under attack.*

Thus, software security is important for software assurance, which provides confidence to systems users. Drivers and passengers are probably unaware of the complexity of a vehicle's software. However, the expectation is that all the vehicle's functions work and the vehicle is safe to use. Ideally, a cyber attack will not change that expectation if software assurance is high. But what constitutes the security of a software-based system? The US National Institute of Standards and Technology (NIST) provides three aspects: Confidentiality, Integrity, and Availability (CIA) [11]. These properties are commonly known as the *CIA triad*. The NIST definitions for the triad are quoted from the US E-Government Act of 2002 [12], though earlier definitions exist [13]:

- **Confidentiality**, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information.

  A loss of confidentiality is the unauthorized disclosure of information.

- **Integrity**, which means guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity.

  A loss of integrity is the unauthorized modification or destruction of information.

- **Availability**, which means ensuring timely and reliable access to and use of information.

  A loss of availability is the disruption of access to or use of information or an information system.

The CIA triad is used to analyse a system's security methods. German and Swedish researchers began viewing vehicle security in terms of CIA in the mid-2000s [14], [15], as did the European Community (EC) E-safety vehicle intrusion protected applications (EVITA) project that ran from 2008 to 2011 [16]. However, there is some confusion in the literature with regards to the definition of security properties.

In [15] authentication and data freshness are listed as security properties alongside the CIA triad. However, authentication and data freshness can be regarded as functions to help preserve integrity and confidentiality. Likewise, in [16] CIA properties and requirements/functions (e.g. non-repudiation) are listed together, however, the NIST definition separates properties from the mechanisms to protect them, thus non-repudiation is functionality required to aid data integrity. In this thesis, the NIST definition is used for the intrinsic nature of security (the CIA triad), and security *functions* are a system's requirements needed to protect those properties.

Indeed, a system will need to implement several security mechanisms to protect its CIA properties. As an example, a system that allows access from a remote location may implement *Triple-A* security, i.e. the functions of Authentication, Authorization, and Accounting (a.k.a. Auditing), as defined in [17]. An example of AAA implementation is the established Remote Authentication Dial-In User Service (RADIUS) protocol [17]. Thus, Triple-A is the mechanism to aid the protection of the CIA triad. This separation of system security properties from the mechanisms that protect them is important. A system can be tested for breaches of the CIA properties (if any), and then security mechanisms applied to resolve the breaches.

2.4 Threats, targets, countermeasures and evaluation

When discussing cyber-security different terms are used for those that perform attacks. Many terms exist, including adversaries, attackers, hackers, and malicious users. What the terms represent are a security threat to a system. Threats can be automated by attackers. A software robot, known as a *bot*, can be used to harvest data, fill in online forms, post to social media and scan for known system weaknesses. The traditional computer viruses and malware are other automated threats.

To provide an all-encompassing term for security threats, an international standard is used. ISO/IEC 15408-1:2009 [18], *Information technology - Security techniques - Evaluation criteria for IT security*, defines the term *threat agent*. Threat agents are not only malicious. Human errors, whether through incorrect design, bugs in the implementation, use or configuration, can all represent a threat.

The standard provides a model for a high level view of threats. In Figure 2.1, derived from the standard, security concepts and relationships are displayed. The model shows how threat agents increase the risk to a system, an asset, and how countermeasures are required to reduce risk. The term threat agent, as defined in the standard, will be used as the generic term for the source of threats.

The ISO/IEC standard provides another useful term. In engineering fields, such as automotive manufacture, the term Device Under Test (DUT) is used when carrying out functional testing of a component. The term System Under Test (SUT) is used when an entity consisting of several

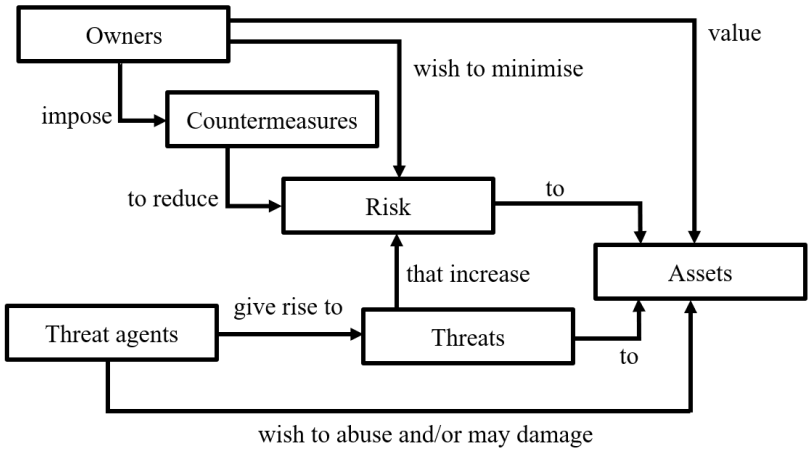


Fig. 2.1 Security concepts and relationships were redrawn from ISO/IEC 15408-1:2009 [18]

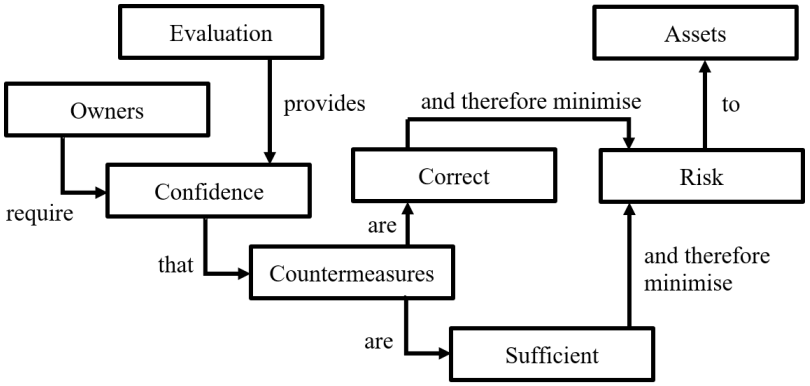


Fig. 2.2 Evaluation concepts and relationships were redrawn from ISO/IEC 15408-1:2009 [18]

components is being tested. The term used by the ISO/IEC standard is Target of Evaluation (ToE). This is a useful term to cover both DUT and SUT and to emphasise the security side of testing, as opposed to the usual functional testing of a DUT/SUT. Thus, ToE will be used as the term to cover the item being subjected to security testing.

The ISO/IEC standard provides a model for understanding the importance of security testing a ToE. The *evaluation* of security, not only through testing but including audits and reviews, is shown in Figure 2.2, redrawn from the standard [18]. It shows the evaluation concept's relationship to reducing risk and, hence, increasing confidence. This relates to the discussion on assurance in the previous section. Both of these ISO/IEC models provide a useful and concise visual aid to understanding the importance of security testing.

2.5 The malleability of the computer

There are many beneficial features of modern vehicles that are the result of computerised systems. What manufacturers need to be aware of is the fundamental security issues this presents. The car, and the data it stores are the assets that are being targeted by threat agents. These agents will look for weaknesses, known as vulnerabilities, in the vehicles computerised systems that they can exploit. This will enable them to carry out malicious actions. This is where the fundamental problem exists, the very nature of the computer itself.

The computer is a general-purpose machine. If you can manipulate or change the software you can change the machine. Computer hacking is all about manipulating the computer software to get the machine to do something beyond its designed functionality, and in doing so compromise the system security. Thus, the act of hacking is regarded as a malicious operation, aimed at breaking the CIA triad (see section 2.3) by gaining access to hidden or private data and resources (confidentiality), changing data or operational settings (integrity), and disrupt, take control or damage systems (availability). In many countries hacking is illegal, although security testers, researchers and *white hats* can obtain permission to do it legally. Knowing that a computer-based system can be manipulated, manufacturers must ensure adequate countermeasures are designed in, and tested, to maintain confidence, i.e. assurance, as illustrated in Figure 2.2. Such cyber-security assurance now applies to the modern connected car.

2.6 A brief automotive cyber-security history

Computers and security have always gone hand-in-hand. The first programmable digital computer, Colossus, was built to breach security. It was designed to help break the German Enigma codes during World War II. In the 1960s the Rand Corporation produced a report highlighting many of the security concerns that are still associated with computer-based information systems [1]. Computer security breaches have been around for almost as long as computers themselves. The original room size computers of the 1960s and 1970s, running time-sharing operating systems, were hacked to allow for cheap access to valuable computing time, and long-distance hacking predates the Internet [19]. However, to compromise a car remotely it needs to be wirelessly connected. Although computers have been in cars for over thirty years, for most of that time they have been a closed box, isolated from other systems and each other. However, with the Connected and Autonomous Vehicle (CAV) that box has been opened. Now the cyber-security problem that plagues enterprise IT, the Internet, and the Internet of Things (IoT) systems, has been released onto vehicular systems.

The interest in car hacking was heightened in 2010 when Koscher et al. published a paper [2]. They demonstrated that previously theoretical cyber attacks [20] against cars where possible. They were able to hack a vehicle and induce dangerous behaviour. They acknowledged that other papers had previously expounded the risks to vehicle computer systems. Unlike previous papers, it was not a theoretical threat assessment or a simulated attack, but a practical real-world cyber attack experiment

against a vehicle. It successfully validated previous assessments and provided proof of the theoretical possibilities of automobile hacking.

Koscher et al. noted that the attack surface of vehicle cyber systems is increasing in size due to the increase in computerisation, including the use of software to control previously mechanical systems. They showed how it was possible to compromise these systems with a variety of methods. The issues identified, and still relevant, include:

- Media systems: These are connected to the in-vehicle data communications networks and have external interfaces (for example USB and Bluetooth), and can be upgraded by vehicle users. Thus, they can be used as staging posts for gaining a foothold into the vehicle systems.
- Component standardisation: Components are used across a manufacturer's vehicle models. This is because it is not economical, in time and cost, for a vehicle manufacturer to build brand new ECUs for every new model. Thus, a security weakness in a single component can affect many models. Such standardisation is beneficial for aftermarket vehicle maintenance since replacement components fit several models. However, this also benefits the community of car modifiers, as it eases knowledge dissemination on how to access vehicle systems, including overcoming security [21].
- Weak or non-existent access controls: Controlling physical access to any system is part of security engineering, however, access to vehicle systems and components is possible once the vehicle is delivered. Furthermore, connections between functional boundaries within the vehicle means that crossing those boundaries was possible, due to non-existent or poor security.

As a result, modern vehicles, with some initial effort, can be compromised. The researchers stated that whilst vehicle systems were designed to be fail-safe, they had not been designed to be attacker safe, the attackers being able to deliberately cause a failure or otherwise override the fail-safe mechanisms.

In their opinion the opportunity for hacking will increase as more functionality is added to vehicles, adding more attack vectors. Their research highlighted plenty of zero-day exploits (previously unknown computer bugs that allow cyber-attackers into a system). The paper highlighted the failure of ECU manufacturers to properly test their designs. The use of fuzzing (systematically trying all values to a component interface) found operational modes that were inherently dangerous and should not be allowed. The researchers could use combinations of settings that probably had not been conceived of as a possibility, or tested, by the manufacturers. No part of the computerised vehicle system was safe from tampering. It was possible for them to conclude that vehicle systems were not designed to be tolerant to cyber attacks.

Criticism that physical access to vehicles was required to propagate attacks was addressed in their follow up paper in 2011 [22]. After some substantial effort, the researchers reverse engineered the software and the communications protocol for the telematics ECU. Used by the vehicle for

Fig. 2.3 In 1997 Citroen demonstrated Intel's Connected Car PC concept, designed to allow for in-vehicle email and voice-controlled PC functionality, see Appendix D for image permission

communicating with a call centre. After discovering vulnerabilities with the code they were able to execute remote attacks from a laptop, using it to control vehicle functions.

In 2015 researchers performed a remote attack against a vehicle travelling on a highway [23]. Substantial effort was required to reverse engineer code and find the vulnerabilities need to achieve the hack. In the multi-stage attack, they used a laptop to remotely connect to the vehicle and control it. The attacked resulted in worldwide media exposure and led to a recall of 1.4 million cars to address the security issues discovered. Recalls are the mechanism to deal with fixing design faults in vehicles that are deemed hazardous. Unlike some recent modern vehicles, where software updates can be delivered over-the-air (OTA), the recall was required due to the update having to be applied locally to the vehicle. Ultimately, the overriding concern is one of public safety, cars can cause injury and death. The demonstrated 2011 and 2015 car hacking could compromise personal safety.

One key difference to point out with the car hacking versus traditional IT systems, is the emphasis on which of the CIA properties are being compromised. Whereas in traditional IT systems confidentiality of data is highly important, in the safety important world of the cyber-physical car, system integrity and availability is crucial.

A common theme in the recent automotive cyber-security research is the lack of resilience within a vehicle's computerised systems. The lack of system protection was known prior to the high profile demonstrations of car hacking. The cyber-security issues had been prophetically discussed in 2004 [20] in a presentation to the commercial Embedded Security in Cars (escar) conference. This long-standing vehicle security conference began in 2003 in Germany [24]. 2003 was the same year the UK government had banned the use of cell phones whilst driving, effectively mandating hands free phone operation in vehicles, and thus vehicle connectivity to the cellular network via the fitting of Bluetooth interfaces to vehicles. Earlier vehicle connectivity includes Intel's Connected Car PC technology, see Figure 2.3, demonstrated in Frankfurt in 1997. By then General Motors had already launched OnStar for wireless-based vehicular services. With security implications of new vehicular technology known in the early 2000s system security was one of the deliverables of the EVITA project [16]:

Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios

For that deliverable, a set of use cases and threats were examined to develop security requirements that were then applied to in-vehicle networks. The EVITA project viewed vehicle security as a set of objectives, those being:

- *Operational – to maintain the intended operational performance of all vehicle and ITS functions;*
- *Safety – to ensure the functional safety of the vehicle occupants and other road users;*
- *Privacy – to protect the privacy of vehicle drivers, and the intellectual property of vehicle manufacturers and their suppliers;*
- *Financial – to prevent fraudulent commercial transactions and theft of vehicles.*

Real-world violation of EVITA operational and safety objectives has certainly been demonstrated [2], [22], [23]. To meet the EVITA objectives, vehicle systems need security functions that protect their fundamental CIA proprieties. However, how can the impact of security violations be assessed? For EVITA the security objectives are assessed against a classification scale [16] which are influenced by, and extended from, the established hazard analysis and risk assessment (HARA) method from the international ISO 26262 (Functional Safety for Road Vehicles) [4], [25]. (ISO 26262 is used by vehicle manufacturers and their suppliers for functional safety testing and risk reduction in their complex electrical and electronic systems.) The EVITA severity classes have influenced subsequent work. They are being used to assess threats to automotive systems [26]. The classes appear in the automotive industry's SAE J3061 (Cybersecurity Guidebook for Cyber-Physical Vehicle Systems) [27], see Section 2.9, which also contains EVITA's Threat Analysis and Risk Assessment (TARA) (used to help assess risks from identified potential threats).

2.7 Insecure vehicle technology

Much of the research into breaking vehicle security has been focused on exploiting technology that was designed before the advent of connected cars, and therefore before cyber-security was a design consideration. Published automotive exploits take advantage of a lack of security in aspects of vehicle technology. In this section, two technologies, common to most manufactured vehicles, are briefly examined. These are the Controller Area Network (CAN), and the On-board Diagnostics (OBD) port. The following sections provide an overview of those two technologies, with further technical details and resources provided in Appendix B.

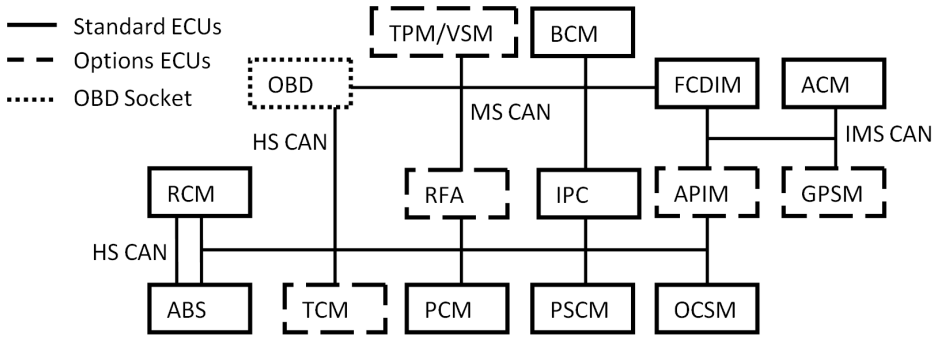


Fig. 2.4 Networks connecting ECUs in a small car, three are at 500 Kbps (2 named HS CAN - probably for High Speed, one named IMS CAN - abbreviation unknown) and one at 125 Kbps (MS CAN, probably for Medium Speed)

Table 2.1 ECUs in a small car, full abbreviations redacted for commercial confidentiality reasons

Abbr.	*Function*	*Abbr.*	*Function*
OBD	On-Board Diagnostics connector or Data Link Connector (DLC)	IPC	Instrument panel
PSCM	Power steering control	APIM	Media functions interface
OCSM	Passenger detection	RFA	Key fob functions
RCM	Safety belt control	ABS	Anti-lock brake system
TCM	Transmission control	PCM	Powertrain control
TPM/VSM	Tire pressure mon./security	BCM	Body control
FCDIM	Passenger display screen	GPSM	Global positioning system
ACM	Audio control		

2.7.1 The Controller Area Network

In examining the communications technology of the in-vehicle data networks, there is a common protocol found across all manufacturers. The data communications between the vehicles ECUs is dominated by the protocol known as Controller Area Network, often referred to as the CAN bus. Figure 2.4 shows the in-vehicle networks for a small compact hatchback from a major vehicle manufacturer. All the ECUs are interconnected via CAN.

Other communications networks are used in vehicles, see Table 2.4. In Figure A.3 in Appendix A the in-vehicle networks for an executive saloon car are shown. In that case MOST is used for multimedia and infotainment systems, Ethernet for software updates and advanced diagnostics, and Flexray for vehicle dynamics (handling). However, it is CAN that is often used to control a majority of the vehicle's functions and thus, is an understandable attack vector.

Despite the sophistication of the modern vehicle some of the underlying digital technology can be relatively straightforward, this includes the CAN bus. CAN is a serial communications protocol for data exchange between computational units operating in harsh and electrically noisy environments, such as the car (or factories).

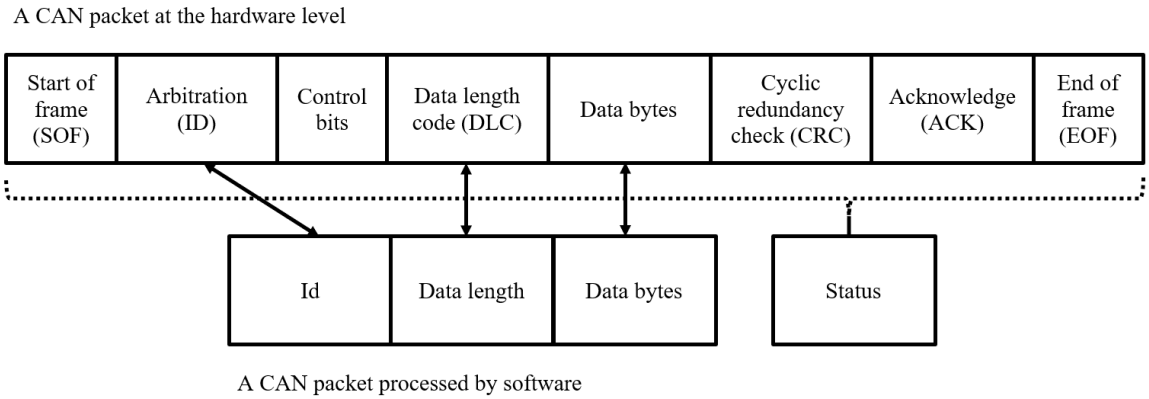


Fig. 2.5 A schematic of a CAN data packet, from the software it is seen as an id, data length and data bytes, the CAN hardware handles the protocol, with any errors reflected in a status code

Table 2.2 Data elements of a standard CAN packet, as seen by ECU software

Item	*CAN name*	*Integer range*	*Description*
Packet id	Arbitration	{0,1,2,...,2047}	Packet identifier
Data length	Data Length Code	{0,1,2,...,8}	Number of data bytes
Data bytes	Data	{0,1,2,...,255}	0 to 8 payload data bytes

Much of the CAN protocol is handled in dedicated hardware. The transmission of the CAN packet, and the handling of any errors, over the cabling is performed by a CAN transceiver chip. This leaves the software in the communicating ECUs to handle the packet contents. From the ECUs viewpoint a CAN packet is in three parts, a packet identifier (id), a data length field, and the data payload, see Figure 2.5 and Table 2.2.

A CAN packet with an id of 8 and one byte of data, would be sent with the length byte set to 1, see Table 2.3. Such a CAN packet could be used to transmit a command to turn on a headlight, with the state of the second bit determining if the headlight is on or off, see section B.3 in Appendix B.

CAN predates the World Wide Web and thus, the advent of the connected car. Today's car hacking was not anticipated and, therefore, CAN was not designed with cyber-security considerations in mind. Its simplicity (compared to other protocols) is part of its functional robustness, and that simplicity is also due, in part, to not having a security mechanism. There are error handling and cyclic redundancy check (CRC) bits as part of the data transmission protocol (see Figure 2.5 and Section B.2.4), but

Table 2.3 Example Basic CAN Data Packet

	Id (Arbitration)	*Data Length Code*	*Payload*
Example	8	1	2
Hex	0x08	0x01	0x02

Fig. 2.6 An OBD port in a vehicle

these are used for communications reliability, not for cyber-security. CAN's lack of security has seen research and proposals into retro-fitting security mechanisms, see Section 2.8.

CAN is a two-wire data bus, onto which multiple nodes (ECUs) can attach simultaneously. All ECUs have simultaneous read/write access to the bus. There is a data packet priority mechanism determined by a packet id (called the arbitration id). Each node does not have a network address, instead, nodes listen out for the data packets with the specific id they want to process. For further information on the workings of CAN see Appendix B.

While CAN's simplicity is good for functional robustness, simplicity becomes a vulnerability once a vehicle's systems have been compromised, making it a target for attacks [2], [22], [23]. Furthermore, alongside the insecurity of the protocol, a physical connection to a car's CAN bus is readily available from the driver's seat. This is covered in the next section.

2.7.2 The exposed OBD port

Several jurisdictions, for example, the USA, Europe, Australia and Japan, require a vehicle to provide exhaust emissions data. This is read through an On-Board Diagnostics (OBD) port, located close to the driver's seat, see Figure 2.6. This port exposes a direct connection to a vehicle's CAN bus. Aftermarket OBD devices can use this port. Such devices are not under the control of the manufacturers, but can still provide access to what manufacturers consider to be a closed internal network.

The aftermarket devices not only receive vehicle data, they are also capable of transmitting data. Researchers have shown these devices can be used to attack the CAN bus. Foster et al. [28] discovered several flaws in an OBD telematics system, proving the security risk of plug-in OBD modules. Furthermore, since the aftermarket units can provide remote access to the OBD port, even a previously non-connected older vehicle model can be made connected, and thus becomes vulnerable.

There are OBD devices that provide a connection to a cell phone app. This opens up the possibility of malicious apps being developed. App stores list apps that are used for the display and logging of vehicle performance data and to access diagnostics information. Despite their intended use for driver, technician and owner services they increase the security risk. An adversary does not have to be physically present within the vehicle cabin. Woo et al. [29] showed how malicious apps can gain

Table 2.4 In-vehicle data communications networks in common use, CAN is the most widely used

Network	*Name*	*Description*
CAN	Controller Area Network	Continuously transmits sensor and actuator data around a vehicle using twisted-pair cabling. The most widely used in-vehicle network. Data rates up to 1Mbps (500Kbps and 125Kbps are common).
CAN FD	CAN with Flexible Data-Rate	A version of CAN that increases the practical data rate to 4Mbps, not as popular as standard CAN.
LIN	Local Interconnect Network	Simple low speed (typically 20Kbps) serial bus to connect switches and actuators close to an ECU.
FlexRay	FlexRay	Deterministic time-triggered protocol using one or two twisted-pairs for up to 10Mbps. Found in power train and braking systems.
MOST	Media Oriented Systems Transport	Fibre optic ring network for multimedia data, 25-150Mbps.
USB	Universal Serial Bus	Provides a connection to an infotainment ECU for devices brought into a vehicle, 1.1 to 450Mbps.
802.3	Ethernet	Standard Ethernet (10-1000Mbps) may be used to connect high speed diagnostic equipment to a central gateway ECU.
FPD-Link	Flat Panel Display Link	Low-voltage differential signalling (LVDS) for relaying vehicle camera data (1-3Gbps) to an ECU. (LVDS is also used for laptop screens.)
100BASE-T1	Automotive Ethernet	A 100Mbps point-to-point two-wire version of Ethernet for high speed data application. 1Gbps (1000BASE-T1) soon available.

access to the in-vehicle network to perform the same attacks as other researchers, again compromising vehicle safety.

In a similar fashion to the IoT security issues, the OBD devices can have very weak security; a survey [30] showed that wireless Bluetooth OBD devices have unchangeable 4-digit PINs (some revealed publicly on the Internet), and the PINs and devices are generally easily discoverable (the Bluetooth process used to initiate a connection).

2.8 Applying existing security to the vehicle

Whilst in-vehicular networks and systems differ from business information systems, in many aspects the principles (networked computational systems) remain the same. Thus, longstanding security fundamentals can be applied. Bejtlich [31] argues that much of the existing body of security knowledge from previous decades remains valid despite changing technology.

In general IT cryptography is widely deployed, and there has been plenty of research and designs for its use on vehicle networks and systems [32]. For the CAN bus there are several security proposals, summarised in [33]. These include, individually or in combination:

- Message Authentication Codes (MAC) (the most common proposal)
- Key distribution
- Message counters
- Hardware Security Modules (HSM)

There is a performance penalty in encrypting CAN bus traffic [34]. CAN has a small packet size, supporting only eight bytes of data, and is designed for low latency data throughput. The act of encrypting the data in a CAN packet adds a large transmission overhead, in both time delays and packet volume. CAN's fitness to support encryption is low because it is being pushed beyond the initial 1980's design objectives. The later CAN FD is better suited to supporting encryption with its 64-byte data payload, however, it brings higher costs and requires additional technical knowledge, and, thus, has yet to be widely deployed.

HSMs were advocated for the EVITA project [16], which proposed an in-vehicle cryptography solution that placed a HSM in each ECU. This was to ensure the efficient handling of cryptography keys and the encryption and decryption processing time [35]. However, one aspect that is lacking in the literature is the impact that encryption delays may have on the human driver. This would be in terms of information display refresh rates and equipment initiation (start-up and activation) times, especially under extreme motoring conditions (e.g. emergency braking). No proposed schemes cover this aspect, therefore, published research is required in that area.

The EVITA HSM solution does not address other issues related to the management of cryptography systems for vehicles. Issues with CAN encryption include the management of cryptographic keys [36],

manufacturer's procedures, ECU parts compatibility and the knowledge of service technicians. These are all issues that need addressing to manage an encryption system for a vehicle's 20-year lifecycle. This management problem may be the reason why CAN encryption has not been implemented by vehicle manufacturers. However, work on adding encryption to vehicle networks continues, for example the AUTOSAR[1] vehicle software stack supports the use of MACs.

For now, for all of the above reasons, no proposed CAN security mechanism has been found to be adequate for production purposes [33]. If encryption is currently unsuitable for vehicle networks what are the other options? In other domains, firewalls are considered important. A gateway firewall is suggested to protect system boundaries in vehicles in [20], and a patent for one appeared not long after the first experimental car PCs appeared [37]. Commercial companies have responded to the emergence of automotive cyber-security by offering in-vehicle network firewalls [2].

In general computing, a firewall uses packet filtering and network Access Control Lists (ACLs) to segment networks and filter traffic for security purposes. CAN does support rudimentary filters, allowing ECUs to block packets not addressed to them. Correctly designed network segmentation and filtering schemes would help in-vehicle systems security, especially considering the CAN weakness to a Denial of Service (DoS) attack once a malicious connection has been made. Despite the weakness of the CAN protocol, an Intrusion Detection System (IDS) may provide useful indicators of attacks. An analysis of normal CAN packet traffic can be used to determine when malicious packets are transmitted [38]. However, just as in traditional IT, the firewall and IDSs are useful for detecting system compromise after product delivery. What can be done to improve vehicle security design?

2.9 The vehicular cyber-security testing requirement

The weakness in vehicle cyber-security resilience has already led to guidelines being published by governments [39]. The United Kingdom (UK) Government has key principles [40] for CAV security, requiring organisations to address security at all management levels, and throughout a products lifetime. The industry was already aware of the risks and has responded. The SAE trade body is interested in standardising automotive security testing. The work in SAE J3061 [27] lays the groundwork for systematic consideration of vehicular cyber-security threats during a vehicle's lifecycle. The J3061 guidelines are influenced by the processes within ISO 26262 [4] that apply to vehicle systems development. J3061 addresses how cyber-security considerations need to be added to the vehicle development process. As such J3061 intends organisations to treat cyber-security similarly to functional safety considerations, designed into the whole vehicle lifecycle. Designing security into a system from the outset is regarded as essential [10], [31], [40]–[42]. To that end, international efforts continue with the formal standard ISO/SAE 21434 (Road Vehicles, Cybersecurity Engineering), scheduled for release in 2020. The functional safety and cyber-security overlap is acknowledged in

[1] https://autosar.org/

[2] https://tekeye.uk/automotive/cyber-security/automotive-cyber-security-companies

the new standard, and within ISO 26262. Finally, the SAE work-in-progress J3061-2 will include recommendations for cyber-security testing methods.

For vehicle manufacturers and their suppliers' guidelines and standards require the implementation of practical security testing, an additional overhead in the development process for new models. Fortunately, their existing investments and expertise in functional testing can be leveraged for the challenges of cyber-security testing, and as the J3061 guidelines indicate, testing processes should not need to change a great deal. Furthermore, if cyber-security testing is performed early enough it can allow for feedback into designs prior to production. Such testing has to include the links beyond the boundary of the vehicle because the vehicle is now part of a wider cyber ecosystem. In the next section the types of security testing to be performed on vehicle systems and components are examined.

2.10 Three non-functional security tests

When engineers design a system they can specify functional security mechanisms, for example, authentication of a user via a logging in facility. Any designed in security mechanisms are to protect the CIA properties of the system. The functional security mechanisms will be defined in the system specifications. The test plans for the system will check that such defined security mechanisms function as intended [43].

What is often exploited by malicious agents is hidden, and unwanted, functionality [44], caused by engineering issues. In a system that uses software for much of its functionality, it is *bugs* that cause engineering issues. These bugs can take the form of:

- Logical errors in code (or models used to generate code) resulting in run-time bugs.
- Weaknesses in system design, for example, if no consideration has been given to data encryption.
- Functional bugs due to a mismatch between what the system specification states and how the system has been implemented, and these not being caught by the functional testing.
- Additional and undocumented features provided by third party components and software libraries. Examples include test functions, or features present that were developed for another use case (e.g. another customer).

Not all bugs can be exploited to reveal weaknesses, however, for the exploitable bugs three types of testing can be performed in an attempt to uncover them [27], [43]. Indeed, Section 8.4.7 of J3061 describes them as *"critical tools in evaluating the Cybersecurity performance of a system"*:

- **Vulnerability testing** - performing tests for security weaknesses and exploits using scanning tools and a corpus of known attacks.
- **Fuzz testing** - dynamically sending the system large amounts of random and malformed data to see how it responds, in an effort to reveal a vulnerability.

- **Penetration testing** - using intelligence and tools to attack a system based on how adversaries would attempt to overcome security mechanisms.

These three security tests are relatively new to automotive engineers [43]. Furthermore, they add to the existing systems testing workload. Integrating security tests systematically and rigorously into the existing vehicle systems testing regimes will take some effort, particularly considering the complexity and number of interfaces that can now be found on a vehicle, Table 2.5 shows the extent of the possible attack surface of a vehicle. Furthermore, an attack route is available via advanced features supported through Internet-based services (see Table A.2 in Appendix A), which can be probed for weaknesses at the client (vehicle) or server (service provider) ends.

Table 2.5 A list of interfaces that do not require physical contact with the vehicle, providing a large attack surface

Interface
Key or fob
Infotainment/hands-free Bluetooth connection
Cellular (including eCall) connection
Cellular Internet connection (e.g. 3G/4G/5G)
Global Positioning System (GPS)
Tyre pressure monitoring system (TPMS)
Ultrasonic sensors
Parking cameras
LIDAR
On-board Diagnostics (OBD) port interfaces
Wi-Fi hot spot provision
Other cameras (vision systems)
Home/service centre Wi-Fi connection
V2X (IEEE 802.11p or 3GPP)

The work required to implement the three types of security tests into vehicle testing regimes has begun [45], and this thesis and research is contributing to the required knowledge (see the publications listed in Section 1.5). In [46] researchers at Chalmers University of Technology have used the three security tests as part of a proposed *Start - Predict - Mitigate - Test* (SPMT) process to systematically analyse vehicular cyber-security. The SPMT process, which has the high-level steps illustrated in Figure 2.7, is very involved due to vehicle complexity and the nature of the possible threats, however, here is a brief summary of the four phases:

- **Start** - Perform an analysis of the vehicle systems to determine what needs protecting.
- **Predict** - Perform a threat assessment to quantify and rank risks.
- **Mitigate** - Apply countermeasures to the ranked risks. Economic considerations influence the application of countermeasures.

- **Test** - Apply the three security testing regimes (vulnerability, fuzz, penetration) to ensure that designs are resilient to attack, and that applied countermeasures are effective. Use test automation, where possible, for efficiency. Any revealed security issues must be reviewed.

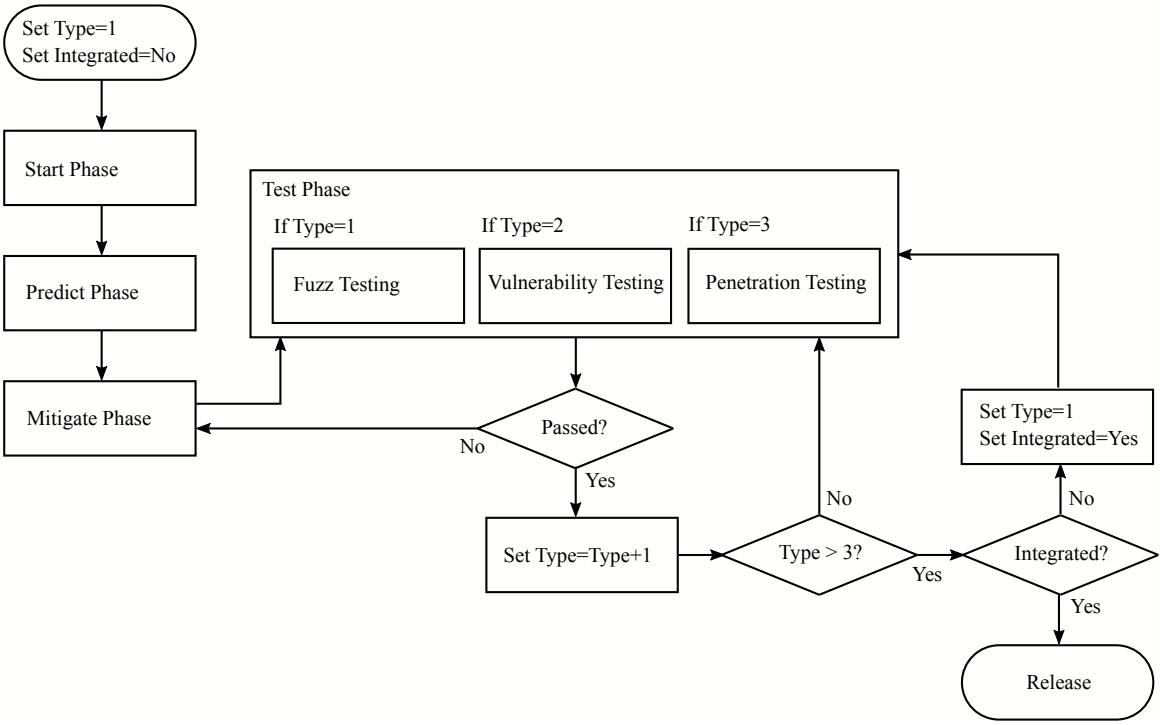


Fig. 2.7 The Start, Predict, Mitigate, Test (SPMT) security testing process, derived from [46], showing fuzz testing as one element of their proposed method, based on SAE J3061

Having already discussed that CAN is a key communications technology within vehicles, how do vulnerability, fuzz and penetration testing apply to CAN? In terms of vulnerabilities, its insecure design has been shown to be exploitable and can be used to endanger vehicles, for example [2], [22], [23], [28]. Indeed, the kudos and bounties available in discovering a vulnerability makes it a popular area of automotive cyber-security research. For penetration testing there has been some work performed on tools and techniques [21], [47]–[50], whereas, the available knowledge on fuzz testing vehicles is limited (see Section 2.12 further below).

Furthermore, J3061 suggests that vulnerability and penetration testing is performed by parties independent of the systems engineering development. Indeed, BMW proved the benefit of having vehicles tested by independent automotive security specialists [51]. Having an unbiased and independent analysis of a system by security experts can reveal unconsidered exploitation paths and system weaknesses.

This leaves fuzz testing as a suitable security test for manufacturers to perform themselves. A lack of publicly available resources and information to implement fuzz testing is likely to restrict fuzz

testing adoption. The following section discusses the application of fuzz testing to the automotive field.

2.11 What is fuzz testing?

As mentioned in the introduction (Section 1.2.4), fuzz testing is a dynamic test method, i.e. performed against a running system, as opposed to the static analysis of source code. Fuzz testing is important because of its track record of successfully revealing issues in the software of traditional information systems [8], [52]. Once software errors are revealed a possible exploitation path may be found. The fuzzed values may reveal unknown functionality, bypass security checks and cause internal buffer overflows or exceptions that disrupt processes because of software crashes. Ultimately these bugs or errors from fuzz testing may provide access to privileged areas or functions, reveal useful information, or allow for code injection techniques, thus, overcoming a systems CIA security properties. This is why fuzz testing is one part of the well regarded Microsoft SDL [3].

A fuzzer is a program that performs fuzz testing. The generation of random input data to a target system is a primary function of the fuzzer. However, to be more efficient, and hence more effective, a fuzzer can operate with an understanding of system data formats, communication protocols and interfaces. The essence of fuzz testing is:

- Random input (fuzz) is sent to a system's interfaces.
- The system response is monitored.
- If a system failure or issue occurs the conditions that caused it are recorded.
- The process is repeated a large number of times to cover a large input value space.
- Fuzz testing is automated for efficiency due to the high volume of tests that are executed.

Despite the established use of fuzzing for security testing [53], its use in general testing of automotive systems is low [54]. In a survey of testing methods in the automotive industry, fuzz testing was given in less than 4% of responses to questions, see Figure 2.8. In regards to fuzz testing in the automotive field there are few papers available, which are examined in the next section.

2.12 Fuzz testing CAN

Vehicle hackers see fuzzing as a useful tool to help reverse engineer systems [21]. This is because the operational details of vehicles internal systems are commercial secrets. Often the only way to determine what the values in a particular CAN data packet mean, is to capture the network traffic while operating a vehicle feature. If that fails then sending fuzzed data may get the required response. But reverse engineering is not the only use of fuzzing in vehicle systems, it can make an effective

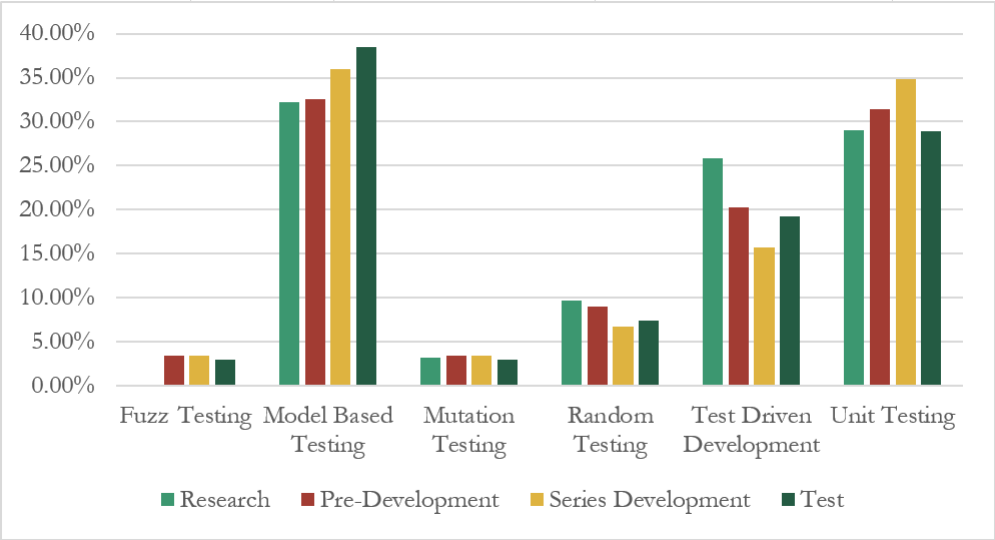


Fig. 2.8 Testing methods used in the automotive industry, derived from data from [54]

denial of service (DoS) attack [2]. However, little work has been done on examining the usefulness of fuzz testing for pre-production security testing. What is available with regards to fuzzing automotive systems?

The design of a fuzzer for Unified Diagnostics Services (UDS), used for ECU diagnostics, is provided by [55]. That report is mainly concerned with the design of the fuzzer, which was tested against a UDS simulator. It did find weaknesses in the simulator's UDS implementation, though no in-depth presentation of the results is provided.

A consideration when performing security tests is the oracle problem, how to distinguish correct from incorrect responses from the ToE [56]. When testing the normal functionality of a system the test oracle can be determined from the system specification, i.e. for a given input, a known output is expected. However, for security testing, the expected behaviour may not be definable, particular for penetration and fuzz testing. For vulnerability testing, the expected responses from the known corpus of attacks is defined.

Developing an oracle for fuzz testing is challenging. A basic test condition is to see if a system stops working (crashes) when the inputs are subjected to fuzzed values. However, an oracle needs to determine the incorrect responses, or for a CPS the incorrect behaviour, to fuzzed inputs when the system appears to be functioning as normal. When fuzz testing CAN the challenges faced by this research include:

1. **The dynamic nature of CAN:** Tests are dealing with running networks and not simple input/output interfaces. The CAN protocol is not complex, however, the vehicle systems are constantly transmitting data, usually without any time references, and despite the small packet size, the number of possible combinations of packets is impractically large.

2. **Lack of network traffic specifications:** There is no access to the specifications to know what a CAN bus should be doing. This is due to the inner workings of vehicular systems being trade secrets. Therefore, it's hard to tell if a test value from the fuzzer is causing an incorrect response, i.e. the value is producing incorrect behaviour.

3. **No access to ECU code:** The full behaviour of individual ECUs is unknown, again having no knowledge of internal ECU software and design, therefore, determining the correct response to an input is difficult.

4. **Interaction with the physical world:** Since a car is a CPS (mixing the virtual world with the physical world) a digitally generated test value or CAN packet may result in a physical, real-world action. Feeding back such physical responses to the oracle is another consideration.

Whilst access to ECU and CAN bus specifications present a challenge for this research, it should not be problematic for the vehicle manufacturers. Their relationships with their suppliers and their own designs provide the required knowledge to aid the development of appropriate oracles. For pre-production systems developed on HIL/SIL testbeds, the test oracle could be derived from the testbed configuration files.

Consideration of the oracle problem is in a commercial report [57], where a link from an automotive HIL and SIL test and development system, into the Python-based open source fuzzer, called booFuzz, is presented. In the report they propose several ways that a test oracle can be used:

- Network communication monitoring: reading the values present in the network data and checking against expected results.

- Monitoring through a component debug interface: a common electronics interface for debugging purposes is a JTAG port (JTAG is for Joint Test Action Group, the body that defined the standard). However, access to JTAG is not usually possible once manufacturing begins.

- Direct and indirect monitoring of simulator system signals: HIL/SIL testbeds can provide programmatic access to the simulated systems internal data values, allowing access to those values from the oracle.

- Accessing internal ECU values: Making use of the automotive Universal Measurement and Calibration Protocol (XCP), which allows remote access to the internals of an ECU.

- Monitoring of the physical responses: the real responses to values input into a system can be detected with external sensors.

However, one point to make, which was not made in [57], the monitoring capabilities maybe used by the attackers themselves, who look for any source of information to help break systems. For example, supporting XCP may help with detailed ECU diagnostics, but it provides another channel that may be exploited. One interesting point noted from [57] is that automotive ECUs have different

Table 2.6 General purposes fuzzers adapted for automotive fuzz testing

Tool	*License*	*Approach*
beStorm [58]	Commercial	Protocol based
Defensics [59], [60]	Commercial	Protocol based
booFuzz [57]	Mixed	Design based
Peach (www.peach.tech)	Mixed	Protocol based

operating modes. For most of its life, an ECU is providing normal operational functions. However, during vehicle servicing an ECU can be locked or unlocked for software updates via UDS. The point was made that it is important for testers to cover all the states of an ECU, as these different states have been exploited [23].

How a commercial fuzzer is configured to interface to an automotive network is provided in [58]. No practical application is given, only publishing results on data packet throughput rates. In [59], presented at a commercial conference, another commercial fuzzing tool is used to test a single ECU. The test environment defines a comparison module that acts as the test oracle, providing a means of verification for the operation of the ECU under test when its communication packets are being fuzzed. Again, there is a lack of detail on the test setup and results.

An interesting solution to the oracle problem is provided by [60]. They propose fuzz testing a ToE alongside an identical system (not subjected to the fuzz testing) as a reference point. Comparing the output from the fuzzed system to the reference system. The proof-of-concept system uses two model cars as the SUTs with the systems of the vehicles controlled by CAN busses. A dSpace HIL system is linked to the commercial Defensics fuzzing tool. The HIL is able to load, start and stop the fuzzer. When the fuzzer is active the measured analog and digital signals from two SUTs are compared, any differences are logged as errors.

As with the other publications, there is a lack of detail. The reported results are sparse, being a single high-level summary table, furthermore, detail on the internal operation of the fuzzer and the communication between the HIL system and SUTs would be beneficial. Information on the performance of the system would be useful, for example, does the analysis of the outputs keep up with system operation. There is no discussion of the variation between the two SUTs. How much of a difference between the outputs of the fuzz tested ToE, compared to the reference SUT, is required before an error condition is considered. An interesting point is made about the reference SUT. If a model of the SUT can run on the HIL system, and the model is accurate, then the reference SUT could be virtual and used within the HIL system itself.

2.13 Summarising the automotive fuzz testing literature

All the fuzzers used in the discussed publications, plus the Peach fuzzer, which is advertised as supporting automotive testing, are listed in Table 2.6. Most of the fuzzers are general-purpose

commercial products or have commercial versions (booFuzz is open source and Peach has an open source version). They all require specific knowledge on configuring them to work with automotive systems. There are two main approaches used by the fuzzers:

- The network protocol approach, based upon knowledge derived from network specifications, for automotive CAN this means using the format of the CAN data packets (see Table 2.2).
- The design approach, using input from the system design, i.e. having pre-existing knowledge of the data packet contents (which could be informed from the source code running in an ECU).

Whilst it can be seen that automotive fuzz testing is not new there are gaps in the literature. The available information in the literature has two major concerns. Firstly, there is a lack of detail in experimental design, data and results. Secondly, most publications are overviews of experiences in implementing a commercial fuzzer, rather than providing details on the effectiveness of the fuzz testing, and the practical methods used and lessons learnt from the experiences. There is a need, and opportunity, to add to the fuzz testing knowledge in the automotive field. This lack of knowledge of automotive fuzz testing does not only apply to CAN. There are many technologies in use in a vehicle, and the large attack surface (see Table 2.5) provides plenty of scope for more research into applying fuzz testing to vehicle systems, particularly around methodologies. The experiences from the CAN fuzz testing experiments in Chapters 6 to 9 is beginning to address the gaps.

2.14 Drawing from the review

The intention of the literature review was to begin to address the questions posed in the introduction. What is cyber-security in relation to a vehicle, and therefore, how can testing for resilience be addressed? Through gaining an understanding of the nature of the automotive cyber-security field it is possible to develop an understanding of related testing issues. The key areas addressed by the review are:

- **A description of issues with the literature discovery:** The field of automotive cyber-security is complex and multidisciplinary in nature, using terminology that can be generic, for example, *system security*, thus, subject terms are generally applicable across domains. To overcome this problem literature was found by combining database searches with the use of references from highly cited works, and recommendations.
- **A historical look at automotive cyber-security:** The security problems in this new field is a continuation of issues that have been with us since the early days of computing. Security issues have emerged from the increasing use of computers in vehicles. Researchers in the 2000's and 2010's revealed the lack of security resilience, mainly through unconsidered security design. However, the cyber-physical nature of the modern car adds another consideration.

- **Two security weak technologies in the modern vehicle:** The CAN bus and the OBD port were discussed. Issues in applying security to these two items were covered. Although there have been some proposals for the use of cryptography in CAN, they have not been widely adopted.

- **The need to design security into vehicle systems:** This was noted with reference to the requirements of J3061 (and other sources). However, all three of the recommended security test methods (vulnerability, fuzz and penetration) are immature in the automotive field. In particular, fuzz testing has seen little work, mainly covered in terms of describing the application of existing fuzzers.

From the literature, it can be seen that the research field of automotive cyber-security is young, less than a couple of decades old. However, it is currently the focus of a lot of commercial and academic research attention. The reports and papers demonstrate that research is occurring in many countries and types of organisations, in Europe, particularly the UK, Scandinavia and Germany; in the US, China, Korea, Japan and Israel; and in a variety of vehicle manufacturers, component suppliers, cyber-security companies, start-ups, academia and governments.

Researchers have shown that for CAVs there exists the possibility of remote infiltration and vehicle component compromise, or the attachment of an aftermarket device that can be compromised or purposefully modified. The result is a CPS operating in the presence of one or more malicious entities, either locally to the vehicle or remotely from a wireless connection. The scenario is similar to man-in-the-middle (MITM) attacks, where the adversaries can manipulate the system for their goals. In such scenarios, where the vehicle systems are breached, it raises issues:

- The threat agents may compromise the safety of the vehicle occupants and the public.

- The CIA security properties of the system have been violated and therefore the system design needs to change.

- Security testing, e.g. fuzz testing, should be performed to improve resilience and reduce the risk from future security breaches.

There is no doubt then, that for the automotive industry, security testing is now a consideration. It will be adding to the cost of vehicle systems development, however, as stated Sections 1.2 and 2.9, testing is part of the vehicle development process, and the industry will implement useful security testing methods.

Whilst the review has looked at technical aspects of automotive cyber-security, the nature of the automotive industry requires the consideration of business factors when implementing new technology. Factors that are seemingly ignored in technical literature. Conversations with stakeholders at a HORIBA MIRA Limited have raised the issues of barriers to research in this field.

2.14.1 Barriers to automotive cyber-security research

For vehicle manufacturers, the complex engineering process is one part of designing a car. Other aspects include design and production costs, weight, maintenance, vehicle servicing and vehicle end-of-life. These aspects also apply to engineered cyber-security solutions, therefore, previously proposed security solutions based around cryptography are challenging to implement [33]. The long-term management of proposed cryptography solutions remains a challenge for manufacturers to address.

Furthermore, research needs to contend with the nature of the automotive domain itself. Automotive engineering is a high-cost enterprise, cyber-security researchers have noted this as an issue [61]. Purchasing vehicles for research is costly. Large facilities are needed to accommodate and to test vehicles. Garage space is required, and access to proving grounds to test vehicles under dynamic conditions. There are the associated costs of facilities (rental or purchase, energy and other utilities, insurance, physical security). Other barriers to research in the field are knowledge based. This includes access to experienced personnel, investment in time (gaining requisite knowledge and reverse engineering systems), and access to vehicle systems protocols and software (often restricted due commercial confidentiality). Nor does the automotive industry sit still, e.g. continuous investment is needed to research next-generation sensors and systems, as used, for example, in the evolving Advanced Driver Assistance Systems (ADAS) and CAVs. Therefore, security testing is not a singular target and will need to evolve as vehicle systems evolve. However, for now, manufacturers need practical security tests that can be fitted into the existing testing processes.

2.14.2 Understanding the motivation for this research

There already exists a complex vehicle engineering processes (see Sections 1.2 and 2.9). Therefore, vehicle manufacturers and their suppliers can consider adapting their existing pre-production test procedures and ISO 26262 considerations to incorporate security testing according to J3061 guidelines. Indeed, J3061 begins to answer the first question posed in Section 1.2:

What does it mean to cyber-security test vehicles?

The answer provided by J3061 is the need to perform vulnerability, fuzz and penetration testing on the vehicle systems. These three classes of testing need to be performed on multiple subsystems, components, networks and connection points. Due to the size and complexity of the systems in vehicles, the testing task is large. However, large scale testing is not new to the vehicle engineering process, but it needs enhancing with the security testing requirements. This needs to happen without being onerous to the manufacturers or suppliers costs, though the initial problem is likely to be a lack of security testing knowledge and procedures specific to the automotive field.

Developing the new security testing knowledge will entail multiple research streams to examine the many topic areas that are impacted. In this research, fuzz testing is seen as the area that requires immediate attention due to an existing scarcity of knowledge, and considering how important the

automotive industry regards it in the J3061 guidelines. Thus, the contribution made in this research is to provide a new CAN fuzzer, Chapter 5, a tool that can be used to introduce fuzz testing early into systems prototyping stages, Chapter 4. This may reduce future vulnerabilities because any issues found during security testing can be fed back to the engineers, see Chapter 9 for an example. Early feedback on security issues supports the concept of designing security into a product.

Chapter 3

Method

Science is what we know, and philosophy is what we don't know.

Bertrand Russell

This chapter presents the methodologies used and developed within this research program, a program based upon empirical evidence. This applied research program aligns with the medium to long term research aims of HORIBA MIRA Limited, Coventry University's industrial collaboration partner. One of the many areas of the collaboration's research programs includes the development of new methodologies suitable for testing aspects of vehicle resilience. As discussed in the literature review in Chapter 2, resilience to cyber attacks is required in order to maintain assurance in vehicle systems.

3.1 Introduction to research methods

At the outset of this research, the industrial collaboration had established a high-level objective. That objective was to research solutions to the problem of cyber-security testing the modern connected vehicle. The literature review covered the complexity of the modern vehicle and that the nature of cyber-security presents a challenge when testing a vehicle's security properties. That challenge needs to be addressed as the review demonstrated the importance of vehicle resilience to hacking attempts.

Beyond the high-level objective, there were no specific topics provided for this research program. Thus, this research involved iterative goal-seeking to establish the work to be performed by the program, perform that work, and provide the final contribution. The goal-seeking was in three major stages:

1. Firstly, the literature review was performed to find a relevant gap in the existing knowledge that could be investigated. This then raised the research question to be examined.

2. The second major stage was developing suitable software to support experimentation (software development), executing experiments with the developed software (experimental design) and evaluating the results (quantitative observations).

3. The final stage was to document the security development testing method that could be moved forward for future development by HORIBA MIRA Limited, and the wider automotive engineering community (methodological design).

Thus, the major research methods were literature review, software development, experimental design, quantitative observations and methodological design. Whilst gathering of data for qualitative observations was not part of this research, some general observations could be made, where appropriate, based upon the work performed throughout the program.

It is important to use a framework to provide some structure to any work being performed. Since the nature of this research was goal-seeking, an iterative process model was followed. The next section discusses the method used.

3.2 The iterative Design Science Research Methodology

Meeting an objective can be a process of stepwise refinement, performing an action, observing the outcome, then modifying the action based on that outcome, and repeating until the desired outcome is obtained. This is a feedback or goal-seeking process, such a process was required in this research. The goal-seeking was to establish the specific research question, design and implement software, execute experiments, and evaluate the results. To frame this iterative systems research the process model used follows the Design Science Research Methodology (DSRM).

DSRM provides a process model to guide systems related research. In this section, a summary of DSRM is provided, for full details refer to Peffers et al. [9]. DSRM is an established process, highly cited in Google Scholar. A search within those citations using the term *(automotive OR vehicle OR vehicular OR car OR truck) AND security* shows a few hundred results, indicating the use of the method in areas related to this research. For example, it is referenced in [62], which is an output of the HEAVENS project (HEAling Vulnerabilities to ENhance Software Security and Safety) which ran for three years between April 2013 and March 2016. HEAVENS was coordinated by Advanced Technology and Research within Volvo Group Trucks Technology. A summary of DSRM is below, and a description of how it relates to this research program follows.

3.3 The DSRM process model

The DSRM process model is shown in Figure 3.1. The six activities in the process are:

1. **Problem Identification and Motivation** - This covers the justification for the research, and what need the research is addressing.

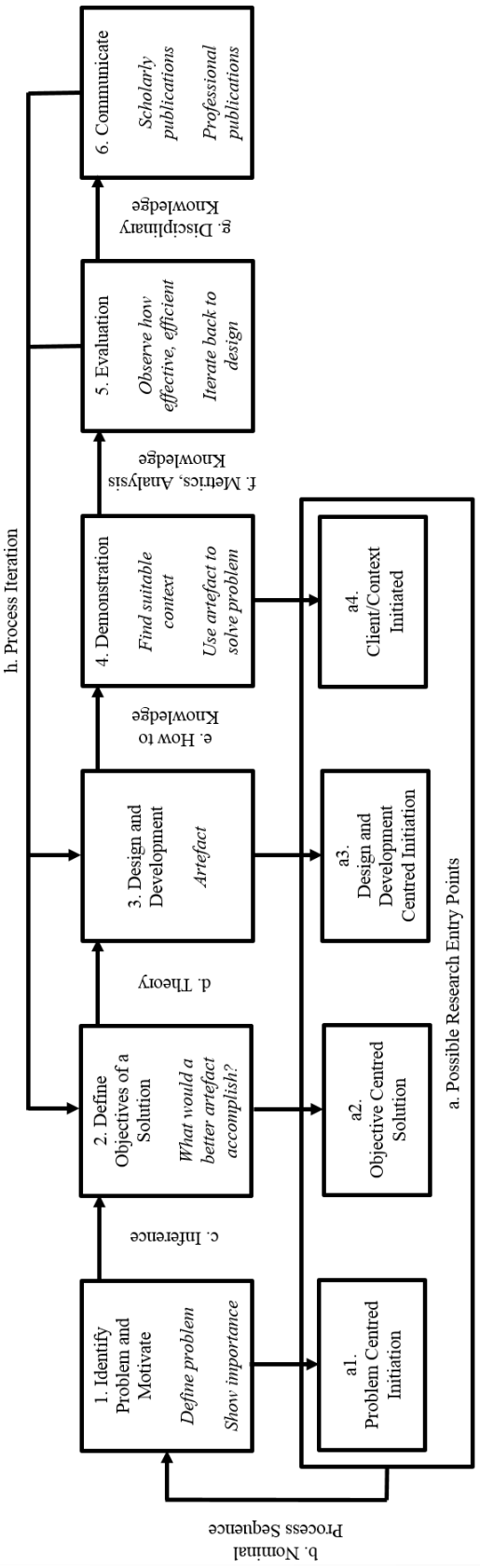


Fig. 3.1 The Design Science Research Methodology was redrawn from [9]

2. **Solution Objectives** - These are quantitative and/or qualitative features of the expected outcome. How the research is expected to solve problems or improve upon existing solutions.

3. **Design and Development** - An artefact is produced to solve the identified problem. Whatever the type of artefact constructed, e.g. a model, method, algorithm, program, device, system, etc., it encompasses the objectives.

4. **Demonstration** - The artefact is used to address the problem. This can be by experimentation, simulation (virtual experimentation), case studies, proofs, etc.

5. **Evaluation** - The effectiveness of the artefact in addressing the problem should be determined. Does the functionality meet the objectives? Does the quantitative data from usage meet expectations? Should the process iterate back to improve the outcomes?

6. **Communication** - The research results are published and made available. The artefact is made available for use. Feedback from the communication, for example, peer review comments, may inform the research and thus, the process may iterate back to objectives and/or design and development.

The DSRM is flexible in its approach. The entry point for the process (a. in Figure 3.1) may be determined by different factors:

1. A formally observed problem, or a problem statement, can start the process from the beginning.

2. Research, business, or industrial need can trigger the process from a specific objective, before the problem has been accurately stated.

3. An existing prototype artefact, maybe developed ad hoc as an interim solution, or an artefact designed for a different use case can trigger the research process.

4. Observations of solutions in other domains, or analysis of existing processes, e.g. from consultations, can begin the research process.

Entry points that do not start at the problem definition stage can use the DSRM to frame the research in a formal process and, thus, provide the justification for the desired outcome, i.e. follow the DSRM nominal process sequence (from b. in Figure 3.1).

The output of each DSRM process stage provides information to the following stage. The problem definition and motivation infers the objectives (c. in Figure 3.1). The objectives are the theoretical basis for the development of the artefact (d.). Knowledge of the artefact determines how it is applied or used (e.). The observations from using the artefact are evaluated using measured metrics and analysis (f.). Finally, the results become a body of knowledge and evidence on the artefact and the solution it offers to solve or reduce a problem. This is then communicated in publications and to stakeholders and interested parties (g.). The outputs from evaluations and the feedback from communications are likely

to trigger iterations in the research (h.), particularly during the early stages, when the information available is not granular enough and experiments are under development, as was the case with this research.

3.4 Applying the DSRM to this research

The aims of Peffers et al. with the DSRM [9] was to provide a formal definition of a method for applied information systems research that had new artefacts for outputs. They also wanted to provide a *mental model* to encapsulate the development process that an information systems researcher goes through before finally presenting their work. They acknowledge that such a formal definition for information systems research has been considered before and have used previous research to aid the design of the DSRM. In doing so the DSRM provides a solid framework to ensure that this research program is correctly addressing the knowledge gap in automotive CAN fuzz testing.

In the introduction, in Section 1.2, it was stated that there was an initial need to investigate cyber-security testing vehicles. This was the high-level objective that initiated this research program. From that entry point the DSRM clarifies this research program:

1. **Problem** - The literature review, in Chapter 2, identified a knowledge gap in automotive fuzz testing. In particular, the ubiquity of CAN is chosen as a target technology for fuzz testing. Thus, the problem is to determine if CAN fuzz testing is indeed suitable as a security test as a component in vehicle resilience testing.

2. **Objective** - Fuzz testing has been applied to vehicular systems and CAN, but the published results are sparse and without detail. This research concentrates on the straightforward generation of CAN fuzz testing data and results. To enable this objective a simple to deploy and configure CAN fuzzer is required, see Section 5.1.2 for further detail on the required aspects of the fuzzer.

3. **Development** - The CAN fuzzer, it's interfacing to ToEs, and the methods of usage from the experiments are the main outputs of this research. Chapter 5 discusses the software development of fuzzer.

4. **Demonstrate** - The fuzzer is used against a variety of ToEs in the experimental Chapters 6 to 9.

5. **Evaluation** - The results from the experiments are used to improve the prototype CAN fuzzer, its functionality, its methods of use and the effectiveness of CAN fuzz testing.

6. **Communicate** - This thesis is the main record of this research. There are also published outputs, see Section 1.5, and additional papers to be produced. The CAN fuzzer has been demonstrated to various stakeholders and the intellectual property contained in the software and outputs are stored in HORIBA MIRA's internal source code management system for access by their engineers.

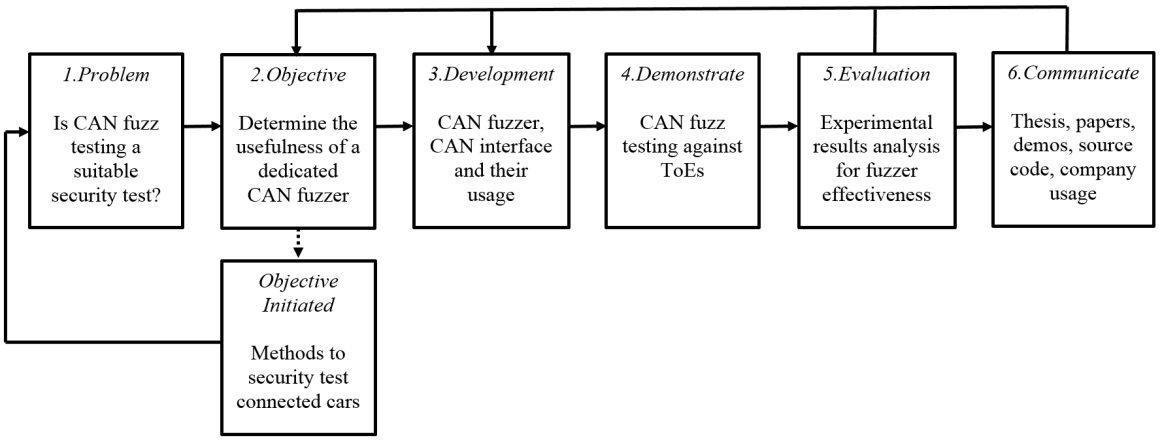


Fig. 3.2 The overall research framed in terms of the DSRM

The DSRM process of this research is summarised in Figure 3.2. DSRM acknowledges the often iterative nature of systems research. Development of new systems engineering methods and the supporting software often involves developmental iterations. In this research the main iterations were in the development and application of the prototype CAN fuzzer during the fuzz testing experiments in Chapters 6 to 9, see Figure 3.3.

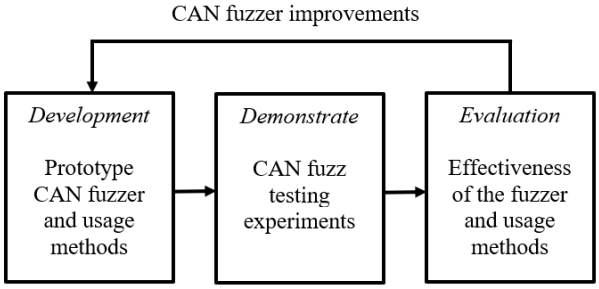


Fig. 3.3 Software development and experimental methods iteration

3.5 An automotive security test development methodology

The iterative process to establish the research question, develop the security test and tooling, and perform the practical experiments resulted in an overall methodology for this research program. It can be used for developing cyber-security resilience testing methods for automotive systems, and possibly other CPSs. A high-level overview of the methodology is shown in Figure 3.4.

There are seven phases to this cyber-security testing development methodology:

1. **Establishment and validation of a test environment** - The complexity and costs of the modern vehicle provide a challenging environment in which to develop new security testing methods. For a new vehicle design the access to car's internal systems and components may

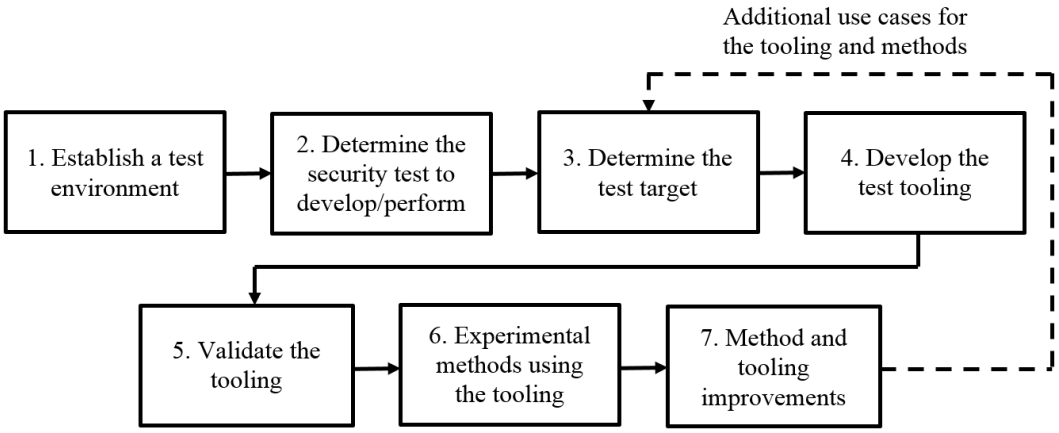


Fig. 3.4 A method to develop automotive security testing

not be possible in the early stages. The use of a testing environment, in the form of a reusable testbed, provides a controlled setting for developing new testing methods. In Chapter 4 a commercial HIL/SIL vehicle design and development system is deployed as a security testing rig. Such systems are traditionally used for functional testing, but this research is stating that they may be used for security testing.

2. **Determine a security test to develop or perform** - The literature review, Chapter 2 identified that three classes of security test can be executed against a vehicle system or component. Out of those three tests, fuzz testing is the most suitable for a manufacturer or supplier to execute themselves. The outcome of this phase is for the researcher to choose one of vulnerability, fuzz, or penetration testing. However, each of these three tests will have specific testing requirements based upon the technology of the ToE (see the next phase), for example, a penetration test for a Wi-Fi interface would differ in its details to one targeted at Bluetooth. In this research, the lack of available detailed knowledge in fuzz testing automotive systems was the reason to choose it over vulnerability or penetration testing.

3. **Choose a technology to test** - Vehicles contain multiple digital technologies. There are many types of components, interconnections, interfaces, and communication protocols. For this research CAN was chosen for its ubiquity within vehicle systems (CAN technology is discussed in Chapter 4 and Appendix B). However, many vehicle technologies (see Tables 2.4 and 2.5 in Chapter 2) require new research on security testing techniques.

4. **Development of the test tooling** - To execute the chosen security test against the chosen technology tooling support is required. In testing digital technologies this invariably means some form of software, but may include interfacing and measurement equipment. The software and equipment can be commercially available or may require bespoke design and development, or a mixture of the two. At the time of this research, no simple to use software tooling was available to support fuzz testing of CAN. Simplicity here is in terms of ease of use and

deployment. Chapter 5 discusses the development of a prototype software tool, known as a *fuzzer*, for use in CAN fuzz testing. It uses off-the-shelf hardware to interface to CAN. The testbed provided the controlled environment to enable the development of the tooling. It is anticipated that developing security testing tooling for other vehicle technologies will be required in the future.

5. **Validation of the tooling** - Any security test tooling will need to be performant for its intending application. Therefore, it needs to be validated. In this research, some validation of the CAN fuzzer was performed during its development (Chapter 5), with further validation during experimental use. A similar validation phase is required for security tooling used against other technologies.

6. **Experimental methods using the tooling** - In this phase, the established test environment and the new test tooling is put to use performing the determined security test and the test results are recorded. However, the lack of knowledge in security testing of vehicle systems means that the experimental methods will require refining as testing experience is gained. For example, the experimental use of the prototype CAN fuzzer revealed unforeseen issues. These are discussed in the relevant chapters, Chapters 6 to 9. Thus, as well as evaluating the results of the experiments, it is important to refine the experimental techniques used, this requires the next phase.

7. **Method and tooling improvements** - This phase is used to improve the effectiveness of the security testing methods and tooling against the targeted technologies. Use of the security tooling can suggest, or require, improvements for future experiments, security tests and use cases. This improvement is illustrated by the dashed line in the schematic in Figure 3.4. The CAN fuzzer development discussion in Chapter 5 includes improvements from using the fuzzer against ToEs. The near-continuous development of new and improved technology in the automotive field will require continuous improvements in security testing tooling and techniques.

This security test development method was used for the CAN fuzz testing processes and construction of the CAN fuzzer tool for this research. In Figure 3.5 automotive components and systems with CAN communications are the targets. It is plausible that the same method is suitable for developing security tests for other technologies and domains. For example, applying the security test development method for the Bluetooth interface, for smartphone targets, is shown in Figure 3.6.

3.6 Addressing the very large CAN state space

In fuzz testing the CAN bus, there is the problem of state space explosion. The combinatorial increase in state-space means that the search for possible problematic system inputs can be impractically

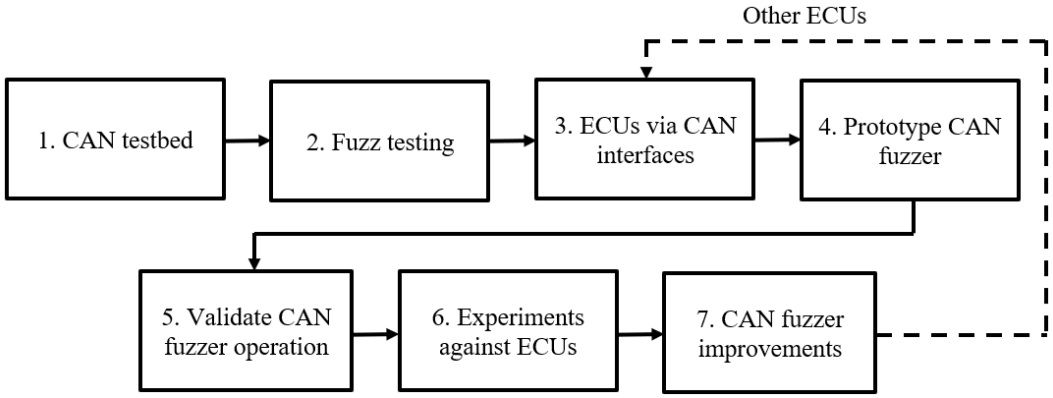


Fig. 3.5 An overview on applying the security test development methodology to CAN fuzz testing

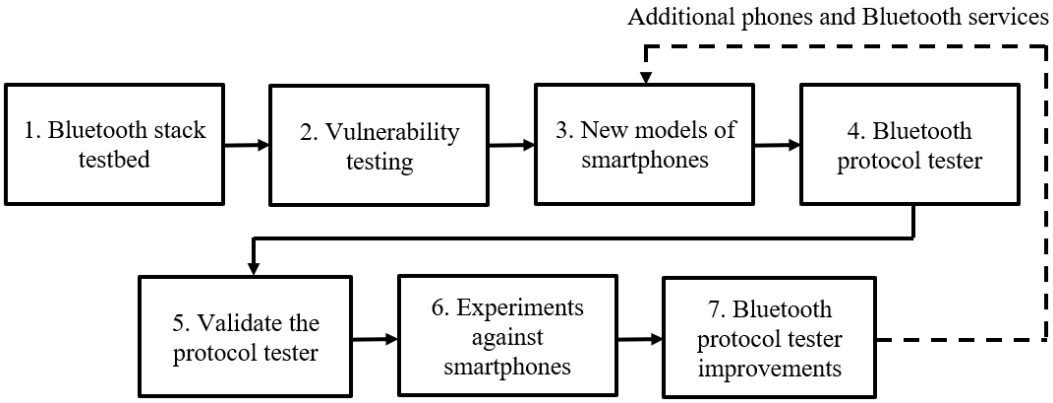


Fig. 3.6 An overview of the security test development methodology applied to vulnerability testing the Bluetooth stack on smartphones

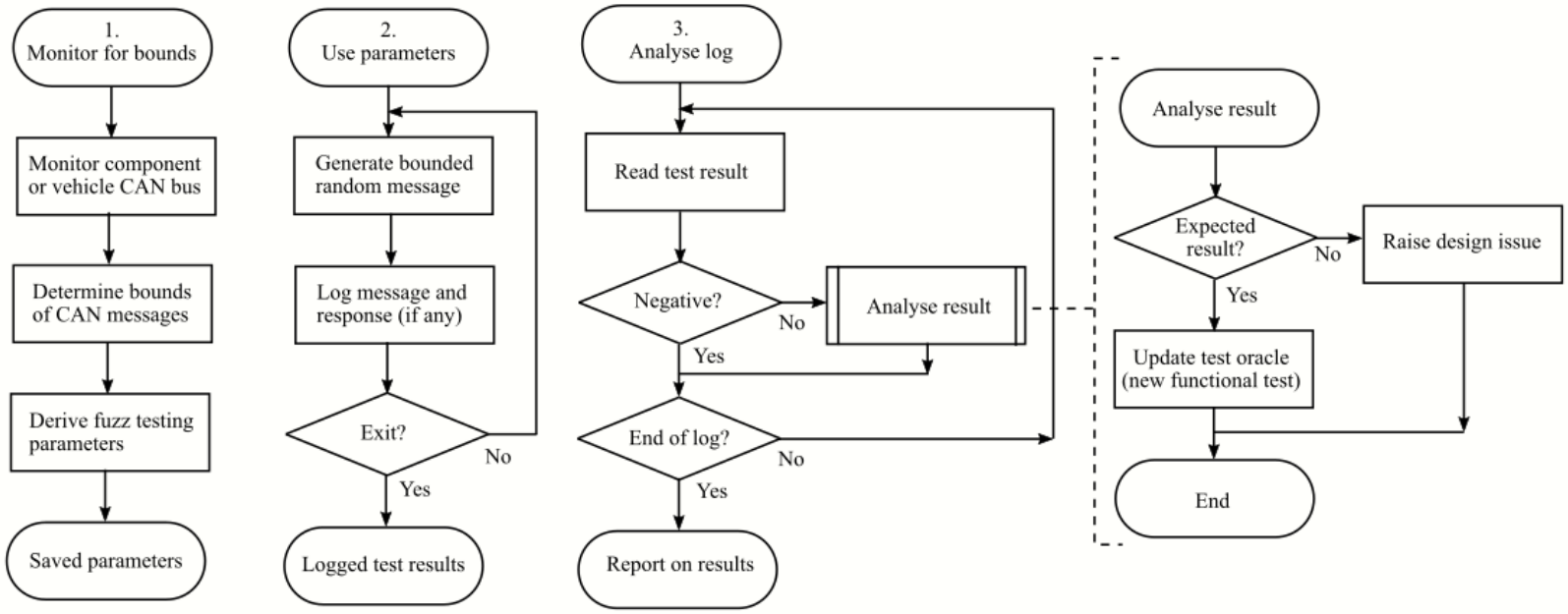


Fig. 3.7 A three-stage process for targeted CAN fuzz testing to address state space explosion

large. Methods and processes to reduce this search space are therefore required. In Figure 3.7 the methodology proposed is a three-stage process for targeted fuzz testing, which is outlined below.

1. The three-stage process begins by monitoring the normal activity of a CAN bus. This is then analysed to determine the upper and lower bounds of the CAN data flowing on the network. The data bounds may include all parameters of the CAN data, the id, length and data bytes (see Table 2.2), plus temporal information. The bounds can be adjusted to allow the fuzzer to generate values outside of the recorded values ranges.

2. The second stage is to use these parameters to inform the generation of the fuzzed CAN data packets. Packets are randomised within the bounds, transmitted to the network, and responses are logged. When the fuzzer exits (after all packet data combinations are exhausted, or after a pre-determined period) the logs are available for analysis.

3. When the test logs are analysed, the positive tests (packets that produce an unexpected reaction) are examined. Each positive test result will indicate either an incorrect expectation (in which case the test oracle must be updated), or unexpected behaviour, and thus the system design will need revisiting.

3.7 Equipment, tools and test facilities used during this research

This research program would not have been possible without the use of equipment, tools and facilities provided by Coventry University and HORIBA MIRA Limited. Details are provided in each experimental chapter. The following list summarises the equipment, tools and facilities used.

- The CCAAR (see the Acknowledgements) facility located at HORIBA MIRA was the location for the laboratory experiments. The CCAAR facility includes garage space and workbenches.
- A garaged vehicle, a small hatchback registered in 2013, was available as a laboratory car. Information on the lab car is not provided due to commercial confidentiality and a responsible disclosure policy. The lab car and components from it, are used in the experimental chapters. The instrument cluster is used in Chapter 6, the media interface ECU is used in Chapter 8, and the display ECU is tested in Chapter 9. The car and the instrument cluster are also used during the testing of the bit rate attack, discussed in Appendix C.
- A Windows desktop PC was used for software development and software testing.
- The Microsoft Visual Studio integrated development environment was used for software development.
- A Windows laptop PC was used for software testing and data collection.

- A Vector Informatik GmbH HIL/SIL system was used to simulate vehicle systems and its CAN busses. The Vector system consists of a CAN hardware interface and the CANoe simulation software environment. The system is used as an automotive cyber-security testbed in Chapter 4.
- PEAK-System Technik GmbH interface devices, called PCAN-USB, are used to connect Windows PCs to CAN busses. They were used for the development of the CAN fuzzer in Chapter 5, and in the experiments in Chapters 6 to 9 A PCAN-USB interface library was developed from sample code provided by PEAK-System to allow access to the devices from the developed CAN fuzzer software.
- An assortment of aftermarket devices, that connect to a vehicles OBD port, were used in the validating the testbed in Chapter 4.
- An assortment of tools and test equipment typically found in an electronics lab was used for cable construction, hardware debugging and data readings.
- An assortment of custom cables were constructed to facilitate interfacing with the lab car, ECUs and other equipment. The custom cables were used in Chapters 4 to 9.

3.8 Summary on research methodologies

This chapter discussed the DSRM process model. DSRM is established in systems research, including use in the automotive domain. The DSRM frames the iterative nature of software and experimental design in systems research with artefact outputs, which was the case for this research. Additional information on the practical methods used in the experiments are discussed in the relevant chapters, Chapters 4 to 9, including any additional materials, equipment and techniques used.

In performing the experimental work the overall research methodology was established. That method contains steps that allow for the development of automotive security tests. In the next chapter, Chapter 4, the method begins to be applied to this research program with the establishment of the test environment, the first step in the methodology given above in Section 3.5.

Chapter 4

A testbed for automotive security testing

Nobody actually creates perfect code the first time around, except me. But there's only one of me.

Linus Torvalds

The literature review, Chapter 2, discussed security weaknesses found in vehicular systems. Such weaknesses should be addressed for new and future vehicle models. Engineering security into products with good design and systematic security testing is advocated, e.g. in SAE J3061 [27]. This is instead of retrofitting security once the functional design is complete. In Section 1.2.1 the use of HIL/SIL equipment for vehicle systems development was introduced. The development process, when vehicle components and systems are being prototyped, is an opportunity to begin security testing.

To verify that automotive HIL/SIL equipment can be used for cyber-security testing an experiment was performed. This experiment provides a proof-of-concept that such equipment is indeed suitable for cyber-testing. The output of the experiment is the establishment of an automotive security testbed for evaluating both countermeasures and tools. For example, the prototype CAN fuzzer in this research (see the following Chapter 5) was developed and initially verified against the testbed.

The establishment of a test environment for vehicle cyber-security testing is the first stage of the automotive security test method, see Figure 3.4 in Chapter 3. This experiment was conducted collaboratively with Dr. Madeline Cheah, who was completing her doctoral program of research within the Systems Security Group at Coventry University. The publications [6], [63] derived from this work are listed in Section 1.5. In [64] the idea of a virtual environment for early cyber-security testing of vehicle systems is being continued, with their working referring to the publications from this research.

4.1 Introducing the experimental work

The verification method to use the HIL/SIL equipment as a testbed for security testing was in two stages. Firstly, to perform a cyber attack on a laboratory vehicle, then, secondly, to perform a similar attack against the HIL/SIL equipment.

An attack on a vehicle's internal communications network was performed wirelessly via an aftermarket *dongle*, a third-party device used to read vehicle ECU data values. The dongle is plugged into the vehicle's ODB port, which contains a connection to the vehicle's CAN bus. The attack method used is then deployed against the testbed, which has been adapted to accept a dongle device. The similar nature of the testbed attack to the real-world attack demonstrates that security testing can be performed early in the development lifecycle.

The vehicle simulation used in the experiment is not as complex as a fully-featured vehicle, but was adequate to demonstrate that cyber-security testing can be performed early in the R&D lifecycle. The experiment provided the confidence to use the testbed for automotive fuzz testing development, and demonstrate that it is useful for developing cyber-security testing algorithms and tools, prior to use against vehicle components and systems.

4.2 Experimental method

Executing the experiment required the following:

- Development
  - Obtain a suitable HIL/SIL platform
  - Determine an attack to perform
  - Implement the attack
- Demonstrate
  - Validate the attack against a vehicle
  - Perform the attack against the HIL/SIL equipment
- Assess the use of the HIL/SIL equipment for use as an automotive cyber-security testbed.

4.2.1 HIL/SIL platform

The complexity of automotive electronic systems has long required the development of specialised equipment that can provide a comprehensive environment for out-of-vehicle testing. Commercial test equipment companies provide HIL/SIL rigs that emulate vehicle functionality to enable unit and integration testing of electronic systems. Such commercial automotive test rigs used to require large

cumbersome racks but modern units can be used on a desk with much of the vehicle functionality simulated in software on a single PC, as done with this experiment.

The test equipment used is a CAN simulator from Vector Informatik GmbH. It is commonly used in vehicle systems design and development. It was chosen as a testbed because it has:

- High powered hardware and software capable of running a multiple ECU simulation. The simulated components reasonably and reliably emulate real-world behaviour.
- Functionally accurate networking and communications aspects.
- Interfaces to allow for external connections to the internal simulation.
- Adequate network traffic monitoring and analysis capabilities.
- Data capture and playback capabilities. This feature is valuable to investigate the repeatability of experiments.

The testbed has CAN interfaces allowing it to be connected to external CAN buses. A comprehensive PC based software package called Vector CANoe works in conjunction with the hardware. The CANoe software allows for the design and simulation of ECU networks. Emulation of a vehicle's network of CAN buses can be full or partial, allowing for a mix of physical hardware and simulated network nodes (HIL configuration). The simulator is capable of running a virtual vehicle, with the vehicle controls operated via a GUI. Using the testbed as a development platform brings the following advantages:

- It provides an isolated and controlled environment for developing and testing new algorithms for security tests, and the software and tools to execute those algorithms.
- It allows new security testing processes to be developed and prototyped prior to deployment against vehicle components and systems.
- It reduces the risks involved in security testing vehicles by allowing developed procedures to be tested initially on the simulator, reducing the cost of using a real vehicle, and the risk of damage to it.

4.2.2 Attacking the HIL/SIL platform

The security test to perform needed to be self-contained, straightforward to implement, suitably time-bound (a short development and execution timescale), non-destructive to any shared resources, and fall within ethical considerations, i.e. legal, in terms of computer misuse and personal data laws, and not causing harm to people or property. From the three classes of security testing (vulnerability, fuzz and penetration) vulnerability was chosen. The vulnerability test performed was the malicious injection of CAN data packets into a vehicle via an aftermarket device. The accessibility of this attack made it suitable to try against the HIL/SIL equipment, necessitating these steps:

Fig. 4.1 Examples of wired and wireless ELM OBD dongles, used to read and write data via a vehicle's CAN bus

1. Obtain a suitable aftermarket device that connects to a vehicle's internal CAN bus.
2. Perform packet injection on a target vehicle to verify the known vulnerability.
3. Connect the aftermarket device to the testbed.
4. Perform packet injection on the test platform.

4.2.3 Aftermarket device threat assessment

There are several use cases for aftermarket devices that connect to a vehicle's OBD port (see Table B.3 in Appendix B). One type of device is known as ELM327, named after a popular microcontroller (MCU) this is used to interface to the OBD port. Examples are shown in Figure 4.1. In essence an ELM device provides a serial terminal connection from a mobile phone or computer to the vehicle, see section B.4.1 in Appendix B. This enables software to not only read vehicle data but to transmit data into the vehicle. This functionality comes with few, if any, security mechanisms.

Five OBD dongles (Table 4.1) were connected in turn to the lab vehicle. Each dongle was used to test packet injection into the vehicle's internal CAN bus, Figure 4.2. These devices ranged from the extremely cheap to more featured (and therefore more expensive) tools.

The first observation of interest was that several dongles had an ELM chip version of 1.5. This is a non-existent version [65] as the version after ELM 1.4b was ELM version 2.1. This could be indicative of a counterfeit chip, but it did not affect the experiments. However, the probable counterfeit ELM chips were not able to accept the full ELM AT command set nor did they correctly implement some of the commands.

The second observation of interest was that every dongle except for the OBDLINK MX powered up and started broadcasting its address as soon as it was connected to the vehicle's OBD port. This

Table 4.1 OBD scanning devices (dongles) investigated, most had a fixed PIN of 1234 and the reported ELM version may indicate the use of a counterfeit MCU

Dongle (OBD Port Device)	*PIN*	*Discoverable*	*Price*	*ELM Version*
Vgate Advanced OBD2 Bluetooth Scan Tool	1234	Always	$13	v2.1
Exza OBD SCAN	1234	Always	$25	v1.5
Vgate ELM327 Mini	1234	Always	$15	v1.5
Pumpkin OBD2 ELM327 Bluetooth Car Scanner	1234	Always	$15	v1.5
Scantool OBDLINK MX Bluetooth	dynamic 6-digits	button press	$100	v1.3a

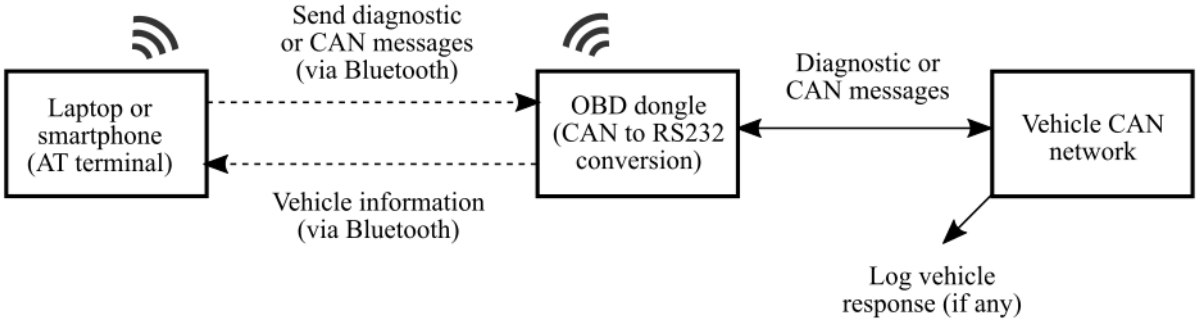


Fig. 4.2 Vulnerability test against a CAN bus via OBD dongles

Table 4.2 Commands sent to the OBD connected dongle

Command	*Purpose*
`ATI`	Information - to find the ELM chip version
`ATSP6`	Set Protocol 6 - sets the operating protocol to one that denotes the standard to which this particular CAN bus adheres to (in this case, ISO 15765-4 for diagnostic communication over CAN)
`ATMA`	Monitor All - a configuration switch that allows monitoring of all traffic on all exposed CAN buses
Mode & PID	Mode and Parameter ID - in the form of two or more hexadecimal bytes
`ATSH xyz`	Set CAN Header id - sets the CAN packet id to xyz for subsequent packets

was true regardless of whether the ignition was turned on. Even with the relatively small generic Bluetooth Class 2 range of ten metres, this is a clear security risk, especially with fixed PINs and the discoverable mode permanently enabled. Security in this regard is improved with the more expensive scan tool, where the dongle has a two-minute discoverable window.

The third observation was that the cheap dongles used an insecure [66] legacy pairing mechanism, with the PIN being `1234`. The more expensive scan tool asked for a comparison with a dynamically generated six-digit PIN on the phone; however, the dongle contained no input and so just selecting *confirm* was enough to pair.

There is communications encryption and authentication, via the Bluetooth standard, between the serial terminal (computer or cell phone) and dongle. However, the security is greatly diminished by the fact that the PIN is short, fixed and easily determined (in the case of `1234`) or just requires a confirmation regardless of the PIN generated. Furthermore, once a phone or other device is paired, it remains trusted by the dongle indefinitely and will pair automatically once within range.

4.2.4 Configuration and diagnostic messages

Once the devices were paired, a series of AT messages are sent in order to configure the dongle and monitor the traffic, from whatever CAN bus was exposed on the OBD port. Appropriate messages were then sent via the dongle through a simple terminal program (running on a Bluetooth enabled laptop or cell phone). The commands used are summarised in Table 4.2.

Modes and Parameter Identifiers (PIDs) are used as a tool to perform diagnostic functions or request data from the vehicle. The first ten modes (`01` to `0A`, described in SAE J1979 [67]) are standard PIDs and generic to all compliant vehicles. Even so, there are privacy implications as, for example, sending the mode `09` with PID `02` retrieves the Vehicle Identification Number (VIN). The VIN is unique to the vehicle and is used for many activities, from vehicle maintenance to the recovery of a stolen vehicle.

The modes and PIDs used in the real-world demonstration are non-standard, with such PIDs being defined by individual vehicle manufacturers. This seeming obscurity does not negate the danger as

Table 4.3 Results of sending one type of manufacturer diagnostic PID message to the lab vehicle

State of vehicle	*Observation*
Ignition off	Sending the mode and PID combination once returned `NO DATA` from the vehicle
Ignition off	Sending the mode and PID combination in a continuous stream caused the key fob to stop working (door would not lock), ignition button is non-responsive, vehicle beeped continuously, vehicle still returned `NO DATA`
Ignition on	Sending the mode/PID combination once caused the electronics (lights, instrument cluster, blowers, boot lock and radio) to flicker once
Ignition on	Sending the mode and PID combination in a continuous stream caused the electronics to flicker continuously, the ignition button was unresponsive (engine would not start)
Ignition on / engine on	Sending the mode and PID combination in a continuous stream caused the electronics to flicker on and off and the engine to cut off intermittently. The engine malfunction light came on until next ignition cycle

many diagnostic functions are dangerous, and one can easily run through every combination of mode and PID to see if there are any adverse physical reactions from the vehicle. Not every diagnostic message elicits a physical response, however, anything that comes back from the CAN bus could be used in future reverse engineering efforts.

The results of trials with one non-standard diagnostic message that did produce a physical response are summarised in Table 4.3. The experiment was performed by sending the message both within and outside the vehicle cabin (within a one-metre range) with the same result, regardless of whether the chip contained within the dongle was a probable counterfeit (stating ELM version 1.5).

4.2.5 Raw CAN packets

The testing of directly injected non-diagnostic (raw) CAN packets was performed. Although the CAN database for a vehicle is usually confidential, packets that have an effect on the vehicle can be reverse-engineered given time and access. Here, a predetermined CAN packet was used, setting the door state using CAN id 534 (0x216 hex) with data `00 DC 02 00 00 00 00 00` (hex). It had been reverse-engineered previously using a wired OBD connection.

The injected raw CAN packets are transmitted through a serial terminal. The process starts with sending `ATSH xx xx xx`, where `xx` is a hex byte, in order to set the CAN packet id that the ELM chip will use when transmitting (the packet id effectively addresses the transmission to a particular ECU). Then an 8-byte payload was sent through the terminal program. The packet sent was a valid one, and although there was no physical response from the vehicle, the data was accepted, as it did not return a '`?`' or a '`7F`' hex value (signifying a negative acknowledgement) within the response.

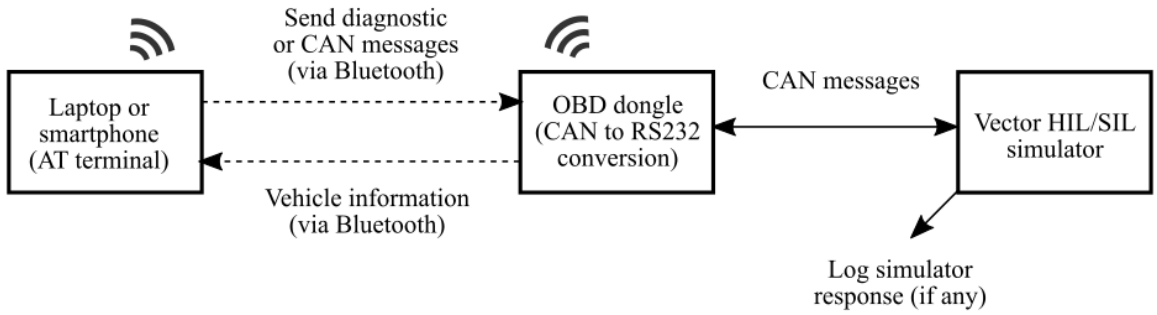


Fig. 4.3 Vulnerability test against a vehicle HIL/SIL testbed via an OBD dongle

4.3 A vehicle security testbed

From the results, it can be concluded that security testing for vehicles is essential, as injection of CAN packets could result in undesired behaviours. The security risk is also increased because the injections can happen from outside the vehicle. Thus, the design and implementation of a testbed is proposed to aid security testing. The testbed addresses issues surrounding security testing, including issues of cost and safety that might appear from testing a real-world vehicle.

A customised OBD cable was constructed to connect an OBD dongle to the simulator. These OBD dongles require power that represents the approximate 12 volts normally supplied by a vehicle. To enable this on the test bench the OBD power wires were separated and connected to a bench power supply (the CAN simulator does not supply vehicle voltages). There are a variety of OBD extension cables and splitter cables available and in this case, an OBD splitter cable was modified by adding the DB9 connectors accepted by the simulator (DB9 is a common format nine pin connector). This modified cable is used to connect any OBD device under test to the simulator. Figure 4.3 summarises the setup of the testbed used to test the OBD aftermarket devices.

In Figure 4.4 the testbed is shown in operation on the bench, with the modified OBD cable connected to a bench power supply on the left. The VGATE Advanced OBD2 Bluetooth Scan Tool (see Table 4.1) is connected to the OBD cable. The other end of the cable is connected to the CAN simulator in the middle. A computer running the Vector CANoe software, used for the simulated vehicle, is shown to the right. The serial terminal device (computer or cell phone) communicating with the OBD dongle is not shown.

The operation of the CAN simulator is based around descriptive databases that provide a virtual model of vehicle systems. To fully validate the testbed a model from a database of a series production vehicle needs to be run on the simulator. A threat assessment against the model can then be performed. The commercial sensitivity of vehicle software, ECU and CAN data packet design means that it is extremely difficult to get real vehicle ECU databases for testing and research purposes. Such databases that have been provided to organisations are usually subject to strict non-disclosure agreements. Even where some information is provided, it is generally in the form of partial databases for specific ECUs or use case functions, such as steering or braking, depending on the need.

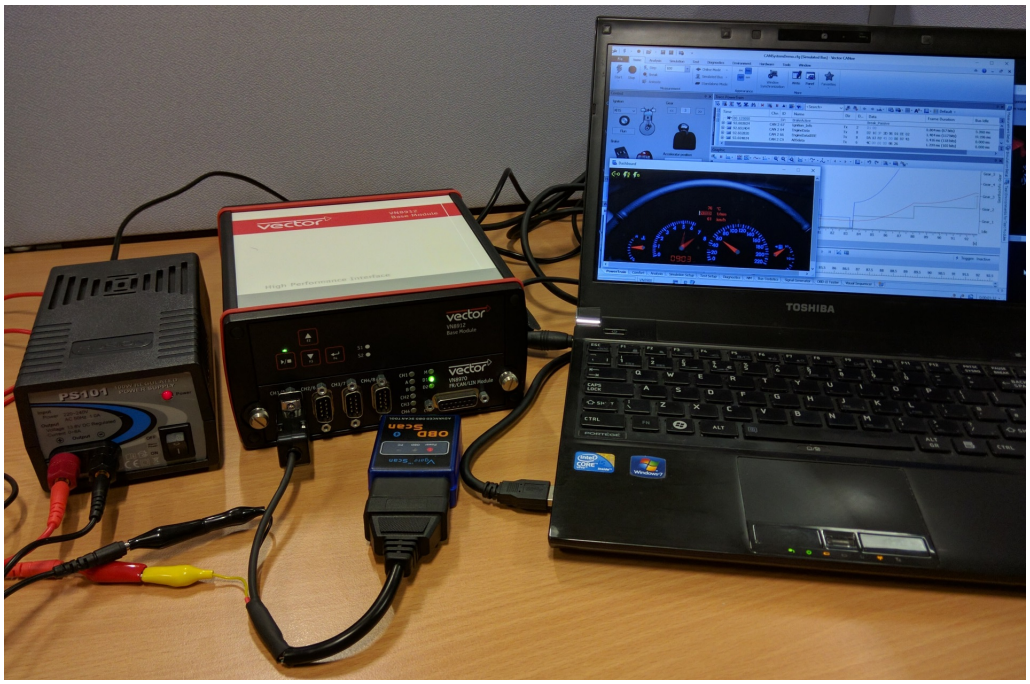


Fig. 4.4 A vehicle HIL/SIL design system has been used for security testing (a power supply representing the vehicle power is on the left, the CAN simulator is central and connected to a PC running Vector CANoe, a modified OBD cable connects to an aftermarket OBD dongle)

An alternative is to generate databases through reverse engineering. This is achieved by monitoring and recording packets on the vehicle network as all vehicle functions are systematically performed. The actions with the vehicle can then be mapped to the recorded bus packets to determine which packets are related to functions performed. This is sometimes impractical as data from a function can be spread across different CAN packets and furthermore, not all data from a vehicle is generated from an occupant accessible action. However, this practice is common for studies involving the evaluation of intra-vehicular networks [2], [22], [23] where the original manufacturer's CAN bus communications database is not available.

Since such databases are difficult to acquire, therefore, a generic database provided by the simulator vendor to emulate a vehicle is used. This database is limited in that it would not have the granularity one would expect of a full database. However, it is envisioned that a manufacturer (who are one of the target end-users of this setup) would be able to use their own vehicle communications databases in order to carry out security testing and analysis. The architecture of the model vehicle is shown in Figure 4.5.

4.4 OBD attack against the testbed

Having set up a simulation model, with a representative network of ECUs and emulated CAN traffic, the attack scenario was configured. Performing an unauthorised pairing with a Bluetooth-enabled

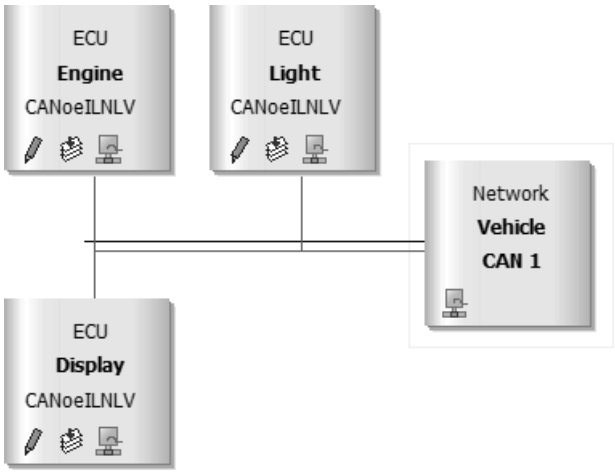


Fig. 4.5 Simulation setup as shown within Vector CANoe, the icons give access to CANoe parameters

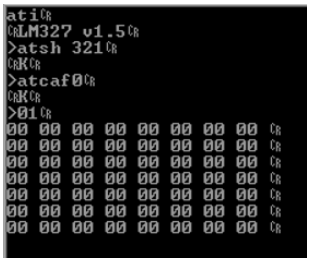


Fig. 4.6 Terminal program window showing `01` (hex), for LightState on, being sent, with CAN id `321` (hex). `ATCAF0` is used to disable auto-formatting.

dongle that is connected to the OBD port of a simulated vehicle. This should result in the injection of a CAN packet that affects an aspect of the simulated vehicle in an undesirable manner.

First, it was shown that a packet can be injected via the aftermarket device into the model system (highlighted in Figure 4.7). This was performed both through a phone and a laptop after having paired with the dongle. Figure 4.6 shows some of the configuration AT commands along with the actual CAN packet that sent via a serial terminal running on the laptop or cell phone.

Next, it was demonstrated that the simulation responds with the undesirable behaviour of turning the headlight on (Figure 4.8). Note, that although the headlight is turned on (the highlighted portion to the left of Figure 4.8), the internal state of the ECU insists that the LightState is *off* (the first byte being `00`). This required overriding with a continuous stream of *on* packets, causing the headlight in the simulation to flicker.

This is similar to the real-world demonstration, where sending the packet once caused the vehicle's electronics to flicker once. This could be caused by the fact that the injected packet had to contend with continuously generated 'true' packets from the attendant ECU. This effect has been observed by other researchers in the automotive cyber-security field. A mechanism to counter this could be a form

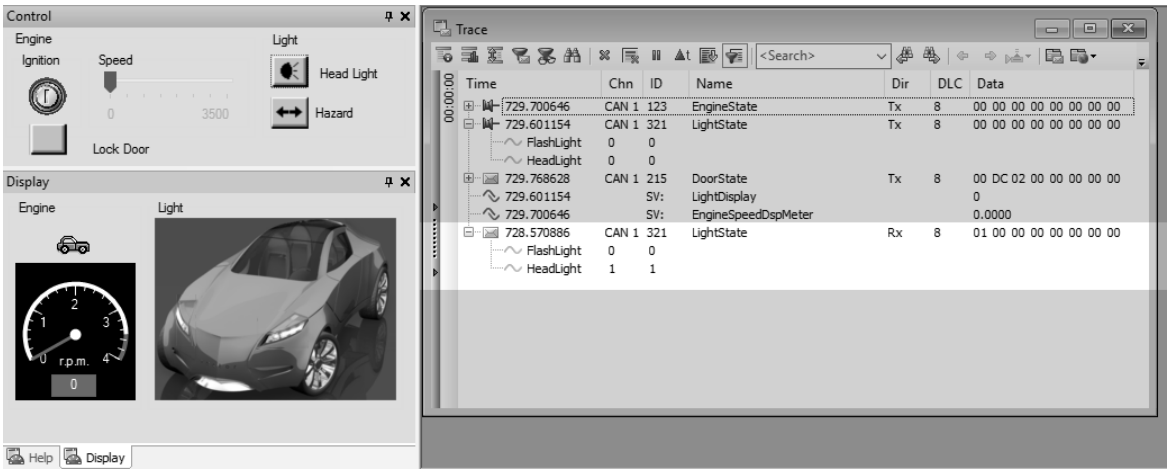


Fig. 4.7 Screenshot of Vector CANoe Trace, Control and Display windows. The highlighted portion is the packet of `01 00 00 00 00 00 00 00` on CAN id `321` (actually a shortened form of `00 03 21`) injected onto the simulated CAN bus. This trace window allows monitoring of the ongoing workings of the simulation and the packets sent on to the CAN bus, both by the simulation and injected from the OBD device

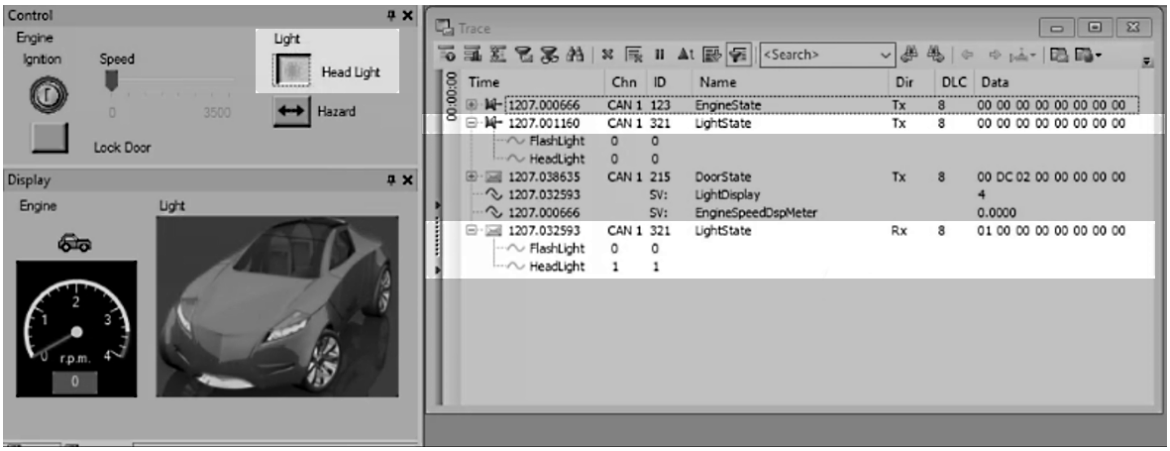


Fig. 4.8 Injected packet causing the headlight to turn on

of CAN traffic suppression. Traffic suppression can be performed by manipulating the individual bits in the CAN data packets flowing on the CAN bus.

Additionally, only the headlight was configured to respond. However, by creating and linking other nodes together within the simulation it would be possible to replicate the real-world demonstration that caused all electronics to malfunction.

4.5 Evaluation of the results

The implementation of the testbed fulfilled the testbed requirements. The networking aspects are accurate; the laptop or phone with a serial terminal is able to send information over Bluetooth to the OBD dongle, and the OBD dongle, in turn, transmits the packet via the customised OBD cable to the simulator. The simulator is able to see, log, and capture any information going through this dongle, and monitor and capture the effects that packets have on the simulated system.

The case study demonstrates the violation of a number of security principles as shown in Table 4.4. This compares directly with the security property violations within the simulated setup. The testbed is performant for cyber-security testing use, being comparable to what happens in the real-world, although acknowledging that the case studies used are simple. However, relevant modifications can be made to the simulation setup to add the required complexity. This can include adding different interfaces other than OBD, testing different protocols (for example FlexRay), and including more nodes within the simulation to allow a more realistic model of a fully functioning production vehicle. Extending the use cases for the testbed is particularly relevant when considering the progressive migration towards advanced driver assistant systems (ADAS), autonomous vehicles and different technologies such as automotive Ethernet [68].

4.6 Assessment of the testbed

The barriers to fully virtualised advanced vehicle systems are in the same vein as the ones found whilst performing the experiment. As pointed out in the literature review, Chapter 2, the information on the internal workings of vehicles is not usually available due to commercial sensitivity, plus, the cost of acquiring advanced vehicles is prohibitive. Future research in formal modelling of vehicle systems is a possible route. Formal models can be used to provide simulation models to be used on the testbed via the Functional Mock-up Interface (FMI) that the Vector tooling currently supports.

At the security engineering level, future work will look at using the testbed in several beneficial ways for automotive cyber-security research. This includes verification of known or emerging attacks in order to reliably reproduce them for study and development of mitigation methods. Previously proposed or new cyber-security solutions for vehicular systems can be implemented on the testbed in order to validate their claims and test their run time and security properties in a realistic and controlled environment. Solutions may be for a particular system design or model, a software algorithm, encryption scheme, communications protocol, optimisation or hardware configuration.

Table 4.4 Security properties/functions and their violation via the real-world and simulated testing

Principle	*Real-world (device on vehicle) testing*	*Testing on the CAN simulator*
Confidentiality	Information about the vehicle (via the CAN bus) is disclosed once the Bluetooth connection has been made to the OBD dongle, with the ability to monitor CAN bus traffic as well as being able to request vehicle information, e.g. the VIN number	Information about the vehicle (via the CAN bus) is being disclosed once the Bluetooth connection has been made to the OBD dongle, through being able to monitor the CAN traffic
Integrity	The state of the vehicle changed when the electronics flicker from 'on' to 'off', and when the 'Engine Malfunction' light appears	The state of the vehicle is changed temporarily, from headlight 'off' to 'on' and vice versa
Availability	Sending a continuous stream meant that the electronics, ignition button and locks stopped working, or worked only intermittently	Sending a continuous stream caused the headlight to flicker, meaning that functionality is compromised
Authenticity	The origin of the packet is from an unexpected (non-authenticated) source	The origin of the packet is from an unexpected (non-authenticated) source
Non-repudiation	Neither vehicle nor the owner can provide 'proof of innocence', defined by [69] as proof that a malicious command was received, or that such a packet was not sent	Vehicle simulation cannot provide 'proof of innocence' although this could be incorporated into later simulation models should this be a principle to be tested against
Freshness	A CAN packet tested on the real vehicle contained a counter, but this did not seem to affect whether the vehicle considered the packet valid. In this sense, the communicated data is not recent and is simply replaying old data	The CAN packet sent is the same, with the same effects regardless of the number of times it was sent. Again, counters, rolling codes or other countermeasures could be incorporated into later simulation models to be tested against

Product solutions can be analysed in detail to verify or refute any claims. Solutions can be examined for functional performance including system initialisation timings, relative performance, signal latency analysis, packet transmission performance, network contention issues and execution issues. The solutions can include new security testing methods, as, in the case of this thesis, the development of automotive fuzz testing.

An additional enhancement for the testbed would be to add appropriate gateway modules to prevent the security principles (Table 4.4) being violated. These modules would, in effect, act as firewalls to filter out packets that could provoke undesirable behaviours. Such devices are already being marketed and vehicle manufacturers will require a method to test the validity of a device's claims.

The thesis objective of a security testing methodology, via fuzz testing, is aided with access to a useful vehicle systems testbed. Having validated the suitability of the Vector HIL/SIL equipment and its software for automotive cyber-security use, the testbed provides an environment to allow for development of the tooling (a fuzzer), and experimentation to develop the processes to apply it. It is envisaged that the testbed could be used for teaching and training, as a tool that can help lower the barriers for future automotive cyber-security research.

Finally, the argument that the use of a HIL/SIL system is beneficial to moving security testing to earlier in the development lifecycle has been pointed out in recent work [60], after the results from this chapter were published (see Section 1.5 in Chapter 1 for the publication outputs).

Chapter 5

A new automotive CAN fuzzer

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

C. A. R. Hoare

The literature review, Chapter 2, identified the following:

- there is a knowledge gap in the application of fuzz testing to automotive systems development;
- fuzz testing is one type of security test that manufacturers can implement and execute;
- there is a sparsity of information on fuzz testing within the automotive security testing field;
- CAN is a ubiquitous protocol for interfacing and interconnecting ECUs;
- a fuzzer is a software tool that is used to perform fuzz testing.

Therefore, fuzz testing is chosen for the second stage of the security test development method shown in Figure 3.4 in Chapter 3. Furthermore, CAN is the determined test target for stage three of the method. This chapter now covers stage four and five of the method. It discusses the construction and validation of the tooling, a specialised CAN fuzzer. The major features of the fuzzer's design are described. The security testbed, developed in the proceeding Chapter 4, is used during the fuzzer development and initial validation. The automotive fuzz testing experiments in Chapters 6 to 9 were performed with the built prototype fuzzer.

5.1 Introduction to the construction of a CAN fuzzer

The objective of fuzz testing is to dynamically test a system for weaknesses. It may then be possible to use a discovered weakness to find a system vulnerability that, ultimately, violates the system's CIA security properties. Whilst finding a system weakness is a step towards exploiting a vulnerability, it has already been shown that fuzz testing itself can violate a systems availability property by causing a denial-of-service[2]. Therefore, a CAN fuzzer can also be used to test a system's response to a CAN DoS attack.

5.1.1 The need for a dedicated CAN fuzzer

There are few existing options with regards automotive CAN fuzzers available to researchers, see Section 2.13. The currently available open source and commercial fuzzers, Table 2.6, are complex programs. They have been developed over many years to support many types of non-automotive protocols and use cases. Some of the technologies that those fuzzers support have been deployed in vehicles, for example, Wi-Fi.

Those general purposes fuzzers can be configured and adapted for use with the specialised technologies used within the automotive field, however, it requires configuration knowledge. The required knowledge also extends to how to install and set up the fuzzer software. Open source fuzzers usually require Linux based machines, whereas automotive businesses often use systems based upon Microsoft Windows machines (the HIL/SIL system for the pre-production security testbed in Chapter 4 requires Windows-based computers). In addition, there are insufficient peer-reviewed detailed results on the effectiveness of the existing generic fuzzers used against automotive systems. Therefore, there was motivation for the development of a specialised CAN fuzzer tool:

- It addresses a lack of dedicated fuzzer tooling for the automotive field. Tooling is required to enable the study of fuzz testing of vehicle systems.
- It was designed to provide a minimal learning curve and easy deployment, allowing for researchers and engineers to start fuzz testing experimentation quickly.
- It allows for the publication of new results in fuzz testing vehicle systems. The lack of practical and detailed knowledge needs to be addressed, particularly the lack of peer-reviewed output. The limited knowledge that does exist is at too high a level.
- It provides a basis for the development of additional fuzz testing use cases within automotive security testing R&D.

To research CAN fuzz testing, and the experimentation to support that research, a prototype fuzzer tool was constructed and used against ToEs. The fuzzer can be used as the basis for further enhancement by engineers in the automotive field, and other transportation and industrial domains.

5.1.2 Required CAN fuzzer aspects

In terms of the vehicle technology required to be supported by a fuzzer, the literature review identified CAN as a common weakness in existing vehicle security issues. CAN is also the dominant in-vehicle network communications bus. Therefore, a fuzzer to investigate weaknesses in the CAN protocol was the aim. With this in mind some desired characteristics of an automotive fuzzer can be stated:

- The fuzzer should be easy to deploy (install).
- Configuration of the fuzzer should not be onerous.
- The configuration of the tests should provide a wide range of options, allowing for variation in one or all of the CAN data packet elements (see Figure 2.5 and Table 2.2 in Chapter 2), and over a full or restricted range of values.
- A fuzzer tool should fit into the existing testing infrastructure, therefore, it is advantageous for the fuzzer to run on a Windows PC.
- The fuzzer needs to work with a readily available interface for the CAN physical network.

Having stated the objective of developing a fuzzer for vehicle technologies, what is the problem it is addressing? Here, it is to test the viability of fuzz testing in the automotive field. Furthermore, it is to aid the generation of knowledge on automotive fuzz testing, via the provision of a prototype tool for use in research. Researchers and engineers gain experimental fuzz testing experience, using a dedicated CAN fuzzer that reduces the complexity with regards to configuration and test execution.

5.2 CAN fuzzer design and development

The starting point was to construct a fuzzer to generate the standard CAN packet format (see Figure 2.5 and Table 2.2 in Chapter 2). The fuzzer executes on a Windows PC (a common platform within commercial engineering). The PC based fuzzer is transmitting CAN data packets into a ToE, and captures the CAN bus traffic to enable monitoring of the ToE, see Figure 5.1. In this case, the ToE is either a vehicle system, subsystem or component that communicates via CAN.

The CAN traffic that is captured can be written to a log file. All the traffic that flows on the network can be logged. That is the packets sent from the fuzzer, and packets that originate from other CAN bus nodes. The fuzzer is dedicated to monitoring the CAN bus, however, a CAN transmission may not have a network-related response. Indeed, most CAN transmissions are sent in response to changes in external sensors and vehicle controls, and the receiving ECUs act upon those packets to alter external, real-world systems. Thus, network monitoring is just one part of CAN bus traffic. The CPS nature of the vehicle will require additional real-world monitoring solutions

The design of the fuzzer is influenced by the technology that it is fuzz testing, see the following Section 5.2.1, and by the experiments for which it is deployed. Thus, the fuzzer's design was

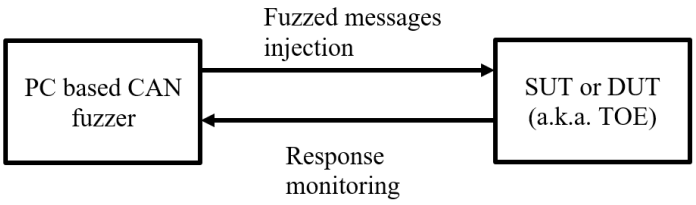


Fig. 5.1 The arrangement of the fuzzer

influenced by the needs of the experiments, and the possible experiments are influenced by the capabilities of the fuzzer. However, the fuzzer and the experiments are part of a process towards a final aim, which is to contribute automotive security testing methodologies, implement the methodologies, plus, have a software tool, the fuzzer, available to aid automotive cyber-security testing.

5.2.1 A PC based fuzzer

From a software viewpoint, the CAN data is a straightforward construct. The CAN interface hardware (containing the CAN transceiver) automatically handles the CAN protocols at the bit level during data packet transmission and receiving. This includes performing the CRC, checking for protocol errors, and checking for the correct arbitration (id) process. This leaves the higher-level software only needing to handle the packet identifier, data length and data bytes.

- Identifier (id), 0-2047 (11-bit) is used in standard CAN, though a value of 0-2^{29}-1 (29-bit) can be defined if an extended CAN id is required (not commonly used in vehicles).
- Data Length Code (DLC), a value of 0 to 8 for standard CAN.
- Data bytes, from 0 to 8 (defined by the DLC), bytes can range in value from 0 to 255.

The fuzzer was developed on a computer and connected to the testbed validated in Chapter 4 (see Figure 5.2). The use of the testbed simplified the development cycle as access to the target vehicle was not required during the development process.

The software for the fuzz test is written in the C# computer programming language. The Integrated Development Environment (IDE) used is Microsoft Visual Studio. The major functional items for the software fuzzer program are the User Interface (UI) screens for command and control, a timing thread for regular CAN data transmission, a random bytes generator for the fuzzed CAN data packets, a communications module, and a CAN bus traffic monitor.

5.2.2 Link to the ToE via the vehicle data bus or ECU interface

A USB to CAN hardware adaptor, Figure 5.3, is used to connect the fuzzer software on the PC to the CAN bus. It is a PCAN-USB product manufactured by PEAK-System Technik GmbH. The USB to CAN adaptor requires a 9-way D-type socket wired to conform to the CANopen specifications

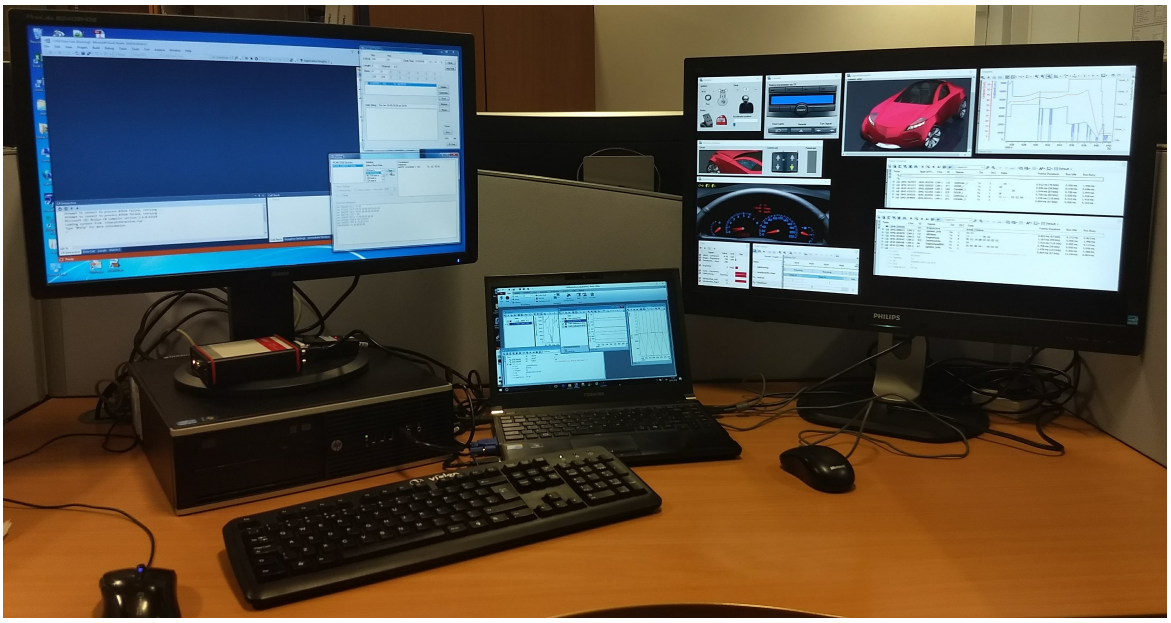


Fig. 5.2 The CAN fuzzer development environment using a PC running Visual Studio connected to a Vector simulation

(CiA303-1)[1]. The two CAN communication wires are referred to as CAN High and CAN Low, with CAN High wired to pin 7 and CAN Low wired to pin 2 on the 9-pin connector (an identical socket is used to connect to the simulator hardware). The CAN bus requires 120 ohm termination resistors to prevent network errors caused by signal reflections[2]. Termination resistors were added to the cabling during this research.

5.3 Construction of the communications

The PCAN-USB device's Application Programming Interface (API) allows it to be controlled from different computer languages, including C#. The fuzzer was initially coded to interface to a single adaptor, however, it was soon evident when testing ECUs that multiple interfaces are required. The fuzzer was re-engineered to support multiple CAN connections. An overview of the communications construction is shown in Figure 5.4:

- The PCAN-USB adaptors connect to one or more CAN busses.
- The PCAN-USB adaptors are controlled by the PEAK System driver. The driver is called `PCANBasic.dll`.
- The driver can be configured through software to log data to a file.

[1]https://www.can-cia.org/standardization/specifications/
[2]https://tekeye.uk/automotive/can-bus-cable-wiring

Fig. 5.3 The PEAK-System USB (PC) to CAN interface device used during the development and use of the fuzzer

- The PEAK System interface file `PCANBasic.cs`, a C# class file, is used for full access to PCANBasic.dll (the interface driver).
- A developed C# class, `CANInterface.cs`, connects the UI to the CAN busses. The fuzzer uses one instance of CANInterface.cs for each PCAN-USB adaptor it needs to control.
- CANInterface.cs uses another developed C# class, called `PCAN-USB.cs`, which controls each PCAN-USB adaptor (via PCANBasic.cs).
- A developed timing class, `Tick.cs`, controls the rate of packet transmissions, the default packet transmission is at 1 millisecond intervals, but can be configured to be slower or faster.
- The status of the CAN busses and the packets sent and received are displayed in the fuzzer's UI (see next Section 5.4).

5.4 CAN fuzzer functionality

The functionality of the fuzzer was developed to enable the experimentation in Chapters 6 to 9. As discussed in the methodology in Chapter 3, the use of the fuzzer resulted in additions, changes and improvements.

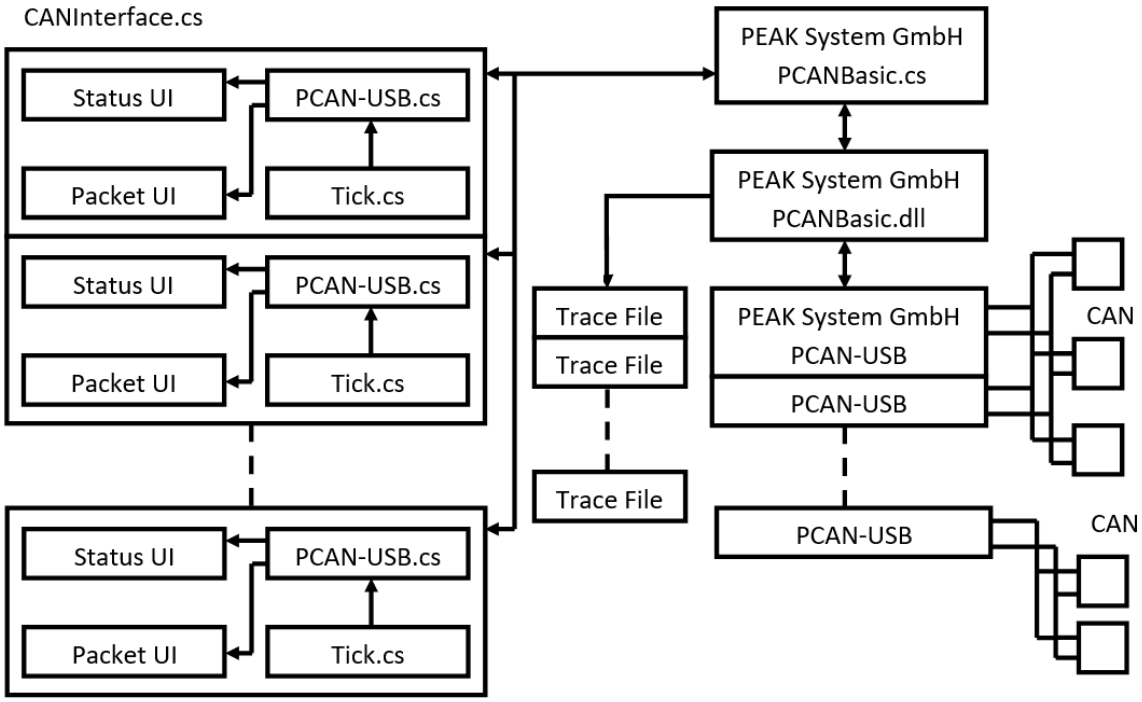


Fig. 5.4 The PEAK-System USB to CAN interface device is provided with an API via the PCANBasic.dll, this is used by the fuzzer code to interface to the CAN bus

The fuzzer is configured through its UI, providing control over the data injected to the ToE (via the CAN bus). Figure 5.5 shows the main CAN fuzzer window, with the various UI screens for different functionality loaded. The main window contains the menu to access all the UIs.

The first step for the fuzz testing is to configure the USB to CAN interface for the ToE. This is done via the UI shown in Figure 5.6a. The fuzzer supports multiple interfaces and they can be given a name (tag) to help with identification. The default bit rate is 500kbps but can be changed via the *Configure* button.

Once the USB to CAN interface is configured, the parameters for the CAN fuzz testing can be set, see Figure 5.6b. The range of random values generated for all parts of a CAN packet can be defined:

- id;
- payload length;
- payload values;

The range can be configured to vary by only a single bit in a single packet, to all the bits in every packet. This feature is important due to the combinatorial explosion problem with the CAN data stream. For example, a standard CAN packet with an 11-bit id and a one-byte payload has half a million packet combinations (2^{19}). At a 1ms transmission frequency (the current minimum

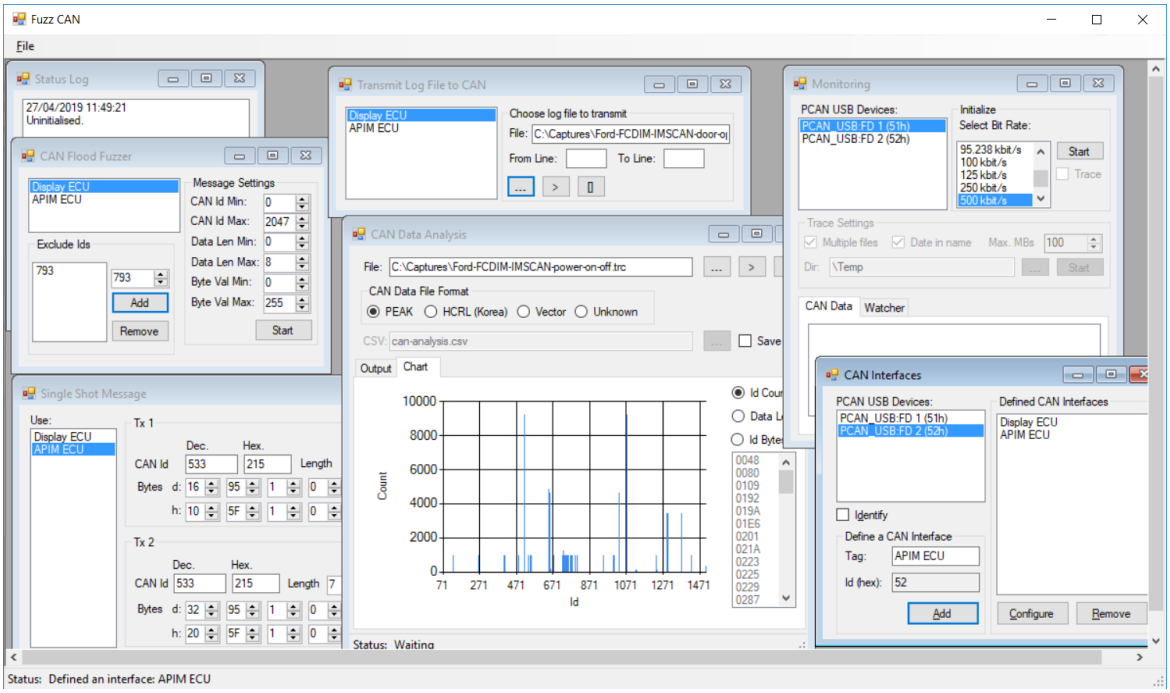
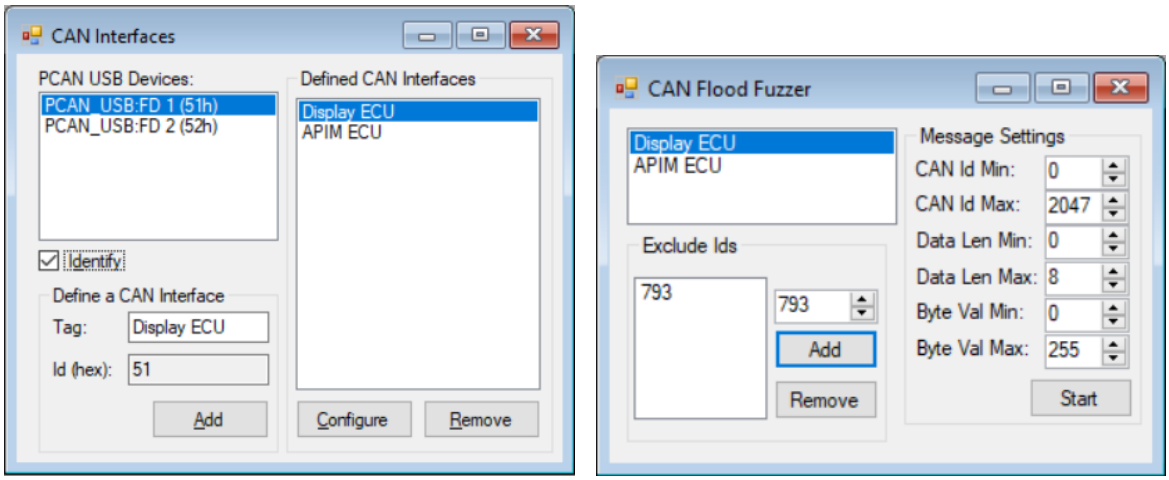


Fig. 5.5 The main CAN fuzzer window houses the other UI screens and the menu to load them.



(a) USB to CAN interface configuration UI

(b) Packet generation configuration UI

Fig. 5.6 CAN fuzz testing configuration UIs

for this fuzzer) it will take over eight minutes to transmit all combinations. Add another data byte and all combinations transmit over a 1.5 days. Beyond that further increases in data length become impractical and the fuzzing may need to be targeted, for example by fuzzing around known packet ids monitored on the CAN bus, or being informed by the design.

The fuzz testing can exclude specific CAN ids, useful for when it is known that certain packet ids must not be generated during the fuzz testing. For example when a previous fuzz testing session found a CAN packet that affected the ToE, or for other known functional ids.

Once configured the *Start* button executes the fuzz testing against the ToE, sending out the random CAN data, as constrained via the configured ranges. If logging is turned on then the CAN bus traffic, including the generated packets, are saved to a *trace* file for later use. The trace file can be processed by the fuzzer for some data analysis, see Figure 5.7a. The analysis includes:

- a count of all CAN packets in the log file;
- a count of packets by CAN id;
- a count of packets by data length;
- the mean packet data lengths;
- the mean value of byte values in each payload position;
- the mean value of all bytes.

The data analysis was used to validate the data generated by the fuzzer. It can be used to look for patterns in the data, and the range of analysis functions will be expanded in further work.

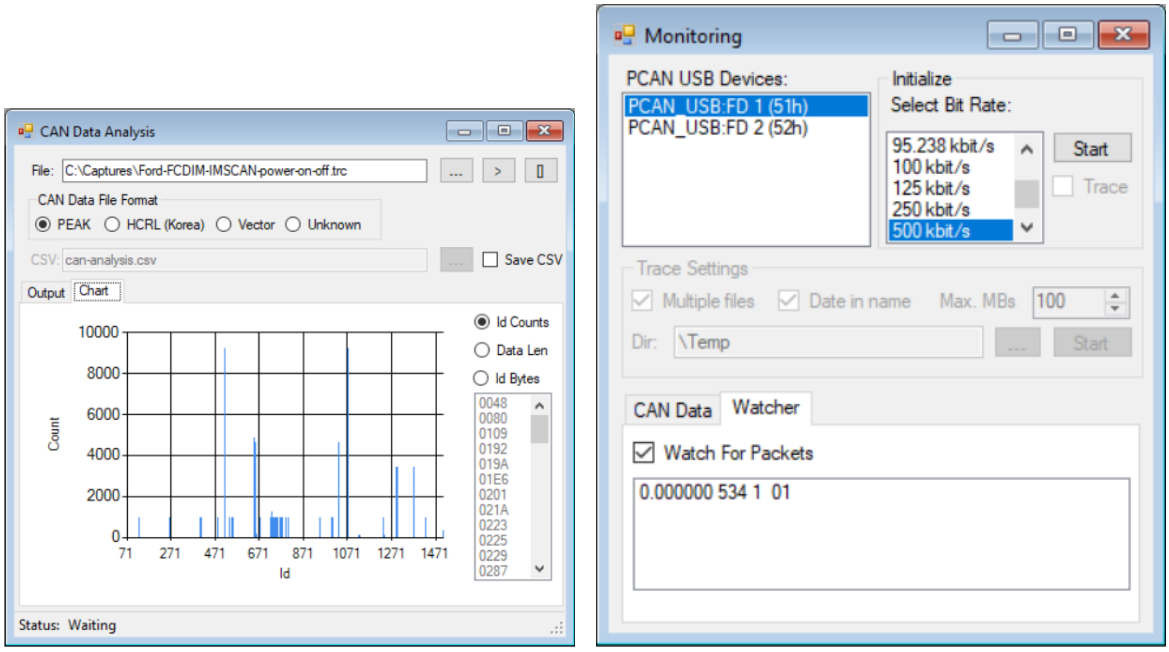
The fuzzer includes functionality to monitor for specific packets on the network, see Figure 5.7b. This can be used when testing for a known event on the CAN bus.

The experiments with the fuzzer revealed the usefulness to be able to transmit individual packets. Figure 5.8a shows the single shot UI. A CAN packet can be configured and sent on a press of the *Tx* button. It also features a repeat function for periodic transmission of a packet.

The fuzzer is able to load a log file and replay the file onto a CAN bus, see Figure 5.8b. The lines from the file to be played back can be configured. This feature is useful when trying to find a CAN packet that causes a system action.

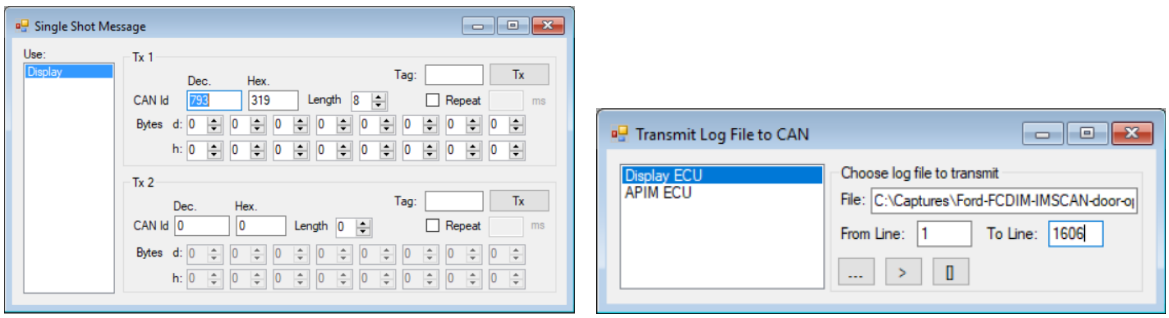
5.5 Using the CAN fuzzer, first validation

The ToEs used during this research program all transmit and receive standard CAN data packets (11-bit ids). The packet parameters (id, data length, payload bytes) that are available to be fuzzed are shown in Table 5.1. From the parameters defined in the fuzzer the random CAN data packets are generated, Table 5.2.



(a) Data analysis UI (b) Packet monitoring configuration UI

Fig. 5.7 Data analysis and packet monitoring UIs



(a) Single shot packet transmission UI (b) Log file transmission UI

Fig. 5.8 Single packet generation and log file transmission UIs

Table 5.1 Fuzzing elements of a CAN data packet

Item	*Range*	*Description*
CAN Id	{0,1,2,...,2047}	All standard packet ids
Payload length	{0,1,2,...,8}	Vary payload length
Payload byte	{0,1,2,...,256}	Vary payload bytes
Rate	> 0	Vary transmission interval

Table 5.2 Sample Random CAN packet output from the fuzzer

Time (ms)	*Id*	*Length*	*Data*
3031.094	000F	6	59 63 BA 5A 77 D5
3032.846	0442	2	AC D3
3035.022	02C4	3	49 01 D8
3036.734	0068	0	
3039.070	0694	5	F5 DA DA 03 A4
3040.854	065A	2	29 95

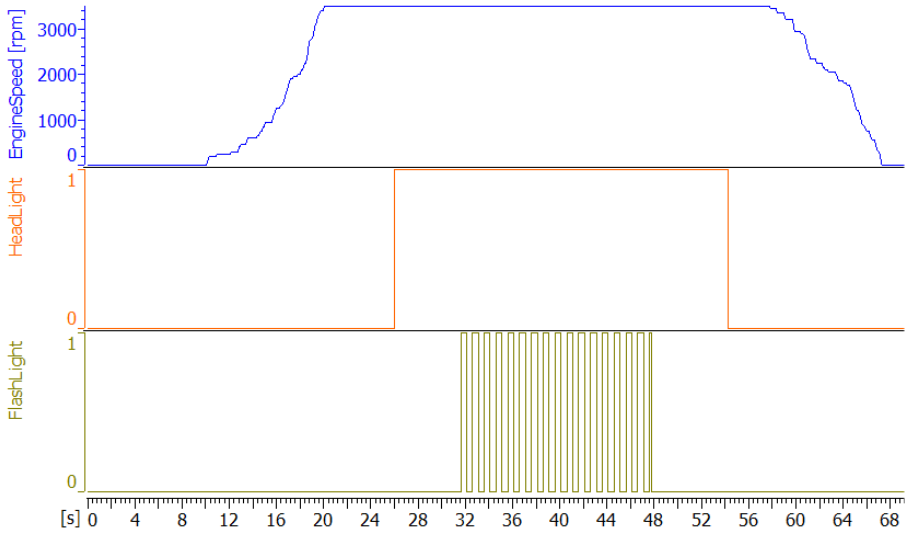


Fig. 5.9 Simulated vehicle signals

The randomly generated CAN packets transmitted from the fuzzer can be measured on the testbed developed in Chapter 4. Simulated vehicle signals are seen in Figure 5.9, whilst Figure 5.10 (shown at a higher rate) illustrates the effect of the randomised data packets on those signals.

The simulator responds erratically when the fuzzer is running and injecting CAN packets. This is caused by the rapid variation in signals induced by the malformed CAN data. In Figure 5.11 the simulated vehicle is displaying a negative engine RPM, showing that the vehicle simulation handles physically invalid values in the same way as physically plausible ones.

The running of the fuzzer against the testbed provides the initial demonstration of the fuzzer tool. Further validation of the tool will come with its deployment and use for subsequent experimentation. However, there was a need to ensure that the random CAN data being produced was valid, this is discussed in the next section.

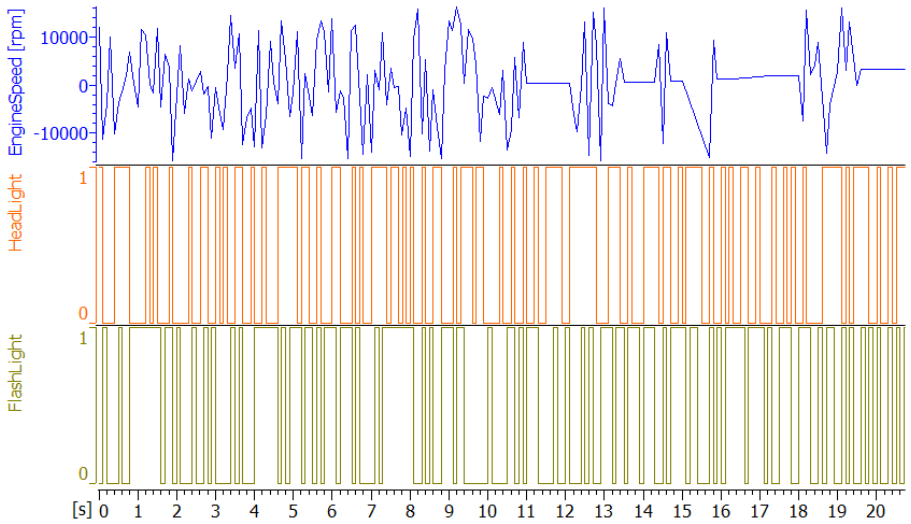


Fig. 5.10 Effect of fuzzing on signals

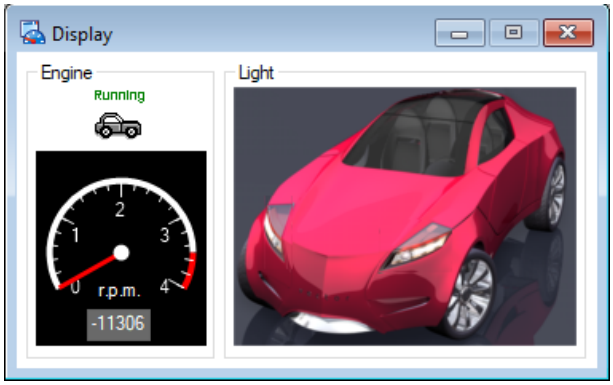
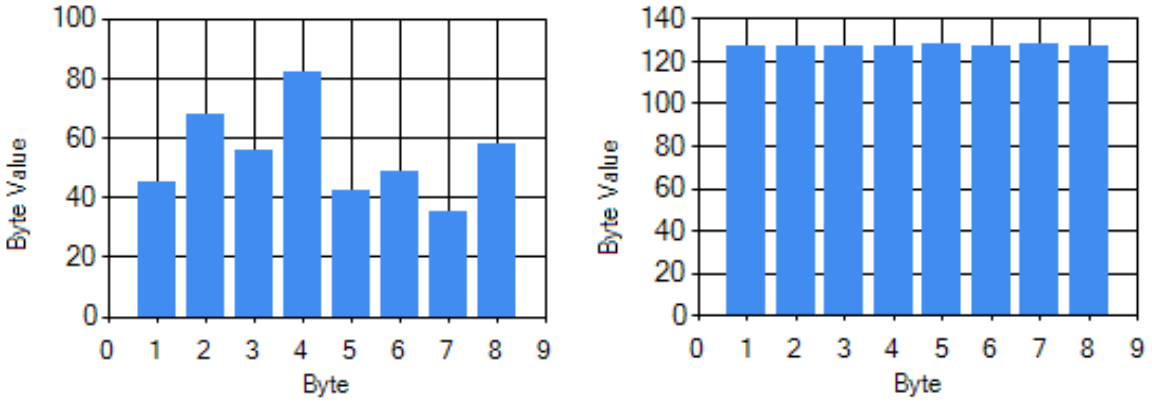


Fig. 5.11 Inappropriate value on a vehicle simulator display via fuzzing



(a) 100000 captured vehicle CAN packets (b) 66144 randomly generated CAN packets

Fig. 5.12 Mean values analysis of the byte values, for each data byte position from the captured vehicle CAN packets

5.6 Fuzzer evaluation

The fuzzer was enhanced to perform analyses on CAN data. This was to done to check the validity of the fuzzer's output, however, it can be used to analyse any CAN data. The fuzzer can be configured to log CAN traffic. Those logs are then run through an algorithm to calculate means on the captured data.

Figure 5.12a shows the mean data byte value for each byte position, calculated from 100,000 CAN packets captured from the target vehicle's network. It shows a non-linear distribution of eight-bit values. In comparison Figure 5.12b shows the same calculation on 66144 CAN packets generated by the fuzzer. The linear distribution, with an overall mean value of 127 for all bytes in all packets, provides evidence that the fuzzer is correctly generating an even spread of byte values.

5.7 CAN fuzzer development summary

This chapter discussed the development of an automotive fuzzer. The chapter's aims were:

- document the rationale behind the need for a specialist automotive fuzzer;
- discuss the software development to produce the fuzzer;
- demonstrate the fuzzer was functional;
- perform an initial evaluation of the fuzzer to verify that it was working as intended.

The motivation for a specialist automotive fuzzer was discussed. An overview of its UI was provided. The fuzzer's use against a testbed demonstrated its operation. The data output of the fuzzer was evaluated to ensure consistency of the generated CAN values. The availability of a simple to use

and configure fuzzer provides a foundation for future enhancements that are required as a result of its application in testing ToEs. The implemented functionality allowed it to be used against physical targets, i.e. vehicles and vehicle components, in Chapters 6 to 9. The practical experiments provided feedback for implementing new features and improving them.

Chapter 6

Automotive fuzz testing

> So in war, the way is to avoid what is strong and to strike at what is weak.
>
> **Sun Tzu, The Art of War**

The literature review, Chapter 2, discussed the few publications that have covered fuzz testing of automotive systems. In reviewing those publications it was apparent that there was a lack of detail in the provided information. The papers or reports concentrated on high-level experiences of deploying commercial fuzzers over reporting detailed results (Section 2.12). This sparsity of detail is not conducive to reproducibility. Therefore, this research provided the opportunity to add to the knowledge on automotive fuzzing testing.

Chapters 4 and 5 addressed stages one to five of the security test development method (see Figure 3.4 in Chapter 3). This chapter addresses stage six, application of the constructed tooling. The prototype CAN fuzzer is deployed against vehicular systems and components. The experimental results are useful to other researchers and engineers developing CAN fuzz testing procedures. Furthermore, the results feed into stage seven of the security test development method, to improve the tooling and CAN fuzz testing techniques.

6.1 Introduction to a fuzz testing experiment

In order to gain knowledge on automotive fuzz testing, it is necessary to perform it. The sections that follow provide coverage on the first full use of the prototype CAN fuzzer. The aim here was to assess the practicality of CAN fuzzing for automotive security testing. The results begin to address how automotive fuzz testing can contribute to vehicle system assurance. Assurance is achieved through engineering countermeasures to discovered weaknesses (see the security concepts diagram in Figure 2.2 in Chapter 2). However, security testing methods to find weaknesses and test developed countermeasures are required. Therefore, the objective in this chapter is to experiment with CAN fuzz testing and in doing so:

1. evaluate the built prototype CAN fuzzer tool for CAN fuzz testing use;
2. understand and document the processes required for CAN fuzz testing to aid the derivation of CAN fuzz testing methods;
3. build the fuzz testing knowledge within the automotive engineering field.

6.2 Method used in applying the prototype fuzzer

This experiment assessed the fuzzer's applicability as a tool for use in vehicle cyber-security testing and to inform further development of the fuzzer. In executing this experiment the tools used were:

- The prototype CAN fuzzer, developed in Chapter 5.
- A Microsoft Windows-based laptop computer as the PC to run the fuzzer.
- A PEAK-System Technik GmbH PCAN-USB device, see Figure 5.3.
- The PCAN-USB device drivers.
- A vehicle instrument cluster is driven from an Arduino Single Board Computer (SBC) provided by HORIBA MIRA. This was the first ToE used in the experiment, see Figure 6.1.
- The lab car was the second ToE used in the experiment.
- Three Arduino SBCs interconnected via a CAN bus. This was used as a ToE to replace the lab car during the development of the experiment, see Section 6.3 below.
- Customised cabling to connect the PCAN-USB devices to the ToEs.
- Custom software developed to act as a proxy for a phone app.

The fuzzer was run and used against the ToEs listed above. The general method of using the fuzzer against a ToE is as follows (it assumes that the PC has the PCAN-USB drivers and the prototype fuzzer installed):

1. The PCAN-USB device is connected between the PC and a ToE.
2. The prototype fuzzer is run on the PC.
3. The CAN data packets to be transmitted by the fuzzer during the fuzz testing are configured.
4. Optionally, configure the data logging file parameters.
5. The ToE is started.
6. Fuzz testing is started.

Fig. 6.1 *Crashing* a vehicle component as a result of fuzzing, the word *CrASH* appeared in the middle of the display during the fuzz testing and would not clear on a power cycle, unlike the other warning lights

7. Observations of the ToE are made during the fuzz testing.
8. Fuzz testing is stopped.
9. Fuzz testing results are evaluated.

There is additional detail on the execution of this fuzz testing method against the specific ToEs in the sections that follow.

6.3 Development of the experiment

The development of the CAN fuzzer in Chapter 5 resulted in an easy to deploy and use program for running on a Windows PC (see Figure 5.1). The testbed used for automotive security testing, see Chapter 4, aided in the development of the fuzzer. The CAN system on the testbed's simulated vehicle provided an environment and target against which the in-development software could be used. This allowed for the fuzzer functionality to be refined and assessed, before deploying against physical ToEs.

For manufacturers, the ToEs will be the systems in their own vehicles. For the experiment developed here, the intended ToE was the lab vehicle. The internal systems of the vehicle are shown in Figure 2.4 in Chapter 2. Previous car hacking research has shown that permanent damage to vehicles is possible, for example, rendering an ECU non-functional (bricking) [22]. Therefore, before testing against the target vehicle the effect of the prototype fuzzer was first assessed against an instrument panel cluster from the vehicle, see Figure 6.1. This is also the same bench based component as used

in the CAN bit rate attack in Appendix C, see Section C.4. The instrument cluster has a connection to a 500Kbps CAN bus within the vehicle (which is exposed to the standard OBD CAN pins).

Use of the instrument cluster was to allow assessment of the possible effects of using the full functionality of the fuzzer on vehicle systems. The initial use of the fuzzer in Appendix C was restricted to controlling the bit rate of the PCAN-USB device and transmitting a single packet. The packet sent for that experiment was restricted to using a single CAN id with fixed values for the payload data. For these experiments the full range of the fuzzers output would be tested, therefore, the possible effect on the vehicle systems needed consideration. Expanding upon the method in Section 6.2:

1. Set the instrument panel cluster as the ToE (the CAN bus bit rate is 500Kbps, the default packet cycle time is 1 millisecond).
2. Perform the CAN fuzz testing using the method above in Section 6.2.
3. Determine if the fuzz testing had an effect on the instrument cluster.
4. If the fuzz testing operated as expected, then change the ToE to the lab vehicle and repeat the fuzz testing method.

6.4 Demonstration of automotive CAN fuzz testing

The steps described in the previous section were followed. There were reactions from the instrument cluster to the running of the fuzzer. It illuminated MILs, played warning sounds, and the gauge needles behaved erratically. A digital display in the centre of the console began to flash the word **crash**. Cycling the power to the cluster removes any MILs that became illuminated. However, the crash message does not clear when the power is cycled. A method to clear this message has not been found, it may require other signals from the full vehicle, or a specialised reprogramming tool.

6.4.1 Affecting a lab vehicle with CAN fuzz testing

The permanent display of the word crash on the component required a reassessment of using the fuzzer against the target vehicle, a shared resource that would incur repair costs if damaged. In this case, the crash message does not clear when the power is cycled to the instrument display, therefore, it is regarded as component damage. The lab does not have the correct repair tools or knowledge to remove the crash message. This meant that a different approach was required when using the vehicle as a ToE, in order to reduce the possibility of the same component damage.

For the lab vehicle, it was decided to limit the experiment to just determine if the fuzzer had an effect. Instead of configuring the fuzzer to generate random data across the entire CAN state space, only a small range of packets would be fuzzed. A range of packet IDs that had been previously

observed on the vehicles CAN buses in normal operation, for example, packets known to affect the instrument cluster gauge needles.

The fuzzer was used against the target vehicle whilst the engine was idling. The fuzzer's generated CAN packets were sent into the two CAN busses exposed by the vehicle's OBD port. The connection from the fuzzer to the OBD port is via the PCAN-USB device using a custom made cable.

The vehicle exhibited similar behaviour to the cluster testing, namely the illumination of various MIL lights, warning sounds from the instrument area, and fluctuating gauge readings. In addition, the vehicle displayed error messages on a central display (the display is tested in Chapter 9) and erratic engine idling speeds. Once it was observed that fuzzing had a significant effect it was halted to prevent possible damage (as in Figure 6.1).

6.4.2 Fuzz testing a bench based CAN bus

Having made the decision to limit the amount of fuzz testing on the vehicle another ToE was required, to continuing validating the CAN fuzzer. Further evaluation of fuzzer was performed against a bench-top hardware configuration.

A bench based configuration was implemented to provide a physical CAN bus with hardware controlled by CAN data packets. The hardware provides a representation of an increasingly common feature of connected cars, namely the control of vehicle functionality via an app, Figure 6.2.

A CAN bus target was constructed from Arduino SBCs fitted with CAN-Bus Shields (containing a MCP2515 CAN bus controller and MCP2551 CAN transceiver). Each SBC acts as a single ECU on the network. The packets on the CAN bus were a subset of those transmitted on the target vehicle's CAN bus (based upon packets captured from the car). One of the ECUs acts as a Body Control Module (BCM), with a Light Emitting Diode (LED) representing the lock status of the vehicle (off for locked, on for unlocked), see Figure 6.3.

A diagram of the replicated functionality is shown in Figure 6.4. The external phone app sends an unlock command to a vehicles infotainment ECU (a.k.a. head unit). This is a secure connection (or should be). The infotainment unit transmits the unlock command over the vehicle CAN bus. The fuzzer is acting as a malicious unit connected to the vehicle network (e.g. via the OBD port or a compromised ECU). When the fuzzer runs it has no knowledge of the CAN data packet to activate the locks.

For this experiment, a PC app is acting as the smartphone app, Figure 6.5, sending the lock and unlock command, effectively as a proxy for the infotainment ECU. This causes the LED to turn on (unlocked) and off (locked), indicating the normal system operation when locking and unlocking the door. Running the fuzzer the unlock (or lock) functionality was activated after a few minutes of randomly generated CAN data. To aid with the detection of the unlock state the testbench was augmented to transmit an unlock acknowledgement CAN data packet. For the real vehicle another detection mechanism would have been required, for example, a sensor on the door lock.

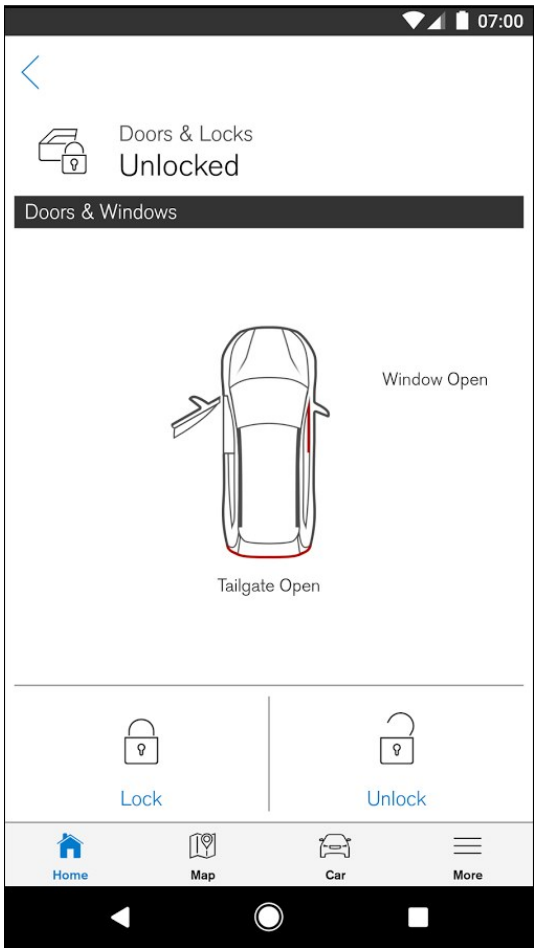


Fig. 6.2 Vehicle control via a manufacturer's smartphone app available from the app stores

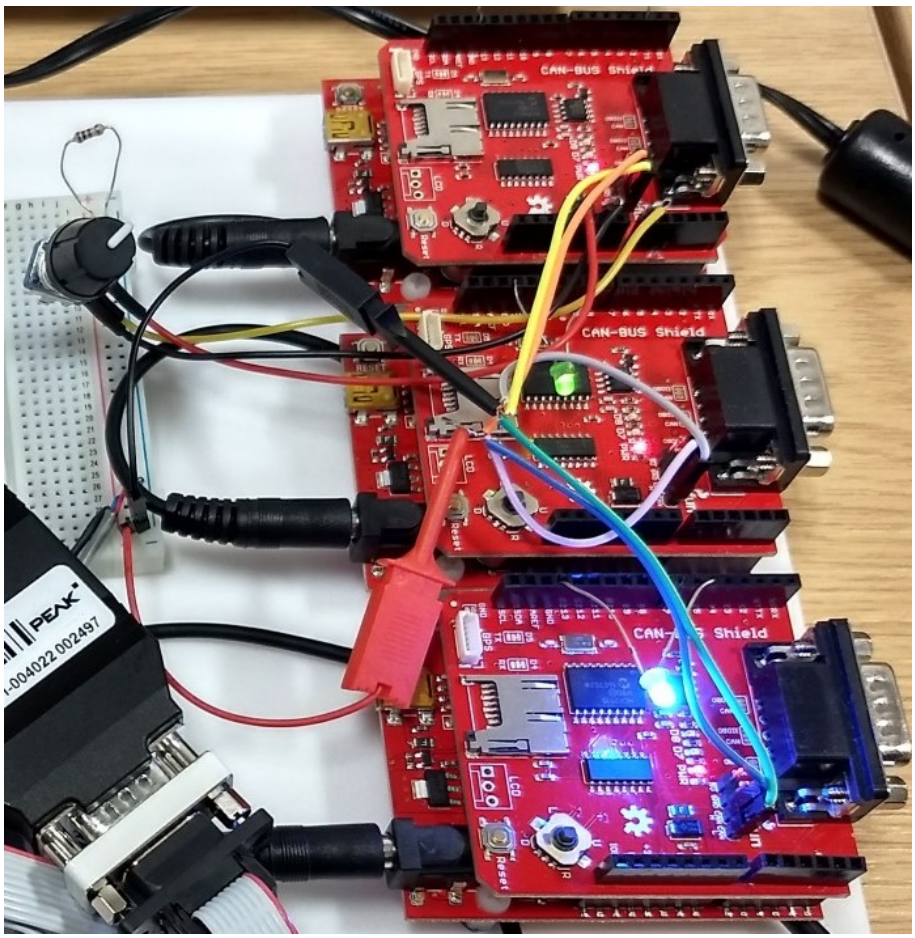


Fig. 6.3 CAN bus connecting three single board computers acting as ECUs, the blue LED indicating the door state, on for unlocked

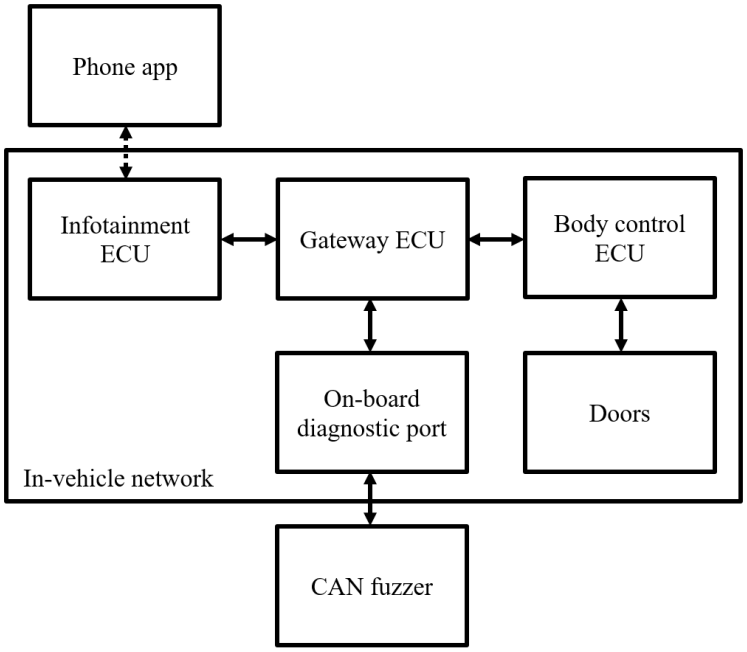


Fig. 6.4 Remote vehicle unlock functionality

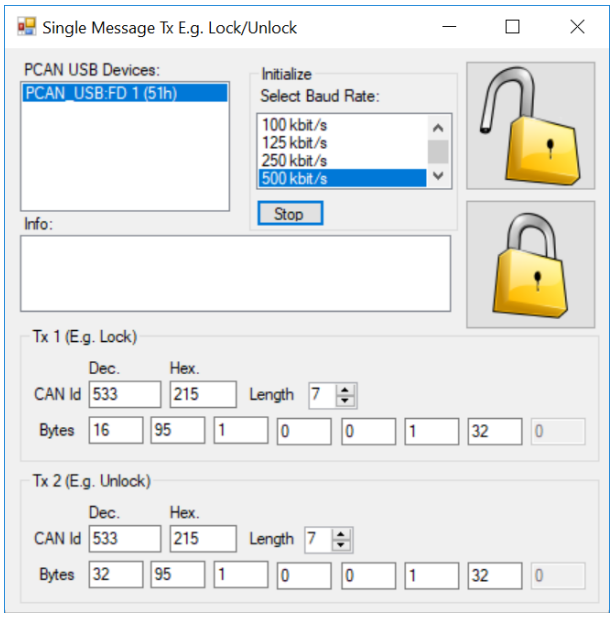


Fig. 6.5 PC vehicle lock/unlock app

Table 6.1 Fuzzer run times to activate unlock

Packet	*Times (s)*	*Mean (s)*
Single id and byte	89, 1650, 373, 400, 223, 143, 773, 292, 21, 559, 572, 80	431
Single id, byte plus data length	3039, 222, 1258, 1330, 314, 277, 959, 3788, 2872, 4472, 3581, 1394	1959

The fuzzer currently has a maximum packet transmission rate of one packet per millisecond. At this rate the mean time to cause the unlock response, based on a small sample of 12 runs, was 431 seconds, Table 6.1. The unlock code was testing for a specific byte value in byte position one in a packet with a specific id. When the code was changed to include a test for the length of the data packet, the mean time increased to 1959 seconds. This simple change in the code greatly increased the time taken for the fuzzer to find the correct unlock packet. If the change had been to check for a two byte value the time increase would have even greater.

The bench based CAN bus is far simpler than the originally intended target of the vehicle. The aim of the bench based CAN bus is to examine the operation of the CAN fuzzer without the possibility of damaging the real vehicle. In the following section, the knowledge learnt from the experimentation is provided.

6.5 Evaluating the fuzz testing

The CAN fuzz testing provided useful information. Prior to these experiments, the known effect of random data injection on to a vehicle's CAN bus had been summarised as having a denial-of-service effect. With the prototype fuzzer connected to a CAN bus the transmitting of random CAN data packets could be performed. In doing so the DoS effect was easily observed against the ToEs. What had not been evident prior to the experiments was how quickly the possible damaging effect of the fuzzer could manifest. The target vehicle's systems and components are not resilient to transmissions of randomly generated data injected onto the CAN bus.

6.5.1 Execution times for fuzz testing

There is one consideration that had not previously been discussed in the few reports related to automotive fuzz testing. That consideration is the potential timescales involved in fuzz testing CAN. In using the fuzzer against the constructed network the time involved was non-trivial.

Initially, the mean time for the fuzzer to find the unlock packet was approximately seven minutes. The unlock command being just a check for a setting of a single byte value in a single packet. Adding a check in the software for a packet length along with the single value quadrupled that mean time to

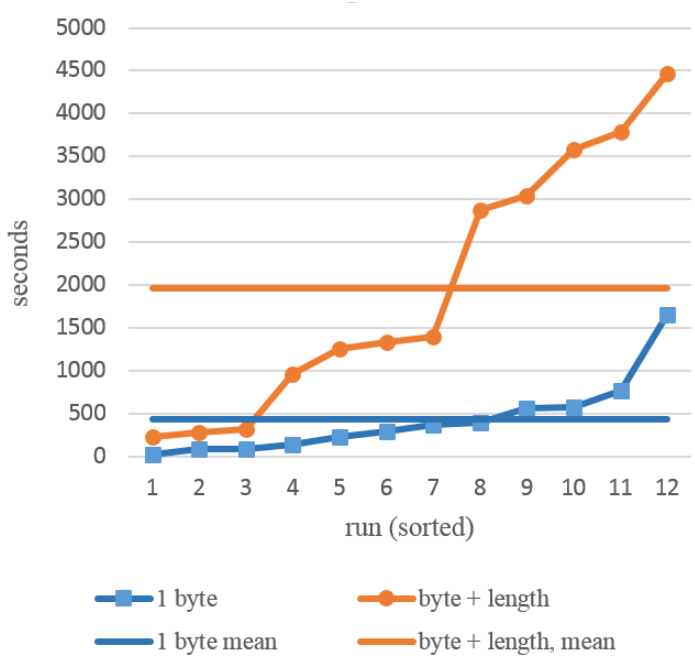


Fig. 6.6 Chart of the (sorted) timings for the fuzzed unlock CAN packet, a simple software changed increased the time to find the unlock CAN packet

approximately 32 minutes, see Figure 6.6. However, if the program had been changed to look for a specific two-byte value that mean time would have been larger, at over one day.

Looking into this further it can be seen that random fuzzing has a combinatorial explosion problem. For standard CAN the packet id ranges in value from 0 to 2047, the payload range from 0 bytes to 8 bytes and each byte holding 256 values. To randomly test all possible CAN values is not sensible. At a 1ms transmission rate for CAN packets the calculated time to transmit all possible packet combinations is provided in Table 6.2. It shows that using only random CAN data is not a practical solution to CAN fuzz testing.

Despite the huge number of packet combinations, the vehicle systems in the lab car were certainly processing the random packets from the prototype fuzzer. In Section 6.4 it was shown that an immediate impact on the vehicle was observed. However, calculations show (see Table 6.2) that the chance of a randomly generated CAN packet triggering a specific ECU function is very low, due to the number of possible combinations of the data payload bytes (8^8). However, the chance that a CAN packet must be read and processed by an ECU is high. This is due to the short 11-bit standard CAN id, providing up to 2048 addressable packets. The time to generate all possible ids is very short, at a 1ms transmission rate all ids can be sent within 2.5 seconds ($2048x0.001 = 2.48$). Therefore, despite the extremely high total number of packet combinations, the CAN fuzzer is often generating a data packet with an id that the vehicle systems will attempt to process, hence, the immediate impact from the fuzz testing on the vehicle systems. The problem for these experiments is a lack of information from the internal design of the vehicle systems, it is black-box testing. Black-box testing is not something that

Table 6.2 Time to run through all possible combinations of standard CAN packets, at 1ms transmission rate with no repeated transmissions, for data payloads of zero to eight bytes

Data Length	*Time*
0	2.048 seconds
1	524.288 seconds
2	1.553 days
3	397.682 days
4	278.731 years
5	713.552 centuries
6	18266.939 millennia
7	4676.336 myr (megayears)
8	3.77789×10^{19} seconds

a manufacturer should have to contend with, and therefore, they will be able to better assess what impact fuzz testing will have on their designs.

6.5.2 Observations from the fuzz testing

As well as learning about execution timing issues with CAN fuzz testing, the experiments did prove useful in confirming observations previously made within the few works available in the literature (see Section 2.12):

1. Security testing vehicles and their components can lead to vehicle component damage. This is demonstrated by the results from fuzz testing the instrument cluster, in Section 6.4, which caused it to develop a fault. In Section 6.4.1 the lab vehicle also exhibited abnormal behaviour during fuzz testing. If testing had continued if may have damaged vehicle components.

2. Fuzz testing can be used to reverse engineer vehicle functionality. In Section C.4 the fuzzer activated the programmed proxy door locking mechanism.

3. Disruption of a vehicle's communication network is not difficult. As for the first point, this was seen on the instrument cluster and within the lab vehicle.

4. Fuzz testing can be used as a form of cyber attack. The denial-of-service effect from the fuzz testing was evident in the behaviour of the lab vehicle (see Section 6.4.1). It would not have been safe to drive the vehicle if random CAN packets were being injected into the vehicle's CAN busses.

Since fuzz testing can have a detrimental effect on a running vehicle it certainly suggests that vehicle systems need additional logic to ignore nonsensical CAN packets, packet data values, and sequences of such values. Thus, it would be beneficial for vehicle manufacturers to add additional countermeasures to protect the CAN busses that operate in a connected car. Such protection of the

CAN bus and vehicle components from external cyber attacks should be a requirement for the design of connected vehicle systems, otherwise, manufacturers risk threats to their products, as discussed in Section 2.6. For such vehicle systems, the aim of this chapter was to show a method that can be used to start incorporating fuzz testing into the security testing regime.

As discussed in the literature review in Chapter 2, security testing is intended to improve system resilience by revealing weaknesses in the security properties (the CIA triad) of a system. Whilst, fuzz testing has achieved that in other domains, this research has begun to do address it in the automotive field. In this chapter the fuzzing test was able to activate security functions without prior knowledge of the system design (confidentiality), it was able to change values displayed on instrumentation (integrity), and disrupt component and vehicle operation (availability). This implies that for connected vehicles designing functionality to correctly isolate security concerns must be a consideration. Indeed the use of protection mechanisms in gateway ECUs in newer vehicles indicates that manufacturers are responding to the issue. Furthermore, simple modifications to a design can improve security. For example, in this chapter adding a check for the length of the CAN data packet to activate the unlock, and, thus, increase the time to find the unlocking feature via random fuzz testing. However, the bench based system is far less complex and further development of the automotive fuzz testing is required against production vehicle systems.

6.6 Conclusion on automotive fuzz testing

The prototype fuzzer developed in Chapter 5 was used against vehicle systems that use the CAN bus. In applying it to a test vehicle it was evident that the vehicle's systems were not resilient enough and there was the possibility of component damage. Another scenario was developed which demonstrated that fuzz testing is a useful technique to be added to the testing tool box. However, it is a technique that has combinatorial limitations. As such its usefulness in the automotive field is likely to be in fuzz testing in a specific CAN packet values space, close to the system's known CAN packets, whether determined from design or CAN bus traffic capture.

It was also apparent that vehicle and component manufacturers need to consider supporting such research. It is not practical for security researchers to have access to a single component or vehicle, ideally, access to several are required. This is due to the cost implications of obtaining, and potentially replacing or repairing, components and vehicles.

In the next chapter, Chapter 7, another vehicle component, an automotive gateway ECU, is chosen as a ToE. This is in order to further assess the capabilities of the prototype CAN fuzzer.

Chapter 7

Investigations into an automotive gateway

I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.

Linus Torvalds

Chapters 4 to 6 have addressed all seven stages of the security test development method (see Figure 3.4 in Chapter 3). Here, the prototype CAN fuzzer is applied to another target and scenario. This is to provide an additional use case and add further knowledge to CAN fuzz testing and determine any tooling improvements. This addresses the improvement mechanism beyond stage seven in the security test development method.

In this chapter a gateway ECU is the ToE, beginning with a brief introduction on gateways and the reasons for choosing a gateway ECU. The methodology used is described in Section 7.2, and a teardown of the gateway hardware is provided in Section 7.3. The initial CAN communications with the gateway are described in Section 7.4. Fuzz testing the gateway ECU did not succeed due to restrictions with the hardware, discussed in Section 7.5. However, the results provided useful knowledge to improve the prototype CAN fuzzer, to understanding the design of the particular gateway ECU targeted, and, thus help with testing other ToEs in this research, see Sections 7.6 and 7.7.

7.1 Introduction to an in-vehicle gateway as a ToE

In general computing, the term *gateway* is common in digital communications systems, though its meaning does vary depending upon the context. Usually, a gateway is used to interconnect dissimilar networks. The networks may have different protocols, data speeds, topologies, physical characteristics or security properties. The gateway is able to read data from one network and pass it onto another network, suitably repackaged.

Gateway functionality within ECUs is not uncommon. Various in-vehicle communications networks exist and are implemented in various protocols and data speeds. A vehicle gateway will interconnect these various networks, see Figure A.3 in Appendix A for an example. Furthermore, the gateway will provide a connection to external networks when a vehicle is serviced, usually via the OBD port. Increasingly, wireless support for vehicle servicing and diagnostics is provided, therefore, gateways may interconnect the external wireless systems to the internal wired systems. In the next sub-section, the reasons for choosing the gateway as a research target for fuzzing are covered.

A gateway was chosen as TOE for fuzz testing for several reasons:

1. It is a key communications component within a vehicle.
2. If a vehicle contains a CAN bus it is likely to be connected to a gateway. This is because vehicle functionality is segmented, for example, the engine and transmission functionality (known as the powertrain) is usually separate from the occupant cabin functionality. However, some data needs to flow across segments, for example, vehicle speed from the powertrain to the in-cabin instruments.
3. It is typically connected to the OBD port and thus easily exposes in-vehicle networks to attackers.
4. Little, if any, work on how a real-world automotive gateway operates is available.

These factors enable any work on a vehicle gateway to provide useful information to researchers. Here, the experiment's aim was to determine if the CAN fuzzer can be used to infer any information about the gateway's operation:

1. Are any packets sent into a gateway CAN connection passed onto another CAN connection?
2. Is it possible to determine other gateway operational characteristics? Examples could include blocking certain packets and the data it contains, or triggering different packets being transmitted in response to packets received.

Answering these questions would have been useful towards testing the security of the gateway ECU, however, as discussed in Sections 7.6 and 7.7 this did not prove possible.

7.2 The experimental method

The reason for security testing ECUs is to try and improve their assurance. In this case, the chosen security test was fuzz testing. The target of this experiment was a gateway ECU accessed via CAN. The initial experimental design was as follows:

1. Decide upon the gateway ECU to use and gather technical information required to use it in the experimentation.

2. Identify the power and CAN connections.
3. Connect the gateway ECU to a power supply with the correct voltage.
4. Configure the CAN fuzzer to send CAN data packets onto the CAN bus.
5. Connect the CAN fuzzer to an ECU CAN input as a CAN data packet sink.
6. Start the fuzzer transmitting randomised CAN data packets.
7. Monitor another CAN bus for a response as a CAN data packet source.
8. After a period of time (determined by the results obtained) stop the CAN fuzzer.
9. Repeat step 7 for each other CAN bus present.
10. Repeat step 5 for each CAN bus present.

The equipment for the experiment is:

- The ECU gateway.
- A laptop PC running the Windows operating systems.
- A PCAN-USB CAN interface to connect the PC to a CAN port on the ECU.
- A PCAN-USB CAN interface to connect to another CAN port on the ECU.
- Power supply to provide DC power to the ECU.

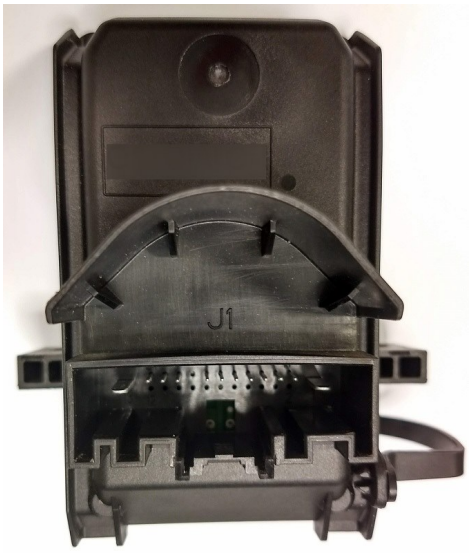
This initial experimental method was not fully achievable due the behaviour of the ECU. This is discussed in Sections 7.6 and 7.7, where the gateway's hardware design and symbiotic operation are noted as restricting the experiment. However, the experiment did provide an insight into the operation of the ECU, and the results provide input into the CAN fuzzer's feature refinement, covered in Section 7.7. The following sections describe the gateway and results from its bench usage.

7.3 Vehicle gateway hardware overview

The gateway used is from a current-generation modern estate car, sold in the global car market. The vehicle details are withheld for reasons of commercial confidentiality. However, the same car is available at MIRA. Access to the full vehicle proved useful. Although not a lab car, in this experiment it was used for simple read-only CAN testing (to ensure that no damage was done).

The gateway ECU component was purchased second-hand, along with wiring diagrams for the vehicle. The wiring diagrams are used to allow for correct interfacing to the gateway, and to apply the correct power connections.

(a) Vehicle gateway showing OBD port



(b) Reverse side of the gateway showing the to-vehicle connection

Fig. 7.1 A vehicle gateway component

With the gateway purchased the investigations into its operation could be performed, the objective was to run the CAN fuzzer on the gateway interfaces. The next stage was to examine the hardware to determine how the gateway is physically connected.

The gateway is shown in Figure 7.1. It does not come with a manual, therefore, the following information was derived from the vehicle's wiring diagram and by examining the device itself.

The gateway is housed in a custom box with lugs for attachment to the vehicle, there is an OBD port on one side, see figure 7.1a. On the reverse, there is another connector that allows for the vehicles wiring to be attached, Figure 7.1b.

The gateway ECU can be opened by separating the two halves of the plastic casing. This reveals a circuit board, Figure 7.2.

Internally the gateway circuit consists of:

- 7 pins going to OBD the connector.
- 12 pins for the ECU connection.
- The power regulation circuit, taking the vehicles normal voltage levels, usually at least 12 volts, and converting it to the power required by the digital circuits, usually 5 volts.
- A MCU, this is a custom Freescale (now NXP) part, SC667275.
- Four CAN ports (with space for one more). The CAN ports use a TJA1042 CAN transceiver.

The electrical connections for the gateway can be determined from the wiring service manual. Seven of the twelve connections to the gateway from the vehicle are exposed to the OBD port, see

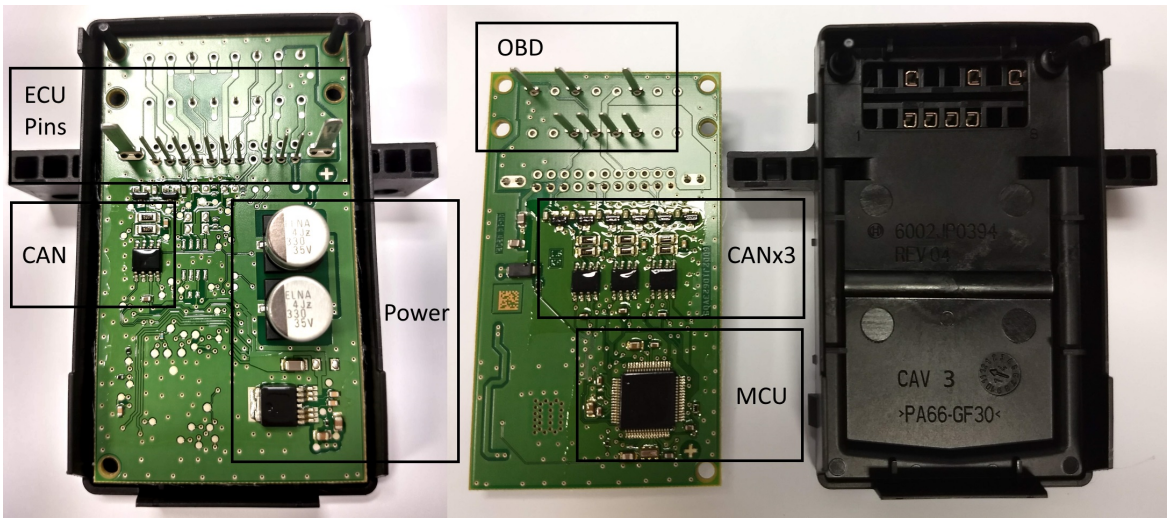


Fig. 7.2 Inside the gateway ECU

Table 7.1. The OBD power pins (16 and 4), supplied from the vehicle ECU pins (1 and 13), are separate from the power going to the voltage regulator (ECU pins 14 and 24) to power the gateway circuit.

The wiring service manual allows for the vehicles internal CAN communications to be mapped out. This is shown in Figure 7.3 (see Table 7.2 for the abbreviations). All four of the vehicles CAN buses are connected to the gateway ECU. Two of those CAN buses are exposed to the ODB port and are thus directly accessible from the vehicle's cabin. See Section 2.7.1 for an introduction to CAN.

On previous vehicles from the same manufacturer, the two OBD exposed CAN busses ran at different transmission rates. On the lab car, the standard OBD CAN port (for reading emissions data) runs at 500kbps, this is known as high speed CAN. The second OBD CAN bus, i.e. the manufacturer's propriety CAN connection, runs at 125kbps. This is known as the medium speed CAN bus.

On the newer model vehicle, both OBD exposed ports run at high speed (500kbps). For the remaining two CAN busses, internal to the vehicle, one runs at high speed, the other at medium speed (125kbps).

7.4 Initial gateway experiment and results

A DC supply was used to power the gateway. The voltage was set to 12.3v, a value measured on the vehicle. The gateway is connected to the PC via the PEAK-System PCAN-USB adaptors, Figure 7.4.

When the gateway was supplied with power there was an initial bust of CAN activity from each of the CAN connections, Table 7.3. Then the gateway stopped transmitting, after 5.6 seconds. The frequency of the observed packets varied. Some of the packets were in the tens to hundreds millisecond range (20ms to 250ms). Some packets are sent once, and several are sent at once per second.

Table 7.1 Gateway ECU connections

Usage	*ECU Pin*	*OBD Pin*
GND	1	4
No pin	2 to 12, 21	1, 2, 7 to 10, 12, 13, 15
VBATT	13	16
SIG GND	14	5
HS3 CAN-	15	n/a
HS3 CAN+	16	n/a
HS2 CAN-	17	11
HS2 CAN+	18	3
HS1 CAN-	19	14
HS2 CAN+	20	6
MS CAN-	22	n/a
MS CAN+	23	n/a
+VE	24	n/a

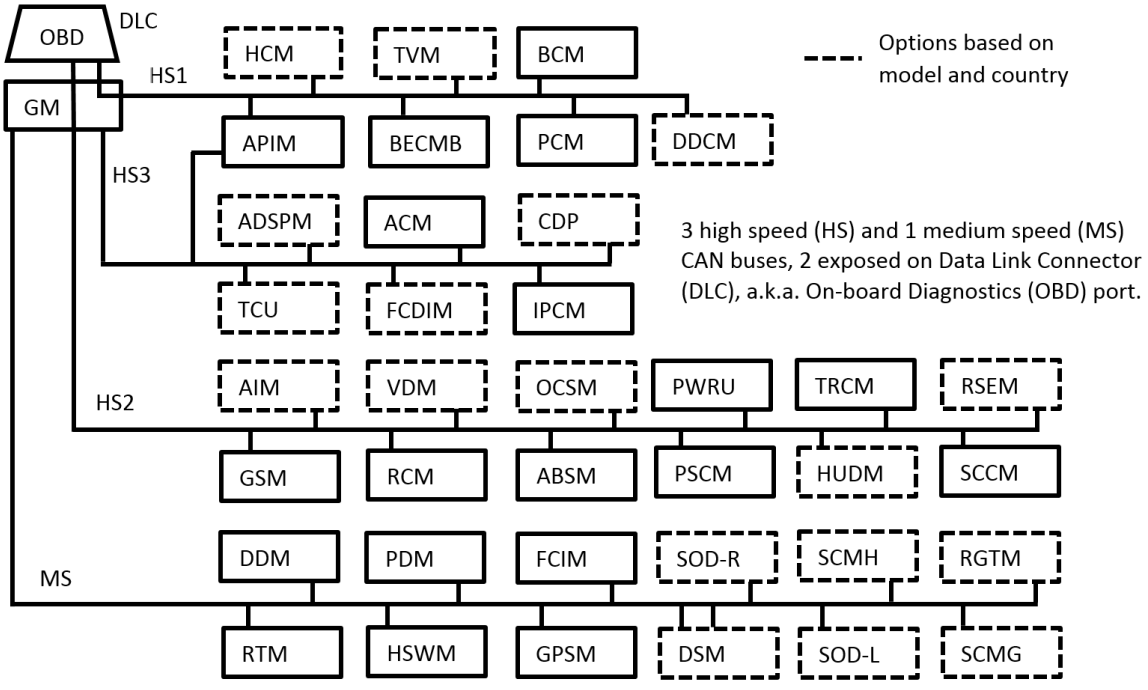


Fig. 7.3 The gateway vehicle's ECU network

Table 7.2 All the vehicle ECUs normally connected to the gateway

Abbreviation	*ECU*
GM	Gateway Module
HCM	Headlamp Control Module (option for adaptive lighting)
APIM	Sync Module
TVM	Torque Vector (AWD) Module (option)
BECMB	Battery Energy Control Module B
BCM	Body Control Module
PCM	Powertrain Control Module
DDCM	DC to DC Converter Control Module (with auto-start-stop system, option for China)
GSM	Gearshift Module
AIM	Auto-dimming Interior Mirror (option)
OCSM	Occupant Classification System Module (not China)
RCM	Restraints Control Module (not China)
VDM	Vehicle Dynamics Module (not China)
ABSM	Anti-lock Brake System (ABS) Module
PWRU	Proximity Warning Radar Unit
PSCM	Power Steering Control Module
TRCM	Transmission Range Control Module
HUDM	Head-Up Display (HUD) Module
SCCM	Steering Column Control Module
TCU	Telematic Control Unit Module (with Telematics)
ADSPM	Audio Digital Signal Processing (DSP) Module (option)
FCDIM	Front Control/Display Interface Module
IPCM	Instrument Panel Cluster (IPC) Module
ACM	Audio Front Control Module
CDP	Compact Disc Player (option)
RSEM	Rear Seat Entertainment Module (option)
RTM	Radio Transceiver Module
DDM	Driver Door Module
PDM	Passenger Door Module
SCMG	Driver Contour Seat Module (option)
SCMH	Passenger Multi-Contour Seat Module (option)
HSWM	Heated Steering Wheel Module
FCIM	Front Controls Interface Module
GPSM	Global Positioning System Module
DSM	Driver Seat Module (with Blind Spot Information System, BLIS)
SOD-R	Side Obstacle Detection Control Module - Right (with BLIS, option)
SOD-L	Side Obstacle Detection Control Module - Left (with BLIS)
RGTM	Rear Gate Trunk Module (option with BLIS)

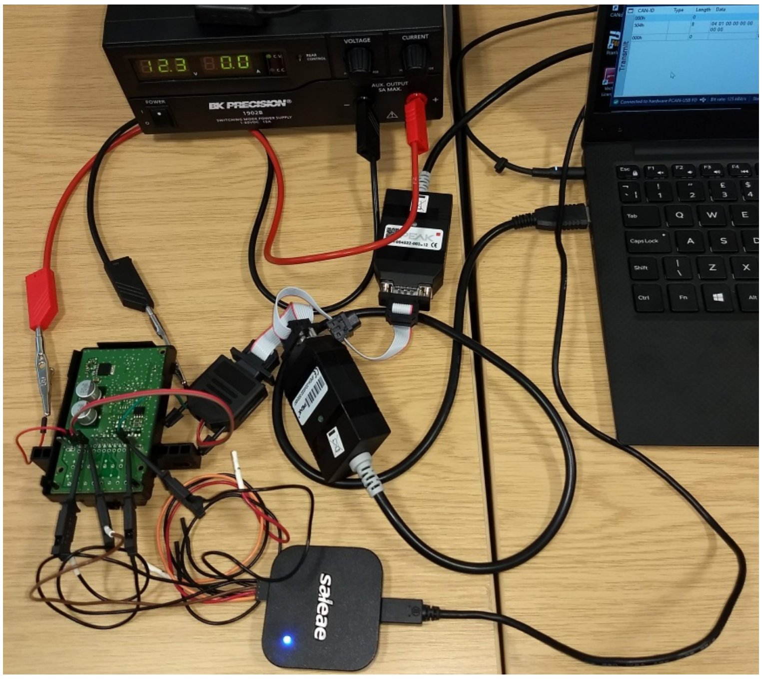


Fig. 7.4 Vehicle gateway connected to the PC

Table 7.3 Gateway CAN transmissions on bench power up

Connection	*#Packets*	*Min Id*	*Max Id*
High Speed 1	10	0x091	0x59E
High Speed 2	6	0x078	0x59E
High Speed 3	33	0x048	0x59E
Medium Speed	10	0x083	0x51E

Table 7.4 Gateway CAN responses to CAN activity, number of packets sent from the connection

	HS1	*HS2*	*HS3*	*MS*
HS1	10	-	-	-
HS2	-	6	-	-
HS3	-	-	33	-
MS	-	-	-	10

Any CAN data packet sent to any of the connections, would cause that connection to send out some packets in response. Those response packets are the same packets observed after initial power is applied. As for the initial power on, the transmissions will stop if no further packets are sent to that gateway's connection. Unlike the initial power on, none of the other CAN connections sent out any packets when one of the other CAN connections saw CAN activity. Thus, apart from initial power-up, the CAN connections remain silent unless CAN activity is present.

This initial examination of the gateway ECU indicated it was at least operating and, therefore, ready for fuzzing experiments.

7.5 Use of the fuzzer with the gateway

The aim of fuzz testing in traditional IT domains is to try and find weaknesses within a system's software. For automotive security research, another use of fuzzing is to reverse engineer component functionality. Using a fuzzer it should be possible to gather information on an ECU's functionality (the range of packets it handles), i.e. reverse engineering. Furthermore, it is worth observing any side-effect behaviours, for the gateway, this could be seeing if different packets get transposed on different CAN interfaces. This operational information can then lead to other possibilities in attacking the component or the vehicles to which it is fitted. The first objective was to connect the fuzzer to the gateway and send in random CAN data packets. This was to check basic operation of the fuzzer with the gateway.

With the gateway ECU powered up, the CAN fuzzer, was configured to send packets into one of the gateway's CAN connections. The first attempt to use the CAN fuzzer against the vehicle gateway revealed an interesting characteristic. The CAN fuzzer was unable to operate with the gateway. Any CAN transmissions to the gateway from the fuzzer saw the PCAN-USB CAN device register errors. An investigation was required to reveal the cause of the issue.

7.5.1 Gateway ECU grounded CAN lines

Investigations revealed that the gateway's CAN connections were pulled to ground, registering a nominal 0 volts. The normal voltage level expected for an idle CAN bus (i.e. in the recessive state) is normally 2.5 volts. When the fuzzer sent a CAN data packet to the gateway the CAN lines returned to that expected voltage level. The gateway's usual power on packets would then transmit.

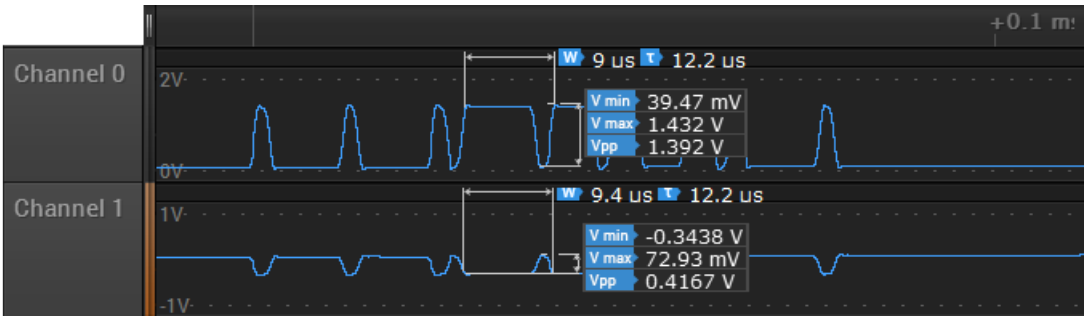


Fig. 7.5 PCAN-USB sends a packet on the gateway's ECU grounded CAN signal, taking CAN high to 1.4v and CAN low to -0.3v, a differential of 1.7v, 0.8v below the 2.5 volt nominal differential required by CAN

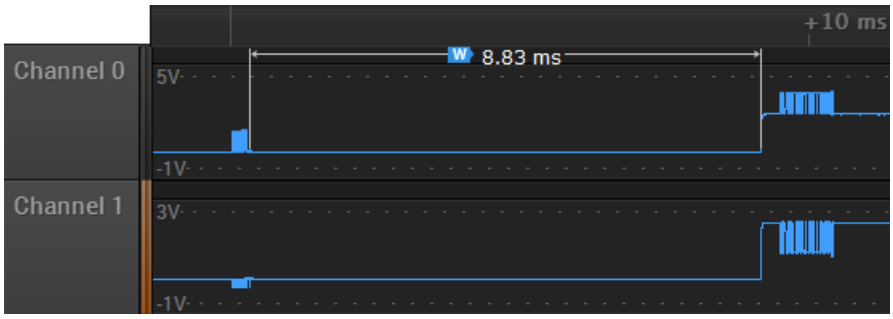


Fig. 7.6 The gateway ECU wake-ups when it sees a CAN data packet, returning the CAN connection from ground (0v) to 2 volts.

Figure 7.5 shows the PCAN-USB interface, used to connect the fuzzer to the gateway, transmitting a short CAN packet (no data bytes) on one of the gateway's CAN connections. The median voltage is around 0V (ground). The PCAN-USB is able to drive the CAN high line to 1.4v and the CAN low line to -0.3v, giving a differential of 1.7v, or 0.8v below the nominal CAN differential of 2.5 volts.

When the fuzzer transmits the packet to a gateway's CAN connection, the gateway responds. After a delay of 9ms, the CAN lines return to the normal CAN recessive voltage of 2.5v. The gateway ECU then begins its data transmissions, as per power on. Figure 7.6 shows this process in action.

However, the packet from the fuzzer to the gateway is not acknowledged, this is the beginning of the errors being indicated. This is as a result of the CAN protocol. A CAN bus needs a minimum of two working nodes for successful packet transmission. One node is the packet transmitting node, and at least one other node to acknowledge the correct transmission of that packets. This does not happen when communicating with the gateway ECU from the fuzzer. What is the reason for the gateway ECU not acknowledging the fuzzer packets?

It was noted in section 7.3 that the gateway uses a TJA1042 CAN transceiver. The following statement was seen in the datasheet[1] for that device:

[1]https://www.nxp.com/docs/en/data-sheet/TJA1042.pdf

> *In Standby mode, the bus lines are biased to ground to minimize the system supply current.*

This indicates that the gateway enters an idle mode and puts the CAN transceivers into a *Standby mode*. When in the Standby mode the data sheet also states that normal data reception is not possible:

> *In Standby mode, the transceiver is not able to transmit or correctly receive data via the bus lines.*

Thus, when the fuzzer sends out the CAN data packet to the gateway, that packet is unable to be acknowledged because the gateway does not correctly receive data. This causes the PCAN-USB interface device to register a CAN acknowledgement error, even with sending the gateway a very simple CAN packet, a CAN id of zero and no data bytes. In Figure 7.7 the transmission error count has reached 136 before the gateway has sent its first packet (signifying it has become operational), 11.5ms later.

However, despite the gateway becoming operational, the PCAN-USB interface continues to indicate errors. The interface then enters a *Bus Off* state. This behaviour suggests that the simple CAN data packet was being ignored by the gateway. How can the fuzzer operate against the gateway device, if the gateway is causing the CAN interface to enter a *Bus OFF* state? This is an issue that needed resolving to continue with the investigations.

7.5.2 In-vehicle gateway operation

As stated in section 7.3, a car fitted with the gateway is available at MIRA. Whilst it is not a lab car available for full experimentation, it can be used to obtain data captures from the car's CAN busses. The gateway on the MIRA car is shown in Figure 7.8a. Two of the CAN busses are available on the OBD port. The other two busses can be connected to via the rear connector of the gateway, Figure 7.8.

There is a difference in the in-vehicle CAN traffic compared to the bench based traffic, illustrated in Table 7.5, where a small section of the captured data from the medium speed bus is present. The capture shows the data from the moment the traffic on the bus began. There are CAN errors seen in the vehicle. The errors are indicated with *ER* in Table 7.5. For example, packets with CAN id *033C* and *03C4* are followed by error indicators. However, they are not acknowledgement slot errors, as seen with the gateway bench testing. That would be indicated with value `19` (hex) in the third byte (see the Peak-System trace format manual[2]).

Observing the vehicle CAN traffic provides information on the data (CAN ids and bytes values) that is travelling on the vehicles networks. This data can be used for the bench testing. It is not possible to know, at this point in the investigations, which of the packets captured on the vehicle, but not previously seen on the gateway, relate to the gateway. Indeed, this gateway has raised some issues that require modifications to the fuzzer to support additional functionality, see the conclusion in Section 7.7.

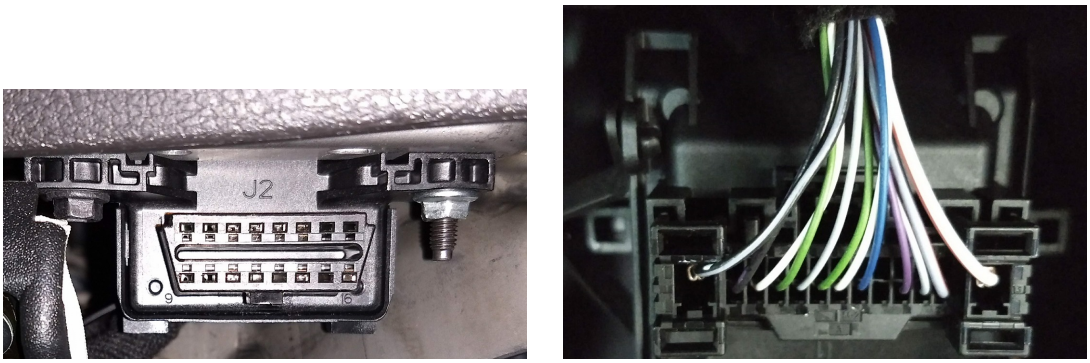
[2]https://www.peak-system.com/produktcd/Pdf/English/PEAK_CAN_TRC_File_Format.pdf

Time	CAN-ID	Rx/Tx	Type	Length	Data
18.2819	000h	Tx	Data	0	
18.2822		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=8
18.2826		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=16
18.2831		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=24
18.2836		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=32
18.2841		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=40
18.2845		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=48
18.2850		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=56
18.2855		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=64
18.2859		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=72
18.2864		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=80
18.2869		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=88
18.2874		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=96
18.2874		Rx	Status	4	BUSWARNING
18.2878		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=104
18.2883		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=112
18.2888		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=120
18.2892		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=128
18.2894		Rx	Status	4	BUSPASSIVE BUSWARNING
18.2898		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=128
18.2903		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=128
18.2909		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=128
18.2914		Rx	Error		Other Error, Tx, Acknowledge slot, RxErr=0, TxErr=128
18.2915		Rx	Error		Bit Error, Tx, Passive error flag, RxErr=0, TxErr=136
18.2934	51Eh	Rx	Data	8	1E 01 00 00 00 00 00 00
18.2939		Rx	Error Counter		RxErr=0, TxErr=207
18.2971	083h	Rx	Data	8	00 00 00 00 00 00 FF FF
18.2981	2FDh	Rx	Data	8	00 00 00 00 00 00 00 00
18.2990	3BCh	Rx	Data	8	00 00 00 96 00 00 00 00
18.3000	3BEh	Rx	Data	8	00 00 00 00 00 00 00 00

Fig. 7.7 The CAN fuzzer signals an acknowledgement error when it sends a CAN data packet to the gateway, and the gateway does not acknowledge the packet (due to being in a standby mode), with the error count increasing by eight on each attempt (as per CAN specification) to the maximum of 128 until the passive error condition is reached

Table 7.5 Car medium speed CAN (left) vs. bench readings (right), data (in hex) differences in bold

Id	*DLC*	*Car Bytes*	*Id*	*DLC*	*Bench Bytes*
051E	8	1E 01 00 00 00 00 00 00	051E	8	1E 01 00 00 00 00 00 00
0083	8	00 00 00 00 00 00 **7E F0**	0083	8	00 00 00 00 00 00 **FF FF**
02FD	8	**4C 00 E1 41 08 10** 00 00	02FD	8	**00 00 00 00 00 00** 00 00
03BC	8	00 00 00 96 00 **01 0C** 00	03BC	8	00 00 00 96 00 **00 00** 00
03BE	8	00 **52 FC C6 65 05 BA 01**	03BE	8	00 **00 00 00 00 00 00 00**
03C4	8	**D7 52 68** 0F 00 00 **92 4E**	03C4	8	**02 00 00** 0F 00 00 **0E 44**
03B4	8	40 11 11 FF 20 20 00 00			
03D5	8	04 00 00 00 00 00 00 00	03D5	8	04 00 00 00 00 00 00 00
042A	8	**C3** 00 00 **80** 00 00 00 00	042A	8	**FF** 00 00 **00** 00 00 00 00
01E6	8	80 0C E2 CE 00 00 00 00			
042C	8	**05 51 54** 00 00 **07 81** 00	042C	8	**00 00 14** 00 00 **00 00** 00
042E	8	**05 51 54** 00 00 **07 81** 00	042E	8	**00 00 00** 00 00 **53 00** 00
0333	8	80 00 00 00 00 00 00 00			
033C	8	24 00 00 00 00 00 00 00			
ER		08 01 1B 01 00			
EC		08 00			
0504	8	04 01 00 00 00 00 00 00			
EC		07 00			
0505	8	05 01 00 00 00 00 00 00			
EC		06 00			
053B	8	3B 01 00 00 00 00 00 00			
EC		05 00			
0332	8	80 FE 90 FB 50 00 00 00			
EC		04 00			
033A	8	00 00 00 00 00 00 00 00			
EC		03 00			
033F	8	02 00 20 00 00 00 00 00			
EC		02 00			
03DE	8	00 01 07 02 80 00 00 00			
EC		01 00			
0084	8	12 00 00 9C 38 2A 0E 00			
EC		00 00			
0346	8	00 00 00 03 03 00 C0 00			
0348	8	00 00 00 00 00 00 00 00			
0359	8	00 3C 00 00 00 00 00 00			
03E0	8	00 00 00 00 80 00 00 00			
03BE	8	00 52 FC C4 00 05 BA 00			
03C4	8	56 52 68 0F 00 00 02 46			
ER		04 01 06 01 00			
EC		00 00			
0083	8	00 00 00 00 00 00 **7E F0**	0083	8	00 00 00 00 00 00 **FF FF**
03BC	8	00 00 00 96 00 **01 08** 00	03BC	8	00 00 00 96 00 **00 00** 00
042A	8	**C3** 00 00 **80** 00 00 00 00	042A	8	**FF** 00 00 **00** 00 00 00 00
042C	8	**05 51 54** 00 00 **06 80** 00	042C	8	**00 00 14** 00 00 **00 00** 00



(a) The vehicle gateway fitted to a car (b) Gateway connected to vehicle

Fig. 7.8 Fitted vehicle gateway

7.6 Gateway testing evaluation

It was apparent, from viewing the gateway functioning in the vehicle, that individually testing an ECU can be problematic. Attempting to operate the gateway on the testbed was not initially possible. Whether the hardware protection of the CAN lines is intentional or not is not known at this time. Even when the gateway was communicating it would not acknowledge packets from the fuzzer. This indicates that some internal filtering is being performed. However, the gateway is constantly communicating when it is part of the vehicle. This suggests that the gateway, as a component, is designed to operate symbiotically, i.e. it needs to be part of the whole vehicle system to operate normally. These observations feed into the functional requirements of the fuzzer. Modifications to test tooling has been anticipated in the automotive security testing development method (see Figure 3.4 in Chapter 3). Continuous improvements to the test tooling is likely to be required to support the many types of ECUs that will require security testing.

The requirement to support multiple CAN busses could be achieved through running several instances of the CAN fuzzer on the same PC. However, this adds complications to test set-ups and cross-network monitoring. Multiple network support for the fuzzer has been added to the communications handling code (see Section 5.3 in Chapter 3). Further modifications will enable support for the cross CAN analyses. This will allow for the possible effect of fuzzing one CAN bus producing a response on another network.

The ability to fuzz packets whilst playing back previously captured traffic from vehicles is a requirement. If components are unable to operate unless part of the whole network, then the ability to simulate a vehicle will be useful. For manufacturers who have full simulations of their systems, it may not be a problem. However, for independent researchers, one method is to capture existing CAN traffic and use the playback as the vehicle simulation. Interleaving the playback with the fuzzed packets would be a requirement, as well as removing the packets to be fuzzed from the playback. These requirements will be fully addressed in future modifications. However, the playback functionality

alone has been implemented (see Figure 5.8b in Chapter 5), as it was useful for other experiments (see Chapter 9).

7.7 Conclusion

The initially proposed experimental steps, see Section 7.2, could not be fully executed. This was due to the hardware construction of the device. Whether that gateway's design is constructed as to intentionally restrict its operation outside of the car is not known. However, this is a recent model vehicle and post-dates early work in the automotive cyber-security field. Therefore, it may have been designed with some protective measures. It is interesting to note how the physical circuit design (ground CAN lines) provides a level of protection against external penetration. Again whether intentional or not from a cyber-security viewpoint is unknown.

What the examination of the gateway has shown is that a fuzzer needs to cater for different scenarios of use. For this component, additional features are required. These features can be added to a list of future modifications for the prototype fuzzer, they include:

- Support for 4 CAN busses.
- A CAN transmit to one CAN bus, whilst logging packets on 3 other CAN busses.
- The fuzzer must be able to detect and reset a CAN bus off state on any CAN interface.

Finally, whilst the original aims of the experiment have not been met, the knowledge extracted on the gateway is useful:

- The physical construction of the gateway, it's wiring to the vehicle's systems, and how it is mounted in the car. This practical experience was useful when using the other ECUs in the research.
- The design of the gateway electronic, and the links to the vehicle's CAN busses. Again, understanding how CAN busses are connected in vehicles is important for this research and future experiments.
- The gateway's use of the low power standby features of the CAN transceivers (grounded lines) and how it prevents testing of the gateway's operation outside of the vehicle. Understanding the nature of the hardware helps overcome similar problems with other ECUs.
- The need for CAN security testing tools to support multiple CAN busses, four in case of the gateway, to fully monitor interfaces when testing vehicle components. The CAN Fuzzer tool was re-engineered to allow for support of multiple CAN interfaces.

The knowledge from this experiment is not known to be recorded elsewhere and, thus, some will find it useful for future research. The experiment can be revisited to build upon what has been learnt.

Chapter 8

Fuzz testing a media interface ECU

Scientists investigate that which already is;
Engineers create that which has never been.

Albert Einstein

In the previous chapter, Chapter 7, a gateway ECU was investigated as a fuzz testing target. The gateway's unique hardware and the current limitations with the prototype fuzzer restricted the useful work that could be done. Another component from the lab vehicle was chosen as a ToE. As for the gateway ECU, this was to further develop the automotive fuzz testing methods and feedback into the design of the CAN fuzzer. Testing more components adds to the overall fuzz testing knowledge, addressing the feedback loop from stage seven to stage three in the automotive security testing development methodology (see Figure 3.4 in Chapter 3).

This chapter details the work in using the fuzzer against the laboratory vehicle ECU called an Accessory Protocol Interface Module (APIM), here referred to as a media ECU.

8.1 Introduction to the media ECU experiment

In order to further develop the CAN fuzz testing methodology, another ToE is used with the fuzzer. The chosen media ECU, the APIM module, is used to provide some of the audio and infotainment functions for the lab car (interfaces to navigation, Bluetooth and radio features, including the optional voice control of functions on some models of the vehicle). The media ECU connections are mainly analogue audio interfaces used for audio signal routing. The media ECU is connected to several other ECUs via two CAN busses, see Figure 2.4 in Chapter 2. The media ECU was chosen for fuzz testing for the following reasons:

1. It is present in the lab car and therefore in-car data readings are possible.

2. An identical ECU can be obtained and used for bench based testing.

3. It has two CAN interfaces, one connects to a CAN bus that is exposed to the OBD port. One connects to an internal vehicle CAN bus. Testing may reveal packets that can bridge from the externally connected CAN bus to the internal CAN bus.

4. It has functionality related to the vehicle's Bluetooth connectivity and navigation functions. Poor security may reveal data stored as a result of those functions. For example map location information and telephone numbers with contact names. However, this was not anticipated as the internal operations of the ECU are unknown.

8.2 Three stage experimental method

For this experiment, the approach is to perform a security test of a vehicle component via its CAN interfaces. The problem being addressed is the security testing of vehicle computational components, in this case, the media ECU is acting as a proxy for a new ECU design that requires security testing. The security test being applied is fuzz testing. The experiment has three high-level stages:

1. The media ECU is examined on the bench to determine the operation of the CAN interfaces. This is used to inform the connectivity to the device for the second stage.

2. The media ECU is monitored to record its communications within the vehicle. The information from this stage is used to inform the configuration of the fuzzer for the third and last stage.

3. The fuzzer is used against the ECU on the bench. The aim is to reduce the number of combinations of packets that the fuzzer needs to transmit, and in turn, reduce the fuzz testing time to a manageable level. This is done by using the information gained from stage 2.

Stage 1, media ECU CAN interface assessment

The equipment used to assess the CAN interface is:

- A bench power supply capable of 12v-13v to replicate the vehicle supply.
- Power leads to connect the power supply to the ECU.
- Two PCAN-USB adaptors to connect the ECU to a computer.
- Two custom cables to connect the PCAN-USB adaptors to the ECU.
- A laptop computer to run the prototype fuzzer and read data from the ECU CAN interfaces.

The method used to determine the operation of the media ECU CAN connections is:

1. Ensure the power supply is set to 12v and turned off.

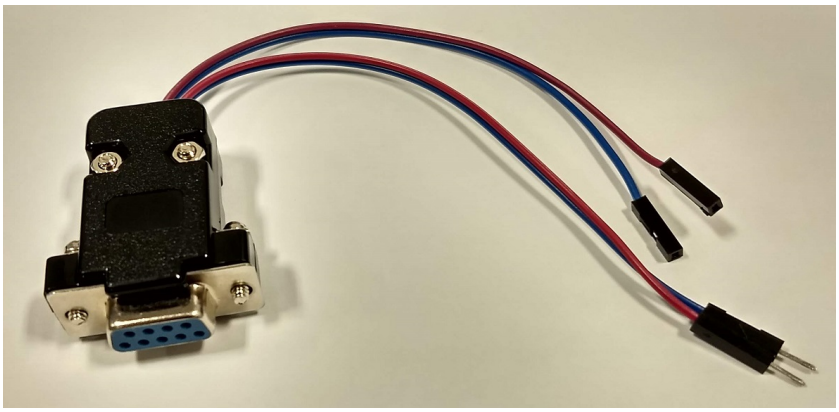


Fig. 8.1 Custom cable for man-in-the-middle CAN monitoring

2. Use the wiring manual for the vehicle to locate the connection points to the media ECU. The connection points are the CAN interfaces and power input.
3. Connect the media ECU to the power supply.
4. Connect the PCAN-USB interfaces to the ECU via custom cabling.
5. Connect the laptop computer to the PCAN-USB interfaces.
6. Start the prototype fuzzer to monitor the ECU CAN interfaces.
7. Turn on the power supply to the ECU.
8. Observe the CAN communications on the ECU CAN interfaces.

Stage 2, media ECU in-vehicle monitoring

The equipment used to monitor the media ECU CAN communications in the vehicle is:

- Two PCAN-USB adaptors to connect a computer to the two vehicle CAN busses connected to the media ECU, Figure 8.1 shows one of the constructed cables.
- Two custom cables to connect the PCAN-USB adaptors to the CAN busses.
- Custom patch cables to allow access to the media ECU connections in situ in the vehicle.
- A laptop computer to run the prototype fuzzer and read data from the ECU CAN interfaces.

The following method is used to monitor the in-vehicle CAN traffic to and from the media ECU. The physical connection method used allows for the ECU to function but the prototype fuzzer to connect in a man-in-the-middle style to monitor the ECU's CAN communications:

1. The ignition for the car is off.

2. Locate the media ECU in the car.
3. Disconnect the media ECU from the vehicle.
4. Using the information learnt in stage 1, splice the two PCAN-USB interface adaptors to CAN busses using the custom cables
5. Using the custom patch cables reconnect the remaining active ECU pins to ensure the normal operation of the ECU.
6. Connect the laptop computer to the PCAN-USB interfaces.
7. Start the prototype fuzzer to monitor the ECU CAN interfaces.
8. Turn on the vehicle ignition to power the vehicle, ECU and CAN busses.
9. Observe the ECU CAN communications on the computer.

Stage 3, Fuzz testing the media ECU

The equipment used for fuzz testing the media ECU is the same equipment used for stage 1 of the experiment. The method used for fuzz testing the media ECU is:

1. Ensure the power supply is set to 12v and turned off.
2. Using the connection information from stage 1 wire the media ECU to the power supply and PCAN-USB interfaces.
3. Connect the laptop computer to the PCAN-USB interfaces.
4. Use the information obtained from stage 2 to configure the prototype fuzzer to generate random packets with CAN ids in the range of observed data packets.
5. Turn on the power to the media ECU.
6. Start the prototype fuzzer to send packets into one CAN bus.
7. Observe the ECU CAN interfaces for packets.

The following sections describe the results of implementing the method stages.

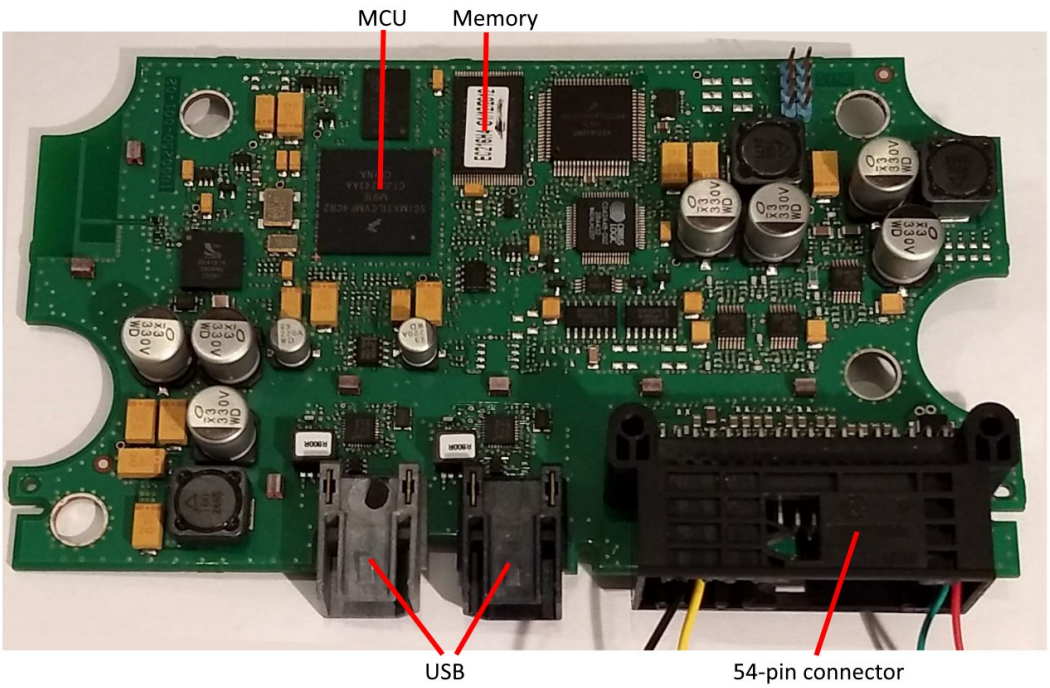


Fig. 8.2 The media ECU circuit board (top view)

8.3 Bench based media ECU CAN interface assessment

The first stage was to determine the information needed to connect and work with the media ECU on the bench and in the vehicle. The method described in Stage 1 of Section 8.2 is used. The electrical/electronic wiring manual for the vehicle is used to determine what physical connections are available on the ECU.

The ECU is connected to the vehicle via a 54 pin plug, see Figure 8.2. 23 of the pins are used, 31 have no connection. Two pins are used to provide power to the ECU, a ground pin and a positive supply pin, usually in the range of 12v to 13v.

There are two CAN bus interfaces to the ECU. One interface is the high speed CAN bus (500kbps) and connects to the same vehicle network that is connected to the OBD port. The other CAN interface bus is the medium speed bus (125kps) and connects to an internal vehicle CAN bus. Table 8.1 provides the connections that were determined, allowing connections from the media ECU to the power supply and PCAN-USB adaptors, via custom cables.

With the media ECU correctly wired on the bench, the power supply could be turned on. The medium speed CAN bus was displaying error messages. Double-checking all the connections did not find any problems. However, a different bit rate configuration from the medium speed bus connection resulted in the correct capture of CAN packets. The configured bit rate for the medium speed bus was changed from 125kbps to 500kps. This means that despite the two CAN interfaces being labelled differently (one high speed and one medium speed) they both required a high speed bit rate configuration of 500kbps.

Table 8.1 Physical connection to the media ECU

Media ECU pin	*Usage*	*Bench connection*
1	+ve (12v)	12v on power supply (+ve)
37	ground (0v)	gnd on power supply (-ve)
16	medium speed CAN high	pin 7 on 1st PCAN-USB adapter
17	medium speed CAN low	pin 2 on 1st PCAN-USB adapter
53	high speed CAN high	pin 7 on 2nd PCAN-USB adapter
54	high speed CAN low	pin 2 on 2nd PCAN-USB adapter

Table 8.2 Bench power on media ECU CAN communications on the high speed interface (Id and Data columns are shown in hex)

Id	*Length*	*Data*	*Frequency*	*Count*
5E2	8	62 00 FF FF FF FF FF FF	1s	6
189	8	00 00 00 00 00 00 00 00	n/a	1
198	8	00 00 00 00 00 00 00 00	n/a	1
199	8	00 00 00 00 00 00 00 00	n/a	1
2E0	8	00 00 00 00 00 00 00 00	1s	9
3EB	8	00 00 00 00 00 00 00 00	1s	8
3F0	8	00 00 00 00 00 00 00 00	1s	8

Although the two CAN interfaces on the media ECU are both configured for the same bit rate (500kbps), the existing labelling will be used for the remainder of the chapter. This means that the term medium speed will refer to the one CAN interface connected to the vehicle internal CAN bus. The term high speed will refer to the CAN interface connected to a vehicle CAN bus that is exposed on the OBD port.

On the high speed CAN bus seven packets were seen, these are shown in Table 8.2.

The observed packets on the medium speed interface are shown in Table 8.3. One of the packets is the same on both CAN interfaces, the packets with an id of 1506, `5E2` hex.

Having determined that the media ECU powered on and CAN data packets were observed, the knowledge could then be applied to enable monitoring of the ECU in the vehicle.

Table 8.3 Bench power on media ECU CAN communications on the medium speed interface (Id and Data columns are shown in hex)

Id	*Length*	*Data*	*Frequency*	*Count*
5E2	8	62 00 FF FF FF FF FF FF	1s	6
455	8	00 00 00 00 00 00 00 00	100ms	81

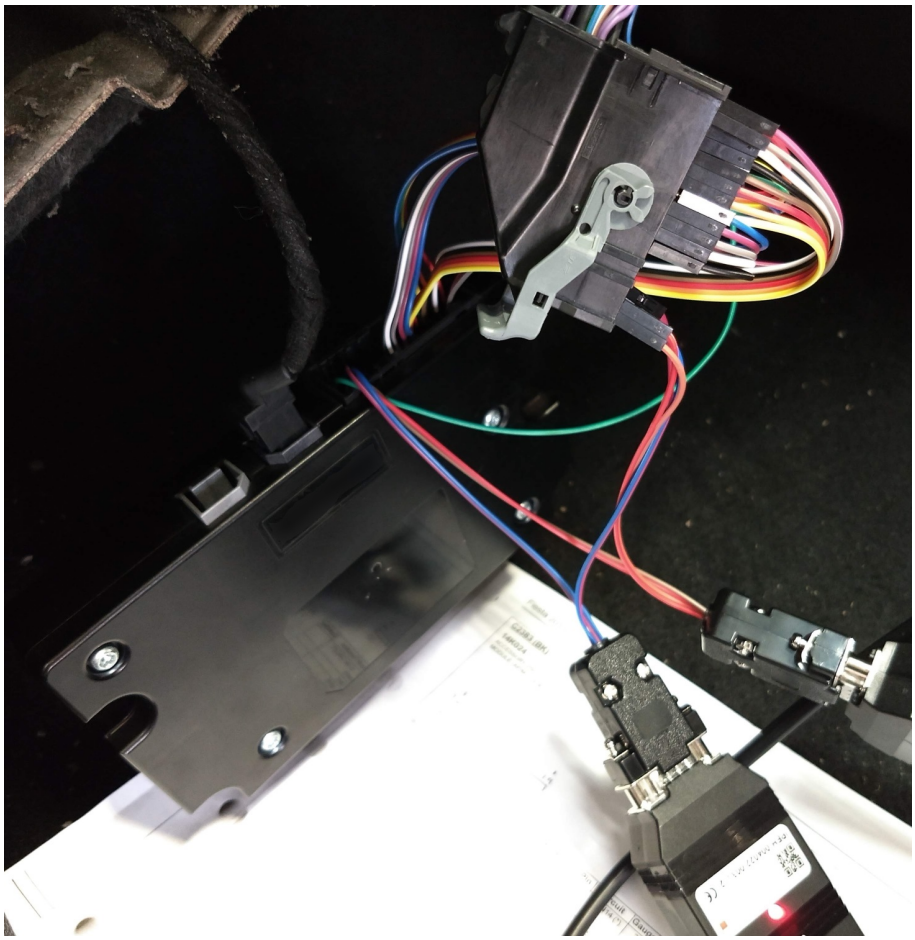


Fig. 8.3 Man-in-the-middle CAN connections to intercept in-vehicle communications to media ECU (used for Bluetooth and audio functions)

8.4 In-vehicle data capture stage

The second stage of the media ECU experiment is to monitor its communications in the vehicle. The method used is described in Stage 2 in Section 8.2. The media ECU is located in a panel underneath the dashboard and behind the glove box. Removing a retaining panel and disconnecting the ECU allows access to the wiring loom that terminates with the ECU's 54 pin connector. A retaining clip on the connector is lifted to allow the connector to be unplugged from the ECU. The custom cables and patch cables are then used to achieve a man-in-the-middle arrangement for the PCAN-USB adaptors, see Figure 8.3. This allows the ECU to operate normally whilst enabling direct monitoring of the ECU's two CAN interfaces.

The two PCAN-USB interfaces were connected to the laptop and configured at 500kbps as discovered during stage 1. With the two CAN interfaces being monitored the car was started. The lower and upper range of the CAN id values observed is shown in Table 8.4.

Table 8.4 The range of CAN packet ids observed when connected to the in-vehicle media ECU, ids shown in hex

CAN bus	*Min id*	*Max id*
High speed	023	5E2
Medium speed	030	5E2

Table 8.5 The range of CAN packets configured for the fuzz testing of the media ECU, ids shown in hex

CAN bus	*Fuzz min id*	*Fuzz max id*
High speed	022	5E3
Medium speed	029	5E3

The monitoring of the media ECU in-vehicle provides a range of values to use as a basis for the bench based fuzz testing.

8.5 Media ECU bench fuzz testing

The third and final stage of the media ECU experiment is to use the information from the previous two stages to perform fuzz testing. Here, the method used is described in Stage 3 of Section 8.2. When the fuzz testing is executing the range of packets to use would be widened by one id step above and below the observed range seen in Table 8.4. That is the CAN id range would be increased by one below the bottom and one above the top seen CAN id values, as shown in Table 8.5.

The media ECU is configured on the bench for the fuzz testing. As for stage 1, it is connected to the power supply and PCAN-USB adaptors. The prototype CAN fuzzer is configured to deliver random CAN packets in the range shown in Table-8.5. The fuzzer is started using a 1ms packet transmission rate (one random CAN data packet is sent every 1ms).

The fuzzer is started against the high speed interface and the output of the medium speed interface is observed. Random CAN packets with ids in the range `022` (hex) to `5E3` (hex) are sent into the ECU's high speed interface. The same packets as when power is first applied are seen on the medium speed. The two packets seen are as shown in Table 8.3 from the initial stage 1 tests.

Next, the fuzzer is started against the ECU's medium speed interface. Random CAN packets with ids in the range `029` (hex) to `5E3` (hex) are sent into the ECU's medium speed interface. The output of the high speed interface is monitored. Packets were observed on the high speed interface. The packets have the same ids as those observed during the initial power on of the ECU. However, three of the packets had different values for some of the data bytes. The packets seen on the high speed interface are shown in Table 8.6. The differences in the data byte values compared to the initial packets, seen in

Table 8.6 Packets observed from the media ECU's high speed interface when fuzz testing the medium speed interface, bold values show changes from the packets observed at power-on (Id and Data columns are shown in hex)

Id	*Length*	*Data*	*Frequency*
5E2	8	62 00 FF FF FF FF FF FF	1s
189	8	**C0** 00 00 00 00 00 00 00	n/a
198	8	**01 D3 01 E5 FE 1D** 00 00	n/a
199	8	00 00 00 00 00 00 00 00	n/a
2E0	8	00 00 00 00 00 00 00 00	1s
3EB	8	00 **0E 70** 00 **20** 00 00 00	1s
3EB	8	00 **0E 70** 00 **20** 00 **40** 00	1s
3F0	8	00 00 00 00 00 00 00 00	1s

Table 8.2 are highlighted in bold. The packet with the id 3EB (hex) appears twice in the table due to the differences observed with the value of the seventh byte, seen as either zero or 40 (hex).

8.6 Evaluating the media ECU fuzz testing

The fuzz testing of the media ECU followed a similar procedure as for gateway in ECU in Chapter 7. Firstly, determining how the ECU hardware can be interfaced to the fuzzer. Secondly, observing its operation in the vehicle, and thirdly assessing the effectiveness of the fuzz testing.

Unlike the gateway ECU, the media ECU is able to operate immediately with the prototype fuzzer. For the gateway ECU its operation was very much tied to the functionality of the vehicle in which it is installed. On the other hand, the media ECU is happy to operate on the bench with the fuzzer sending it packets.

However, what is missing from the bench based testing of the media ECU is the other ECUs against which it is designed to communicate. In this regards, it is similar to the symbiotic operation of the gateway ECU. This means that other CAN packets to invoke the meaningful operation of the media ECU are not present. What impact this has upon the bench based fuzz testing is currently not determined. For example, does operating without the other ECUs restrict the possible interactions that could result from the fuzz testing?

Even though the media ECU is not operating in a fully functional system, the fuzzer was able to evoke a response when randomised CAN packets were input to the medium speed CAN interface. In that case, the output from the high speed interface differed in three of the observed packets. This is summarised in Table 8.7 where three CAN ids have four different data payloads compared to the power on payloads.

Having shown that it is possible for the fuzzer to evoke a reaction from an ECU, how can that be used for security testing? The end goal is to determine a weakness that could lead to a violation of the CIA security properties. Steps towards that goal need to be investigated.

Table 8.7 The data differences for 3 out of 7 of the observed media ECU CAN packets, id 3EB (hex) has 2 different data results

Id (hex)	*Length*	*Power on data*	*Fuzz testing data*
189	8	**00** 00 00 00 00 00 00 00	**C0** 00 00 00 00 00 00 00
198	8	**00 00 00 00 00 00** 00 00	**01 D3 01 E5 FE 1D** 00 00
3EB	8	00 **00 00** 00 **00** 00 00 00	00 **0E 70** 00 **20** 00 00 00
3EB	8	00 **00 00** 00 **00** 00 **00** 00	00 **0E 70** 00 **20** 00 **40** 00

It has not been determined from the testing of the media ECU so far that there has been any impact on its internal operation. This makes it difficult to investigate the integrity and availability properties. In terms of confidentiality, the meaning of the data values in the packet payloads are not known since there is no access to the ECUs design specifications. However, the conditions for the triggering of the data packets could prove useful, if not for this ECU, for other ECUs tested in the future. Is it possible to determine which input packets cause the outputs seen?

It was not a direct objective of this experimentation to reverse engineer the media ECU operation via its CAN packets. However, the results observed may provide the opportunity to find the packets that do trigger the transmission of the other data values in the payloads. This would be an aid to revealing confidential system information, i.e. the security property of confidentiality.

What is required is a method to determine which packet, or packets, from the hundreds of packets sent into the CAN interface every second, evoke the media ECU response. In Table 6.2, in Chapter 6, the impractical fuzz testing of all possible CAN values was illustrated. The experimentation here was to restrict the CAN packets being transmitted from the fuzzer based upon the packets seen in the normal ECU operation in the vehicle. How does that restriction help with determining the packets that cause payload data transmissions?

It is possible to perform some calculations to compare the restricted fuzz testing range with the full range of packets. The medium speed fuzz testing CAN id range was from 029 (hex) to 5E3 (hex). That is a spread of 1466 packets. Although less than the full standard CAN id range of 2048 packets, it still means an inordinate amount of time would be needed to fuzz all possible payload combinations. However, the media ECU was sending out payload packets with data almost as soon as the fuzz testing began. Therefore, the trigger for transmission must be very simple.

At the fuzzers default packet transmission rate of 1ms per packet is does not take long to randomly generate nearly all the possible CAN ids (a few seconds), and less time for the 1466 range used in this experimentation, even though all possible CAN packet combinations with all data bytes takes immensely longer. Thus, it is possible that only a packet id is being used to trigger a response. Having functionality within the prototype fuzzer to help determine if this is the case would be a useful improvement to consider for future new development.

Another useful improvement to the prototype CAN fuzzer would be for it to support multiple ranges of CAN packet ids. The spread of CAN data packets, calculated from monitoring the vehicle

communications, still result in a too high a number of CAN packet combinations, more granular support for ranges of packet ids is required.

8.7 Concluding media ECU fuzz testing

The development of the experimentation on the media ECU was restricted in two ways. Firstly, as for the gateway ECU, the bench based testing does not provide a fully functional environment for the media ECU. Secondly, the prototype fuzzer requires data analysis capabilities to help process the fuzz testing results.

In terms of providing a fully functional testing environment, a vehicle manufacturer and its suppliers have the advantage of access to the designs of the vehicle system in which an ECU sits. That system knowledge can be useful for the testing processes, for example by using the same complex HIL/SIL equipment that is used to design the vehicle systems. In doing so it can aid a bench based security testing process [6]. This was not possible with this experiment, therefore the functionality of the media ECU is restricted. Indeed, only eight CAN ids are observed on the bench, compared to the many more seen when the media ECU is in the vehicle. However, whilst that restricted functionality reduces what can be done on the bench, it also reduces the possible combinations that need to be considered for testing, in this case only eight different packets are observed, of which three can be triggered by the fuzz testing.

Determining the cause of the ECU's packet transmissions resulting from the fuzzer's input packets requires new fuzzer functionality, or another program that can analyse the data as it is generated. That new functionality needs to run dynamically for efficiency. It also needs to change the fuzz testing parameters to help narrow down, and possibly find, the exact packet or packets that trigger the transmissions. These requirements are to be addressed in future development work. The following Chapter on fuzz testing a display ECU covers a similar functional requirement. In the next chapter, the search for CAN packets triggering ECU activity was performed with a semi-automated method.

Chapter 9

Fuzz testing a display ECU

C is quirky, flawed, and an enormous success.

Dennis Ritchie

The testing of various targets in Chapters 6 to 8 revealed knowledge on CAN fuzz testing and provided requirements to the CAN fuzzer that were unknown prior to this research. This chapter builds upon those results by fuzz testing a vehicle dashboard display ECU. Testing another ToE allows for further development of CAN fuzz testing methods and the fuzzer tooling. Further requirements for the CAN fuzzer tool were revealed and weaknesses in the software and security of the lab vehicle's message display system were discovered.

9.1 Introduction to the display ECU

The ToE is a vehicle display ECU called a Front Control Display Interface Module (FCDIM), present in the lab car, see Figure 9.1. This display ECU provides an interface to the vehicle's media system, air conditioning controls, and to display programmed informational messages to the vehicle occupants. Some of the observed messages seen during normal stationary operation of the lab vehicle are shown in Table 9.1.

The display ECU is connected to other vehicle ECUs via CAN. A schematic for the full vehicle network can be seen in Figure 2.4 in Chapter 2. A partial schematic showing the display ECU's immediate ECU neighbours is shown in Figure 9.2. See Table 2.1 in Chapter 2 for the functions of the ECUs.

According to the wiring diagram for the vehicle, the display ECU has two CAN bus connections. One connection is for the vehicle's medium speed (125kbps) CAN bus, MS CAN, which is accessible via the vehicle OBD port. The other network connection is for an internal CAN bus labelled IMS CAN, the meaning of the abbreviation IMS is unknown. What is known, based on the experience

Fig. 9.1 The display ECU in the laboratory vehicle, a variety of messages are displayed in response to occupants operating the vehicle

Table 9.1 Examples of some of the operational messages displayed on the laboratory vehicle's display ECU

Display ECU messages see during garage use
Turn ignition off use Power Button
No phone found Retry Cancel
To start press clutch
Key Battery low Replace Battery
Driver door open
Passenger door open
Climate control on
Climate control off
Climate control A/C on
Climate control A/C off
Climate control Auto mode on
Climate control Auto mode off
Blower
Temperature

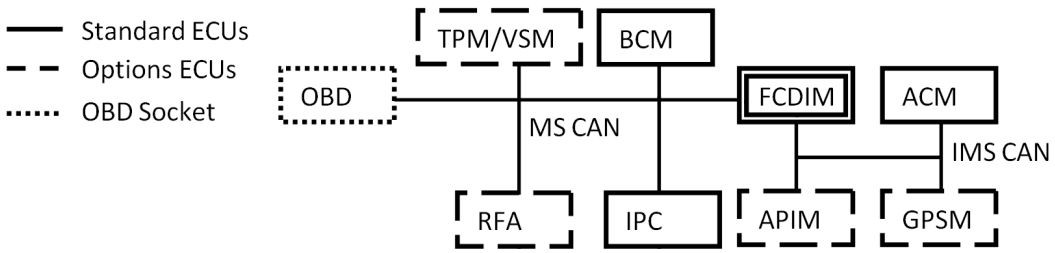


Fig. 9.2 The communications networks connecting the display ECU (the double box) to other neighbouring ECUs, one network is named IMS CAN (abbreviation unknown) running at 500kbps, and one MS CAN, probably for Medium Speed, at 125 Kbps.

with the media interface ECU in the previous chapter (Chapter 8), is that the IMS CAN is known to run at 500Kbps.

Although the display ECU's CAN interface makes it suitable for CAN fuzz testing, it is the graphical display that helps with this research. It has already been noted that a problem with fuzz testing vehicle systems is the cyber-physical aspect. A CAN data packet may not cause a detectable reaction from an ECU via the device's CAN bus, instead the ECU triggers outputs that interface with the real world. Experimenting on an ECU with a built-in visible aspect allows for testing that is not reliant upon the expense and time to reverse engineer and recreate other vehicle sub-systems. Further factors for choosing the display ECU are similar to those for the media ECU examined in Chapter 8. Here are all the factors influencing the choice of the display ECU:

1. it is available on the laboratory car to allow data capture from a working CAN bus;
2. a spare ECU can be obtained for bench based fuzz testing;
3. it has CAN interfaces;
4. it has a cyber-physical (visible) interface.

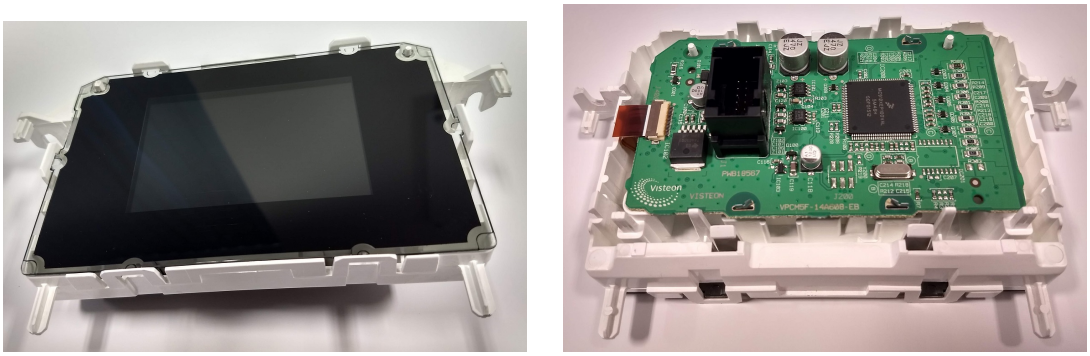
9.2 Experimental method

The approach to the experiment was the same as that taken in Chapter 8. The display ECU was acting as a proxy for a new and untested vehicle component. The security test to be performed is fuzz testing. The experimental stages follow the same stages as those for the experiment on the media ECU, see Section 8.2, substituting *display* for *media*. There is a significant difference with the presence of the screen. Which provided a visual indication of the fuzz testing action. Likewise, the equipment and practical methods used are similar to those used for the display ECU, any differences are discussed in this Chapter where required.

9.3 Display ECU CAN interfacing

In Chapter 6 and Appendix C the possibility of component or vehicle damage from fuzz testing, however small, was discussed. To prevent possible damage to the lab car's display ECU a spare, shown in Figure 9.3, was obtained for the experiments. The display is attached to the vehicle using a 12 pin plug (the black socket for the plug is seen in Figure 9.3b). The function of each pin, obtained from the vehicle's wiring manual, is shown in Table 9.2.

The connections used for fuzz testing are power (pins 1 and 9), connected a bench power supply, and CAN (pins 2 to 5), which are connected to PCAN-USB adaptors, see Figure 9.4.



(a) Display ECU front

(b) Display ECU internal view

Fig. 9.3 Display ECU component, the rear cover has been removed to show the internal circuit board, the MCU used in the display ECU is a Freescale (now NXP) MC912XEP100VAL, this is a 16 bit computational device with various inputs and outputs, and on-board flash and static memory, also visible were transceivers for LIN (a TJA1020) and CAN (a TJA1042/3). A power regulator (7A6050Q1) converts the vehicles 12v power to 5v

Table 9.2 Display ECU connection pins, two are for power, four for CAN busses, three are unused, three are connected to switches (including one labelled for controlling Media Oriented Systems Transport (MOST), which is not present in the lab vehicle)

Pin Number	*Function*
1	+VE (battery supply)
2	MS CAN+
3	MS CAN-
4	IMS CAN+
5	IMS CAN-
6	not used
7	control switch
8	control MOST
9	GND (battery ground)
10	switch
11	not used
12	not used

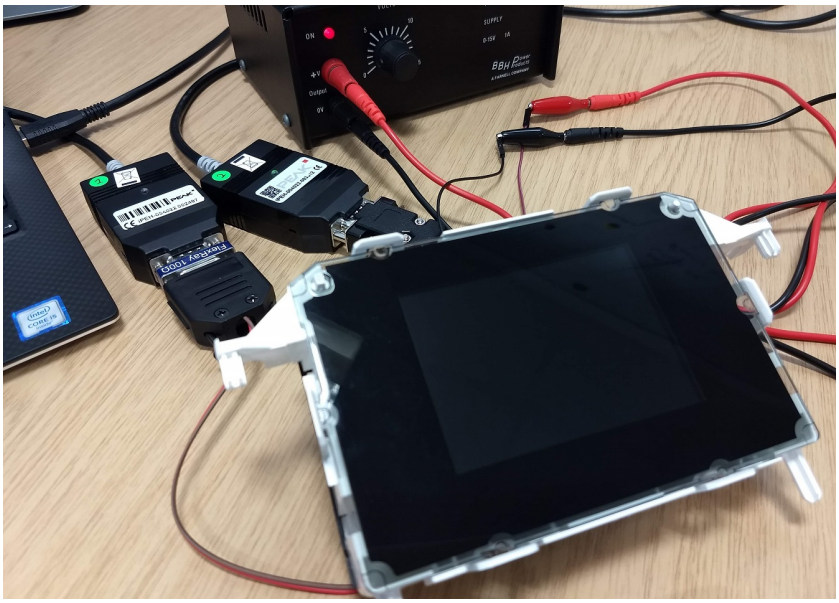


Fig. 9.4 The display ECU is connected to a bench power supply and PCAN-USB adaptors

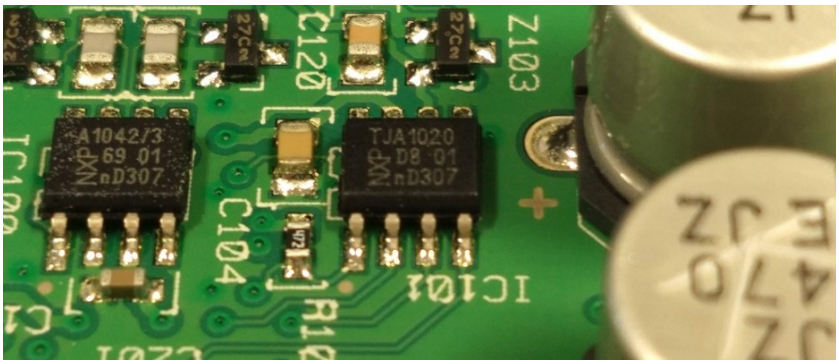


Fig. 9.5 The CAN transceiver chip in the display ECU is a TJA1042/3, shown in the left of the figure, this supports a low power standby mode and wakes up on bus activity

9.4 Debugging the display ECU CAN bus connections

The display ECU has the same standby CAN issue that was seen with the gateway ECU in Chapter 7. The display ECU uses a similar CAN transceiver (a TJA1042[1], see Figure 9.5). The CAN transceiver supports a low power standby mode, i.e. it will *wake up* when CAN traffic is detected. This feature causes an issue with the CAN fuzzer interface which registers an error state. This is due to the first packet from the fuzzer not being acknowledged by another CAN node, therefore, an error is generated (as per the CAN specification). The media ECU in Chapter 8 exhibits the same behaviour.

The solution was to debug the display ECU's CAN connections using two interfaces with the CAN fuzzer. The experimental setup is illustrated in Figure 9.6. This aided monitoring of the CAN

[1]https://www.nxp.com/docs/en/data-sheet/TJA1042.pdf

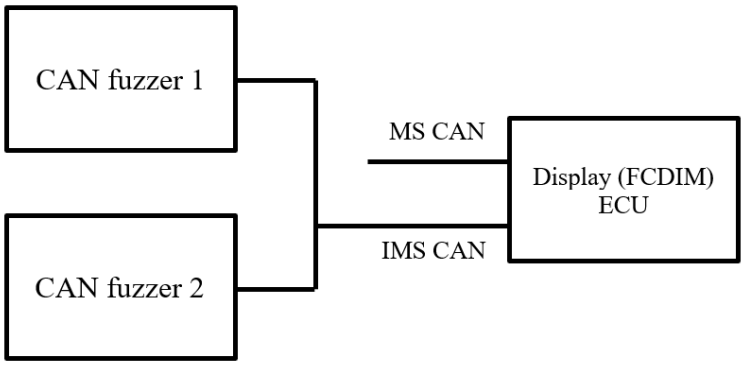


Fig. 9.6 Instead of interfacing to both CAN connections on the display ECU, two CAN fuzzers are used to debug one CAN connection at a time, one fuzzer for sending a data packet and one for monitoring the bus

(a) Display ECU access (b) Display ECU rear connection

Fig. 9.7 Using a trim removal tool the cover of the display ECU is removed, two Torx screws hold the component in place, once removed the ECU plug is accessible

bus on one interface whilst data was being transmitted on the other. It also enabled the first data packet from the CAN fuzzer to be acknowledged and prevent error conditions from occurring.

The MS CAN connection and the IMS CAN connection were sent CAN data packets. There was no observed CAN data or response from the MS CAN connection. To try and determine the issue with the display ECU's MS CAN connection its operation in the lab car was investigated.

9.4.1 Lab vehicle display ECU

The car's display ECU was swapped with the one obtained. Access to the ECU required removal of the plastic trim surrounding it using a trim tool, Figure 9.7.

The spare display ECU is not identical to the one fitted to the vehicle, compare Figures 9.3a and 9.7a, however, when fitted it functions correctly, see Figure 9.8.

Fig. 9.8 The bench ToE display ECU in the laboratory vehicle, the obtained display ECU is a lower specification component than the one fitted to the lab car, however, it functions correctly (the correct vehicle model and year was used to order the bench ToE, but after removing the vehicle's display ECU it was seen that the twelve character part numbers differed in the last two characters)

9.4.2 Monitoring the display ECU connections

In Figure 9.9 cabling is attached to the display ECU for monitoring. The same custom man-in-the-middle cables were used (see Figure 8.1 in Chapter 8) as for the media ECU. Jumper wires ensure the car's power reaches the ECU.

The data traffic for the IMS CAN connection was easily captured. However, no CAN traffic was visible on the MS CAN connection. A continuity tester was used to check the connections from the MS CAN pins on the OBD port to the MS CAN pins on the display ECU plug. No connection was found. This means that the display ECU's MS CAN connection, shown in the vehicle's wiring diagram, is not present. This was checked by removing the man-in-the-middle connection to the MS CAN pin, it did not affect the operation of the display. It still displayed messages when actions on the vehicle were performed (e.g. opening doors). Therefore, for this model of vehicle, the display ECU's wiring diagram did not correspond to the physical connections for the MS CAN bus in the car. This means packets related to functions not directly controlled by the display ECU, e.g. a door open message data packet, are being routed through the media ECU, see the network schematic Figure 2.4 in Chapter 2. This test discovered that only the IMS CAN interface is used on the display ECU, thus, only one CAN interface required fuzz testing.

9.5 Display ECU CAN packets

When the display ECU is functioning in the vehicle up to 75 different CAN ids were observed, this compares to the 12 CAN ids seen when monitoring the display ECU on the bench, see Table 9.3. All the CAN packets seen on the IMS CAN have eight bytes of data. The display ECU is only responsible for one-sixth to one-quarter of the IMS CAN traffic.

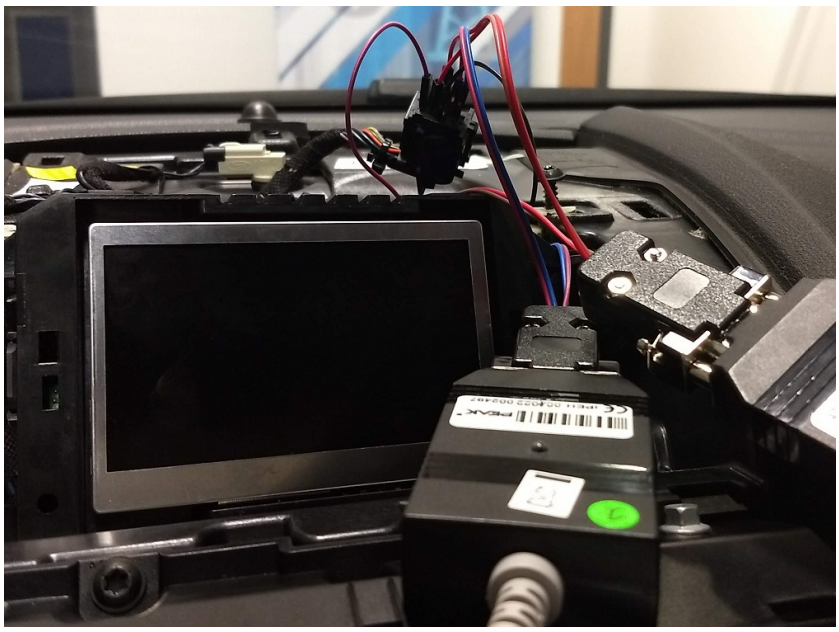


Fig. 9.9 Man-in-the-middle monitoring of the display ECU CAN busses

Table 9.3 The number of CAN packets seen at the display ECU's IMS CAN connection within the vehicle

Number of CAN ids	*Vehicle Operation*
60	Vehicle ignition on, then off
72	Vehicle, ignition on, driver door open and then closed
75	Vehicle start and stop and wait for bus to go idle
52	Vehicle idle, waiting for CAN bus to go silent
12	Display ECU bench based CAN interfacing

Table 9.4 The display ECU's CAN packets, and packet cycle times, in the order observed on initial bench testing

CAN id	*Id (hex)*	*Cycle time (ms)*
1371	55B	1000
402	192	1000
417	1A1	24
477	1DD	24
553	229	1000
647	287	200
656	290	200
675	2A3	1000
739	2E3	1000
801	321	1000
802	322	1000
1235	4D3	1000

During the bench testing, twelve CAN packets (see Table 9.4) are sent from the display ECU when a single CAN packet is transmitted to it. In the absence of any further CAN packets being sent to the ECU, it will stop transmitting after 5.5 seconds. The screen of the ECU is not turned on when the ECU is powered up, even though it does communicate when first switched on. It was discovered, when the fuzz testing was performed, that the screen is only turned on when it is told to display a message.

The isolated behaviour of the display ECU, compared with its in-vehicle environment, again, raises the issue of the effectiveness of testing a single ECU component outside of the complete vehicle system. A point discussed further in the final Chapter 10.

9.6 CAN Fuzz testing of the display ECU

The fuzzer was connected to the ECU and configured to generate random CAN packets. The data length was fixed at eight bytes, reflecting the number of bytes seen during the vehicle monitoring. The full standard CAN id range was set, 0 to 2047. Thus, the CAN id and byte values were being varied. The configuration of the fuzzer is shown in Table 9.5. Later testing would examine variations in payload length, see Section 9.12. The fuzzer's packet generation rate was set at 1ms.

The fuzzer was started and after a short period (10s of seconds) the display flashed the message *Park brake applied*, see Figure 9.10. This physical response to the CAN fuzz testing demonstrated that the fuzzer generated a message that the display ECU is programmed to show. This is similar to the initial fuzz testing in Chapter 6, where the fuzzer found the unlock CAN packet. Having the fuzzer generate a response from the display meant that further testing with observable results was possible.

Table 9.5 Fuzzing elements of a CAN data packet targeting the display ECU

Item	*Range*	*Description*
CAN Id	{0,1,2,...,2047}	All standard packet ids
Payload length	8	Number of data bytes
Payload byte	{0,1,2,...,256}	Vary payload bytes
Rate	1ms	Packet transmission interval

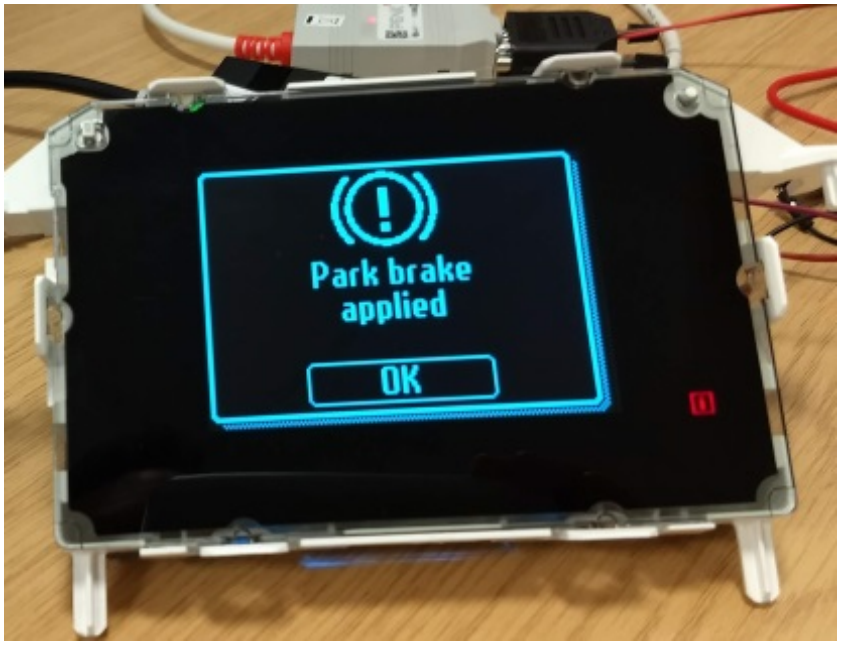


Fig. 9.10 Displayed message during fuzz testing

9.7 Using the CAN fuzzer to find ECU functionality

The results from fuzz testing the display ECU allowed for reverse engineering. The functionality of vehicle systems and components can be difficult to obtain, mainly due to commercial confidentiality. However, reverse engineering vehicle systems is useful for several reasons:

- for operational knowledge, by commercial competitors (vehicle manufacturers and component suppliers) and independent repair companies;
- functional safety engineers, to understand the operation of vehicle systems;
- security engineers, to use system operational knowledge to aid penetration and vulnerability testing;
- adversarial agents who have an interest in attacking vehicle systems.

The log files from the fuzz testing are available, and knowing that a response from the display ECU was seen, then the CAN data that caused the response is determinable. There is a constraint because transmitted CAN packets may not invoke an immediate reaction from the ECU, due to processing delays. The short time delays also mean that correlation between the transmitted CAN packet and the ECU's reaction is difficult to determine from observation alone. However, by playing back the logged CAN data packets, and systematically reducing the number of packets being played back, it is possible, by a process of elimination, to determine a packet that invokes an ECU reaction.

9.8 Log file search for a CAN packet

Using the CAN fuzzer's log file playback ability, the log file from the fuzz testing was divided in half and transmitted to the display ECU. This was repeated if the observed message was seen, otherwise, the other half of the sub-divide log file was played back. This was a search for the CAN packet that caused the display ECU to react. (For the detailed method on the search see Appendix F).

When the log file for the *Park brake applied* message was played back another message was also seen, *Auto StartStop Switch ignition off*, see Figure 9.11. This new message was not observed on the initial fuzz testing run. This required the search to be performed twice (to find the CAN packets causing the display of both seen messages).

9.8.1 Isolating message generating CAN packets

The results for the searches for the two seen messages, *Park brake applied* and *Auto StartStop Switch ignition off* are shown in Tables 9.6 and 9.7. In those tables the line numbers (from the log files) of the packets played back are listed, with the number of messages seen during the playback of those packets. For the first two steps in the search, the log file playback causes both messages to display.

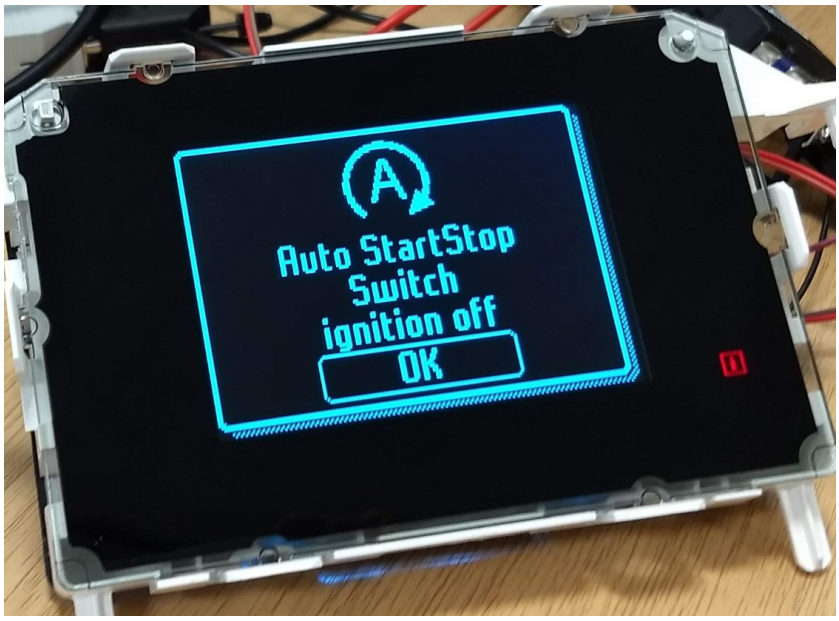


Fig. 9.11 2nd display message seen during fuzz testing

The second, and subsequent divisions of the log file cause only one, or no messages to be displayed on the ECU's screen.

However, when down to playing back the last few data packets in the search, the message display is inconsistent. Several attempts at playing back the last few log file search lines would, or would not, cause a message to be displayed.

During the search for the CAN packet that displays *Auto StartStop Switch ignition off*, at the third division of the playback (lines 653 to 1303), another message was seen. The message was *Passenger door* with an overlay of *Preset 23 stored*, see Figure 9.12.

This behaviour was confirmed. Playing back all or half the file showed the two messages *Park brake applied* and *Auto StartStop Switch ignition off*. Isolating those messages individually to one-quarter of the log file showed either *Park brake applied*, or *Passenger door/Preset 23 stored*. As the search progressed the message displayed changed again, from *Passenger door/Preset 23 stored* to just *Passenger door open OK*, see Figure 9.13.

It would later be found that this behaviour is because individual bits in the data bytes are responsible for different messages, see Section 9.11, and the CAN packet transmission rate affects the display ECU operation. The impact of packet transmission rate is briefly examined in Section 9.11.3.

9.8.2 Resolving inconsistent CAN packet search results

As already noted in Section 9.5, when the ECU is sent a CAN packet it wakes from a standby mode (and transmitting 12 different packets for 5.5 seconds). This standby mode was one reason for the message display inconsistency. A CAN packet is essentially ignored whilst the display ECU is

Table 9.6 Results on a search on a fuzz testing log file for the message 'Park brake applied', the playback observed 2 messages, to begin with, and had inconsistent results once four data packets remained

Playback start line	*Playback end line*	*No. messages seen*
1	2606	2
1	1303	2
1	652	1
1	326	0
490	652	1
571	652	1
612	652	0
571	611	1
591	611	0
571	590	1
571	581	0
582	590	1
587	590	0
582	586	1
584	586	Inconsistent
582	583	Inconsistent

Table 9.7 Results on a search on a fuzz testing log file for the message 'Auto StartStop Switch ignition off', again, the playback observed 2 messages to begin with and had inconsistent results once 4 packets remained

Playback start line	*Playback end line*	*No. messages seen*
1	2606	2
1	1303	2
1	652	1
653	1303	1
653	978	1
653	815	1
653	734	1
653	693	0
694	733	1
694	713	0
714	733	1
714	733	1
714	723	0
724	733	1
724	728	0
729	733	1
729	731	Inconsistent
732	733	Inconsistent

Fig. 9.12 Different display message seen during the search for message two

Fig. 9.13 The passenger door message without the overlay during the search for the second message

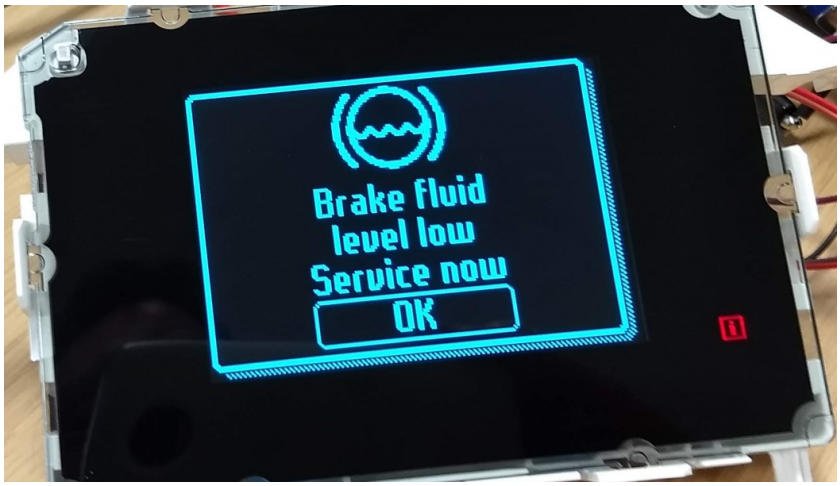


Fig. 9.14 A brake fluid message seen during CAN packet discovery

in standby. Furthermore, as mentioned in the previous Section 9.8.1, the rate of the CAN packet transmission can also effect on the message being display.

To resolve the inconsistent CAN packet search results the displayed ECU has to be awake. Thus, for component testing purposes a *keep alive* facility is likely to be required by the CAN fuzzer when testing ECUs. This was noted in the testing of the gateway ECU in Chapter 7, and further provides evidence that a component functions as part of a larger system.

The packets from the inconsistent log file lines were sent to the display ECU using the CAN fuzzer's single packet transmission facility, whilst ensuring the ECU was not in standby (by sending the packet twice). This found two packets that caused the display ECU to operate:

1. CAN id 793 with data bytes `94 41 1D DF 2B 9B AE E3` (hex), for the *Park brake applied* message.

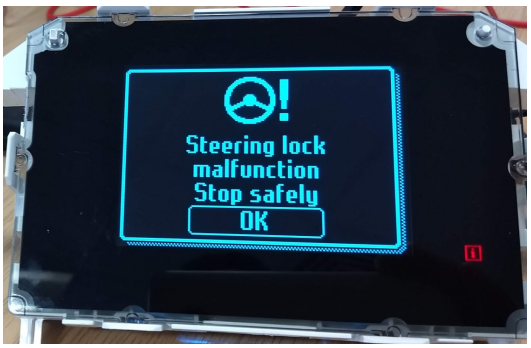
2. CAN id 752 with data bytes `57 29 61 43 CB 7B 79 59` (hex), caused the messages *Auto StartStop Switch ignition off* or *Passenger door open*.

Sending the same CAN packet several times on its own resulted in different messages being displayed. For example, for CAN packet with id 752 the message *Brake fluid level low Service now* was also seen, see Figure 9.14.

A further variation on the transmission rate of CAN packet 752 caused additional messages to be displayed, see Figures 9.15 and 9.16 for examples, including the message *Auto StartStop Switch ignition off*, which stopped being displayed during the search.

In summary, the observations noted when finding the message displaying CAN packets are:

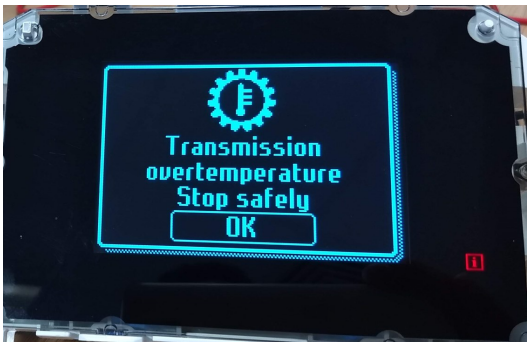
1. As mentioned in Section 9.4, the display ECU needs to be awake and active to respond to CAN packets.



(a) Steering lock malfunction message

(b) Steering lock service message

Fig. 9.15 Steering lock messages seen using CAN packet id 752



(a) Transmission over temperature message

(b) Transmission service message

Fig. 9.16 Transmission messages seen using CAN packet id 752

Table 9.8 ECU displayed messages from CAN packet id 739, data bytes shown in hex

Data bytes	*Message*
94 41 1D DF 2B 9B AE E3	Park brake applied
94 00 00 00 00 00 00 00	Check tyre pressures
44 00 00 00 00 00 00 00	MyKey Vehicle at top speed
55 00 00 00 00 00 00 00	Active City Stop Sensor blocked Clean screen
55 00 00 00 00 00 00 00	Active City Stop not available

2. A single CAN packet controls the display of several messages, but the rate of CAN packet transmission can affect which of several possible messages are displayed.

9.9 Testing individual CAN packet bytes from found messages

During the fuzz testing and search, different messages had been seen on the ECU's screen, including messages not applicable to the lab car, see Figure 9.17. The discovered CAN packets responsible, with ids 793 and 752, were examined in further detail.

The CAN data packets in the fuzz testing are randomly generated. The first line in Table 9.8 shows CAN packet with id 739 that generated the *Park brake applied message*. Using the fuzzer's single shot transmission ability, see Figure 9.18, an attempt to isolate the byte causing the message to display was performed. First, all bytes except one were zeroed, see the second line in Table 9.8, and a CAN packet was sent. The result was another message, *Check tyre pressures* (see top left in Figure 9.17).

Attempts to isolate the data responsible for a particular message was performed. The bit patterns `01010101` (41 hex) and `10101010` (55 hex) turn on and off each bit in the byte. This revealed three more messages, see lines 3 to 5 in Table 9.8. For the data bytes `55 00 00 00 00 00 00 00` (hex) one message is seen on the first transmission of the packet, then subsequent transmissions display the second message. (The power to the ECU needed to be turned off and on to get it to display the original message again.)

In attempting to isolate the data that causes the originally observed *Park brake applied* message seen during the fuzz testing, several messages were being found. A systematic approach to isolate the data causing such messages was required.

9.10 Single byte testing for individual display messages

The identified CAN data packets have eight bytes of data. Testing the effect of the byte values can be done in three different ways:

1. Treat the individual bits in a byte as flags. Testing involves setting and clearing different bits in each of the packet's bytes.

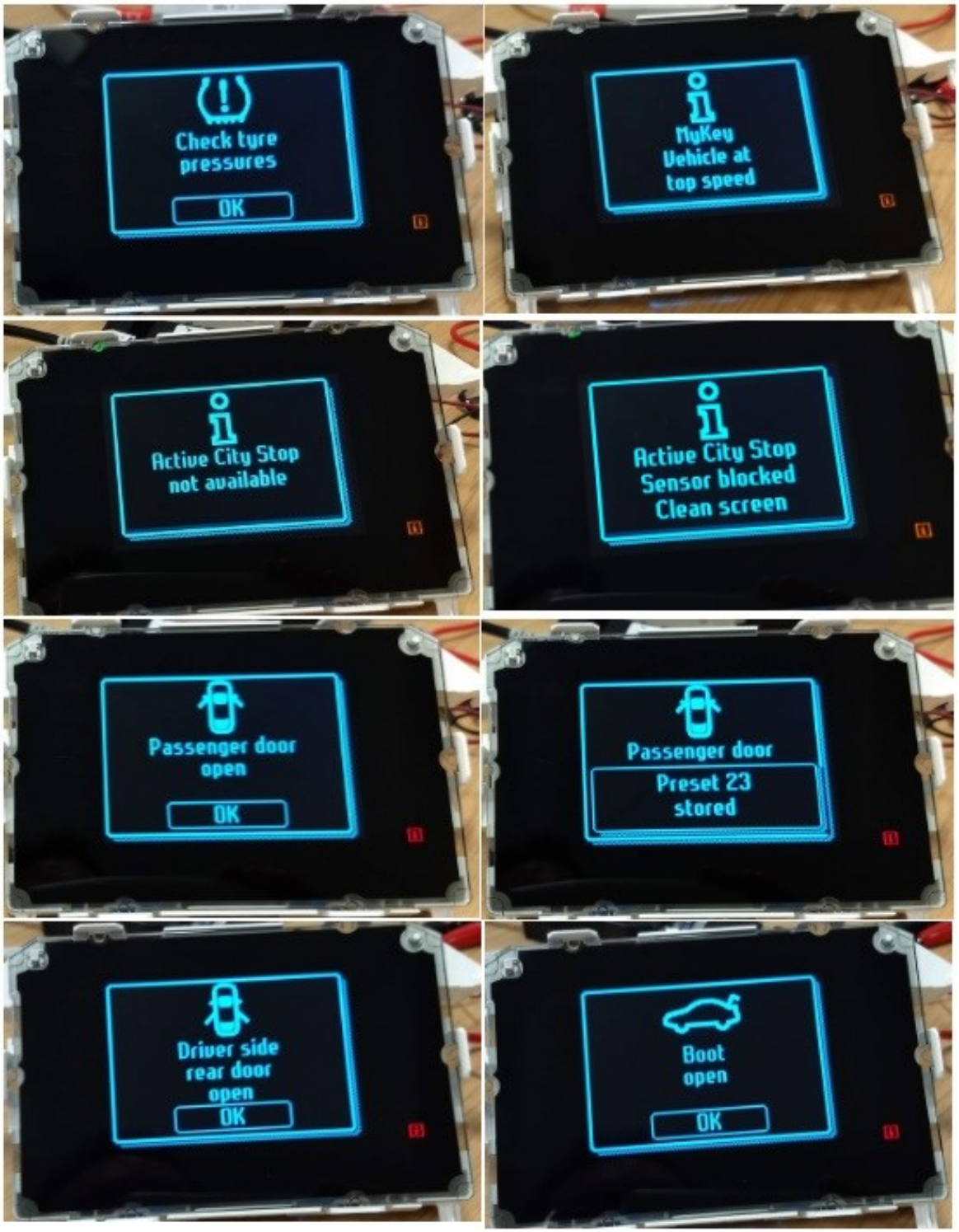


Fig. 9.17 Some display messages discovered from log file analysis via a search, the first four messages and last but one message relate to systems not present on the lab vehicle, for example, the vehicle does not have rear doors, even though the ECU functions in the lab car

2. Treat the data bytes as integer numbers and increment the values from 0 to 255.
3. Generate random values over a byte's range from 0 to 255.

Each of the methods has issues. The main problem, which was discussed in Section 6.5.1 in Chapter 6, is a combinatorial explosion issue. Each byte has 2^8 values and there are eight bytes in the standard CAN packets used in the lab vehicles. This gives a total number of discreet packet combinations for one CAN id as 2^{64}, or nearly 18.5 Exa (10^{18}) possible values.

The use of random bytes to investigate a particular packet's properties is not efficient, due to the number of possible packet combinations, unless each byte is considered isolated. In which case randomising over the range 0 to 255 for the eight bytes gives 2048 combinations ($256 * 8 = 2048$). However, even for the 2048 possible tests some form of automation is required.

When viewing the bytes as bit flags the number of combinations is vastly simplified, with eight bits per byte and eight bytes, there are only 64 bits to test. However, if combinations of bits are used then the number of combinations can explode.

Since restrictions are required on the testing to limit the effects of a combinational explosion, the experiments start on a restricted range of values to help determine the best approach. The first experiment tests bit settings and second experiment tests combinations of bits, which also equates to testing the first few integer values.

9.10.1 Experiment to test packet bit settings

The following method is used to test the individual bit settings in a CAN packet for the display ECU:

1. The experiment starts by setting the contents of the data bytes in the CAN packet to zero.
2. The CAN packet is transmitted to the display ECU.
3. The screen on the display ECU is observed for any reaction.
4. Each bit is set to one in order (starting at the first bit in the first byte).
5. If the last bit has been set then finish, otherwise go to step 2.

9.10.2 Experiment to test CAN packet byte values

The following method is used to test the individual byte values in a CAN packet for the display ECU:

1. The experiment starts by setting the contents of the data bytes in the CAN packet to zero.
2. The CAN packet is transmitted to the display ECU.
3. The screen on the display ECU is observed for any reaction.
4. Each byte is increment by one in order (starting at the first byte).
5. If byte values reached a required number then finish, otherwise go to step 2.

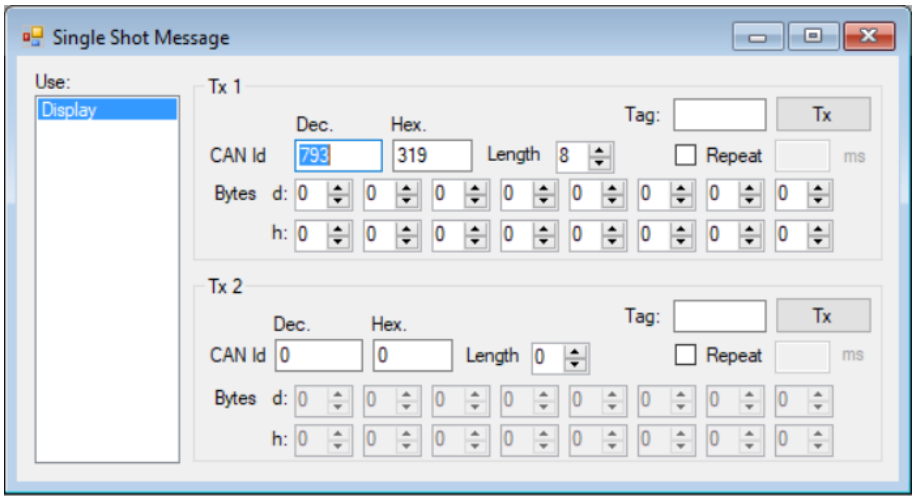


Fig. 9.18 Testing for individual display messages

9.10.3 Modifying the CAN fuzzer to aid the experiments

To allow for easy manipulation and transmission of the CAN packets to the display ECU, a new single packet transmission facility was added to the CAN fuzzer. The new *Single Shot* interface was based upon functionality used in Chapter 6, the lock/unlock app seen in Figure 6.5.

The single shot interface allows for a CAN packet to be defined, with any value set for the data bytes. The data values can be incremented and decremented using the arrows next to them, see Figure 9.18.

9.10.4 Known display ECU messages

The fuzz testing and search revealed some messages, see Figure 9.17. Other messages were seen when operating the vehicle, Table 9.1. To determine the range of possible messages that could be found the user manual for the lab car was examined, available from the manufacturer's customer website (the reference has been redacted due to commercial disclosure reasons). The car's user manual lists 78 possible messages that the display may show. Some of these relate to options and equipment that may not be present due to the vehicle model. The messages from the user manual are listed in Table 9.9. However, only two of the messages initially observed on the lab car's original screen, see Table 9.1, are listed in the user manual (*Driver door open* and *Passenger door open*), one message is similar, *To start press clutch* compared to *Press clutch to start*. From the other eleven initially seen messages eight are related to the climate control system.

Table 9.9 Messages listed in the owner's manual (reference redacted due to responsible disclosure)

#	*Message text*	#	*Message text*
1	Active City Stop Auto braking	40	MyKey Park aid cannot be deactivated
2	Active City Stop Sensor blocked Clean screen	41	Auto wiper/lights malfunction Service required
3	Active City Stop not available	42	Park brake applied
4	Active City Stop off	43	Traction control off
5	Airbag malfunction Service now	44	Sport mode
6	Alarm triggered Check vehicle	45	Electronic stability control off
7	Interior Scan deactivated	46	ABS malfunction Service now
8	Driver door open	47	ESP malfunction Next service
9	Alarm system mal... Service required	48	Tyre monitor mal... Service required
10	Driver side rear door open	49	Engine start pending Please wait
11	Passenger door open	50	Engine start cancelled
12	Passenger side rear door open	51	Diesel filter overloaded See manual
13	Boot open	52	Press brake to start
14	Bonnet open	53	Press clutch to start
15	Engine preheating	54	Press brake and clutch to start
16	Immobiliser malfunction Service now	55	Cranking time exceeded
17	Transmission overtemperature Stop safely	56	Auto StartStop Press a pedal to start engine
18	Key not detected	57	Auto StartStop Switch ignition off
19	Key outside car	58	Auto StartStop Manual start required
20	Key Battery low Replace battery	59	Auto StartStop Select neutral
21	Turn ignition off Use POWER button	60	Power steering malfunction Service now
22	Buckle up to unmute audio	61	Steering lock malfunction Stop safely
23	Transmission not in Park	62	Steering lock malfunction Service now
24	Close boot or use spare key	63	Check tyre pressures
25	Steering lock engaged Turn steering wheel	64	Tyre pressure sys malfunction Service required
26	Left indicator malfunction Change bulb	65	Hill start assist not available
27	Right indicator malfunction Change bulb	66	Tyre sensors not detected Check manual
28	Brake fluid Level low Service now	67	Transmission malfunction Service now
29	Brake system malfunction Stop safely	68	ESP off
30	Engine oil pressure low Stop safely	69	Use brake Stop safely
31	Engine malfunction Service now	70	Vehicle not in Park Select P
32	MyKey ESC cannot be deactivated	71	Select N or P to start
33	Engine oil change due Service required	72	Select N to start
34	MyKey vehicle at top speed	73	Door open apply brake
35	MyKey active Drive safely	74	Transmission hot Stop or speed up
36	MyKey Speed limited to XX mph	75	Transmission hot Wait...
37	MyKey Speed limited to XX km/h	76	Transmission ready
38	MyKey Check speed Drive safely	77	Press brake to unlock selector lever
39	MyKey Vehicle near top speed	78	Selector lever unlocked

9.11 Results varying individual packet bit and bytes values

The initial fuzz testing and search discovered two CAN ids that resulted in messages being shown on the display ECU. The first experiment, using the method described above in Section 9.10.1, was to set each bit position in the CAN packet's data for the CAN packets, with ids of 793 and 752.

9.11.1 CAN packet ids 793 and 752 testing results for single bit settings

The results from setting each bit in turn for CAN packets with ids 793 and 752 are split over in six tables. In each case, only a single bit in one of the data bytes was being set. All the other bits in the remaining seven data bytes were turned off (i.e. the byte value was zero):

- CAN packet id 793 - Table 9.10 shows bit setting results for bytes 1 to 3.
- CAN packet id 793 - Table 9.11 shows bit setting results for bytes 4 to 6.
- CAN packet id 793 - Table 9.12 shows bit setting results for bytes 7 and 8.
- CAN packet id 752 - Table 9.13 shows bit setting results for bytes 1 to 3.
- CAN packet id 752 - Table 9.14 shows bit setting results for bytes 4 to 6.
- CAN packet id 752 - Table 9.15 shows bit setting results for bytes 7 and 8.

9.11.2 CAN packet ids 793 and 752 results discussion for single bit settings

There are several observations that can be made from the results in Section 9.11.1. It can be seen that for CAN packet 793, out of the total 64 bit positions there are 22 bits that result in a message being displayed. However, one message is repeated twice, the message *Park brake applied*. Furthermore, two messages are not present in the user manual, the message *Engine on OK*, and the message *Selector lever unlocked*. Thus, the bit set testing for CAN packet 793 revealed 20 of the 78 known messages listed in the user manual.

For CAN packet 752, out of a total of 64 bit positions there are 55 bit positions that result in a message being displayed. However, not all 55 bit positions result in a unique message. Two messages that are displayed via CAN packet 752 are also displayed by CAN packet 793. Two messages are repeated by different bytes. One message is not listed in the user manual. Three messages have similar text to messages in the user manual. Two messages are repeated by several bits. Here are the details:

1. The message *Auto StartStop engine starting*, displayed via bit 8 in byte 3 of CAN packet 752, is not listed in the user manual.
2. The message *Bonnet open* is repeated by CAN packet 752, displayed via by bit 2 in byte 3 and bit 3 in byte 4.

Table 9.10 Display ECU CAN packet id 793 bit setting results for bytes 1 to 3, all the other bytes in the eight byte packet were set to zero

Byte 1	*Byte 2*	*Byte 3*	*Message and #*
0000 0000	0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	0000 0000	Press brake and clutch to start (54)
0000 0010	0000 0000	0000 0000	Press brake to start (52)
0000 0100	0000 0000	0000 0000	Press clutch to start (53)
0000 1000	0000 0000	0000 0000	Active City Stop Auto braking (1)
0001 0000	0000 0000	0000 0000	Active City Stop not available (3)
0010 0000	0000 0000	0000 0000	Active City Stop Sensor blocked Clean screen (2)
0100 0000	0000 0000	0000 0000	Check tyre pressures (63)
1000 0000	0000 0000	0000 0000	Tyre pressure sys malfunction Service required (64)
0000 0000	0000 0001	0000 0000	Engine on OK (not listed)
0000 0000	0000 0010	0000 0000	MyKey active Drive safely (35)
0000 0000	0000 0100	0000 0000	MyKey Speed limited to 160 km/h (37)
0000 0000	0000 1000	0000 0000	MyKey vehicle at top speed (34)
0000 0000	0001 0000	0000 0000	none
0000 0000	0010 0000	0000 0000	none
0000 0000	0100 0000	0000 0000	none
0000 0000	1000 0000	0000 0000	none
0000 0000	0000 0000	0000 0001	MyKey Check speed Drive safely (38)
0000 0000	0000 0000	0000 0010	none
0000 0000	0000 0000	0000 0100	none
0000 0000	0000 0000	0000 1000	none
0000 0000	0000 0000	0001 0000	none
0000 0000	0000 0000	0010 0000	none
0000 0000	0000 0000	0100 0000	none
0000 0000	0000 0000	1000 0000	none

Table 9.11 Display ECU CAN packet id 793 bit setting results for bytes 4 to 6, all the other bytes in the eight byte packet were set to zero

Byte 4	*Byte 5*	*Byte 6*	*Message and #*
0000 0000	0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	0000 0000	MyKey Park aid cannot be deactivated (40)
0000 0010	0000 0000	0000 0000	MyKey ESC cannot be deactivated (32)
0000 0100	0000 0000	0000 0000	none
0000 1000	0000 0000	0000 0000	none
0001 0000	0000 0000	0000 0000	none
0010 0000	0000 0000	0000 0000	none
0100 0000	0000 0000	0000 0000	none
1000 0000	0000 0000	0000 0000	none
0000 0000	0000 0001	0000 0000	Press brake to unlock selector lever (77)
0000 0000	0000 0010	0000 0000	Cranking time exceeded (55)
0000 0000	0000 0100	0000 0000	none
0000 0000	0000 1000	0000 0000	none
0000 0000	0001 0000	0000 0000	none
0000 0000	0010 0000	0000 0000	none
0000 0000	0100 0000	0000 0000	none
0000 0000	1000 0000	0000 0000	none
0000 0000	0000 0000	0000 0001	none
0000 0000	0000 0000	0000 0010	none
0000 0000	0000 0000	0000 0100	none
0000 0000	0000 0000	0000 1000	none
0000 0000	0000 0000	0001 0000	none
0000 0000	0000 0000	0010 0000	none
0000 0000	0000 0000	0100 0000	Transmission not in Park (23)
0000 0000	0000 0000	1000 0000	Selector lever unlocked (not listed)

Table 9.12 Display ECU CAN packet (id 793) bit setting results for bytes 7 and 8, all the other bytes in the eight byte packet were set to zero

Byte 7	*Byte 8*	*Message and #*
0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	Park brake applied (42)
0000 0010	0000 0000	none
0000 0100	0000 0000	none
0000 1000	0000 0000	none
0001 0000	0000 0000	none
0010 0000	0000 0000	none
0100 0000	0000 0000	none
1000 0000	0000 0000	none
0000 0000	0000 0001	none
0000 0000	0000 0010	none
0000 0000	0000 0100	none
0000 0000	0000 1000	none
0000 0000	0001 0000	none
0000 0000	0010 0000	MyKey Vehicle near top speed (39)
0000 0000	0100 0000	none
0000 0000	1000 0000	Park brake applied (42)

3. The message *Boot open* is repeated by CAN packet 752, displayed via by bit 1 in byte 3 and bit 4 in byte 4.

4. The message *Key Battery low Replace battery* is repeated by CAN packet 752, displayed via by bit 3 in byte 5 and bit 6 in byte 5.

5. The message *Press brake to start* is repeated, displayed by bit 2 in byte 1 of CAN packet 793, and bit 2 in byte 5 of CAN packet 752.

6. The message *Press clutch to start* is repeated, displayed by bit 3 in byte 1 of CAN packet 793, and bit 1 in byte 5 of CAN packet 752.

7. The message *Diesel filter overloaded Refer to manual*, displayed by bit 5 in byte 1 of CAN id 752, is similar to the message *Diesel filter overloaded See manual* in the user manual.

8. The message *Transmission hot Wait 4 mins*, displayed by bit 8 in byte 8 of CAN id 752, is similar to *Transmission hot Wait...* in the user manual.

9. The message *Switch ignition off Press POWER*, displayed by bit 8 in byte 6 of CAN id 752, is similar to *Turn ignition off Use POWER button* in the user manual.

10. The message *Alarm triggered Check vehicle* is displayed by bits 1 to 4 of byte 8 in the CAN packet 752.

Table 9.13 Display ECU CAN packet id 752 bit setting results for bytes 1 to 3, all the other bytes in the eight byte packet were set to zero

Byte 1	*Byte 2*	*Byte 3*	*Message and #*
0000 0000	0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	0000 0000	none
0000 0010	0000 0000	0000 0000	Transmission overtemperature Stop safely (17)
0000 0100	0000 0000	0000 0000	Transmission malfunction Service now (67)
0000 1000	0000 0000	0000 0000	Engine malfunction Service now (31)
0001 0000	0000 0000	0000 0000	Diesel filter overloaded Refer to manual (resembles 51)
0010 0000	0000 0000	0000 0000	Engine oil change due Service required (33)
0100 0000	0000 0000	0000 0000	Engine oil pressure low Stop safely (30)
1000 0000	0000 0000	0000 0000	Hill start assist not available (65)
0000 0000	0000 0001	0000 0000	Airbag malfunction Service now (5)
0000 0000	0000 0010	0000 0000	Brake system malfunction Stop safely (29)
0000 0000	0000 0100	0000 0000	Brake fluid Level low Service now (28)
0000 0000	0000 1000	0000 0000	ABS malfunction Service now (46)
0000 0000	0001 0000	0000 0000	none
0000 0000	0010 0000	0000 0000	none
0000 0000	0100 0000	0000 0000	Power steering malfunction Service now (60)
0000 0000	1000 0000	0000 0000	none
0000 0000	0000 0000	0000 0001	Boot open (13)
0000 0000	0000 0000	0000 0010	Bonnet open (14)
0000 0000	0000 0000	0000 0100	none
0000 0000	0000 0000	0000 1000	none
0000 0000	0000 0000	0001 0000	none
0000 0000	0000 0000	0010 0000	none
0000 0000	0000 0000	0100 0000	none
0000 0000	0000 0000	1000 0000	Auto StartStop engine starting (not listed)

Table 9.14 Display ECU CAN packet id 752 bit setting results for bytes 4 to 6, all the other bytes in the eight byte packet were set to zero

Byte 4	*Byte 5*	*Byte 6*	*Message and #*
0000 0000	0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	0000 0000	Left indicator malfunction Change bulb (26)
0000 0010	0000 0000	0000 0000	Right indicator malfunction Change bulb (27)
0000 0100	0000 0000	0000 0000	Bonnet open (14)
0000 1000	0000 0000	0000 0000	Boot open (13)
0001 0000	0000 0000	0000 0000	Passenger side rear door open (12)
0010 0000	0000 0000	0000 0000	Driver side rear door open (10)
0100 0000	0000 0000	0000 0000	Passenger door open (11)
1000 0000	0000 0000	0000 0000	Driver door open (8)
0000 0000	0000 0001	0000 0000	Press clutch to start (53)
0000 0000	0000 0010	0000 0000	Press brake to start (52)
0000 0000	0000 0100	0000 0000	Key Battery low Replace battery (20)
0000 0000	0000 1000	0000 0000	Auto StartStop Press a pedal to start engine (56)
0000 0000	0001 0000	0000 0000	Alarm system malfunction Service required (9)
0000 0000	0010 0000	0000 0000	Key Battery low Replace battery (20)
0000 0000	0100 0000	0000 0000	Interior Scan deactivated (7)
0000 0000	1000 0000	0000 0000	Auto wiper/lights malfunction Service required (41)
0000 0000	0000 0000	0000 0001	Auto StartStop Manual start required (58)
0000 0000	0000 0000	0000 0010	Close boot or use spare key (24)
0000 0000	0000 0000	0000 0100	Steering lock engaged Turn steering wheel (25)
0000 0000	0000 0000	0000 1000	Steering lock malfunction Stop safely (61)
0000 0000	0000 0000	0001 0000	Steering lock malfunction Service now (62)
0000 0000	0000 0000	0010 0000	Key outside car (19)
0000 0000	0000 0000	0100 0000	Key not detected (18)
0000 0000	0000 0000	1000 0000	Switch ignition off Press POWER (similar to 21)

Table 9.15 Display ECU CAN packet id 752 bit setting results for bytes 7 and 8, all the other bytes in the eight byte packet were set to zero

Byte 7	*Byte 8*	*Message and #*
0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	Door open apply brake (73)
0000 0010	0000 0000	Select N to start (72)
0000 0100	0000 0000	Press brake to start (52)
0000 1000	0000 0000	Auto StartStop Select neutral (59)
0001 0000	0000 0000	Select N or P to start (71)
0010 0000	0000 0000	Vehicle not in Park Select P (70)
0100 0000	0000 0000	Auto StartStop Switch ignition off (57)
1000 0000	0000 0000	Immobiliser malfunction Service now (16)
0000 0000	0000 0001	Alarm triggered Check vehicle (6)
0000 0000	0000 0010	Alarm triggered Check vehicle (6)
0000 0000	0000 0100	Alarm triggered Check vehicle (6)
0000 0000	0000 1000	Alarm triggered Check vehicle (6)
0000 0000	0001 0000	Transmission hot Stop or speed up (74)
0000 0000	0010 0000	Transmission hot Stop or speed up (74)
0000 0000	0100 0000	Transmission ready (37)
0000 0000	1000 0000	Transmission hot Wait 4 mins (similar to 75)

11. The message *Transmission hot Stop or speed up* is displayed by bits 5 and 6 of byte 8 in the CAN packet 752.

Taking the results from the bit testing, and excluding repeated messages, and the one unlisted message, then at least 44 messages are displayed via CAN packet 752. This means that CAN packets 752 and 793 are capable of displaying 64 of the 78 messages listed in the user manual, suggesting that one or more additional packets could be responsible for the remaining messages if combined bits, covered in Section 9.11.3 below did not introduce additional messages.

9.11.3 CAN packet 793 testing results for byte values

Having determined that bit positions within a CAN packet's data do result in messages being shown on the display ECU, the next test was to set byte values.

Using the method described in Section 9.10.2 the first 50 values of the first byte were tried for CAN packet 793. The first 50 values are covered by the first byte in the payload of a packet. The results from trying values 0 to 49 are shown in Table 9.16, where each number in the messages column corresponds to a message listed in Table 9.9.

The value set for byte 1 can be equivalent to setting multiple messages bits, for example, in Table 9.10 bit 2 corresponds to *Press brake to start*, and bit 3 corresponds to *Press clutch to start*. A

Table 9.16 Display ECU CAN packet (id 793) value setting results for byte 1, all other 7 bytes set to zero, the messages displayed corresponded to the bits set

Value	*Msg(s) #*	*Value*	*Msg(s) #*
0	n/a	25	3, 53, 1
1	54	26	3, 52, 1
2	52	27	3, 54, 52, 1
3	52, 54	28	3, 53, 1
4	53	29	3, 54, 53, 1
5	53, 54	30	3, 53, 52, 1
6	52, 53	31	3, 54, 53, 52, 1
7	52, 53, 54	32	2
8	1	33	2, 54
9	54, 1	34	2, 52
10	52, 1	35	2, 54, 52
11	52, 54, 1	36	2, 53
12	53, 1	37	2, 54, 53
13	53, 54, 1	38	2, 53, 52
14	52, 53, 1	39	2, 54, 53
15	52, 53, 54, 1	40	2, 1
16	3	41	2, 54, 1
17	3, 54	42	2, 52, 1
18	3, 52	43	2, 54, 52, 1
19	3, 54, 53	44	2, 53, 1
20	3, 53	45	2, 53, 1
21	3, 54, 53	46	2, 54, 53, 1
22	3, 53, 52	47	2, 54, 53, 52, 1
23	3, 54, 53, 52	48	2, 3
24	3, 1	49	2, 3, 54

value of 6 in byte 1 is equivalent to both bit 2 and 3 being set. The results show that each set bit can cause its equivalent message to be displayed. However, there are a couple of conflicts.

Transmitting multiple copies of the same data

The value was set in byte 1, and all other bytes set to zero. This was then sent repeatedly using the CAN fuzzer's single shot facility. This was done by pressing the single shot button or using the fuzzer's auto-repeat facility. When two or more data byte bits are set, then a fast or slow transmission rate would vary which of the expected messages were displayed.

Rate of packet transmission

Packet transmission rate would affect how long a message was shown. At the bottom of each message that is shown on the display ECU a small progress bar can be seen. It updates each second and after ten seconds the bar is full or nearly full, at which point the message clears. The rate of CAN packet transmission can affect this progress bar functionality. When the display ECU receives packets it will perform one of several actions:

1. the messages being displayed will remain and the progress bar is reset;
2. the screen will clear (revert to blank);
3. a new message is displayed;
4. the current message will display for the ten seconds as the progress bar fills.

As an example, the display ECU was observed when sent packets at different rates. The value of 3 was set in the first byte of CAN packet 793, and all other byte values were zero. The value of 3 has the bits set for messages *Press brake to start* and *Press brake and clutch to start*. The result of sending this packet to the display ECU at four different transmission rates is shown in Table 9.17. At all the tested transmission rates, when the transmission of the CAN packets stop, the screen will clear and go blank straightaway.

Without having access to the source code for the display ECU, it is difficult to understand the variation in the display ECU's behaviour due to different packet transmission rates. It is likely that a single rate was designed for the vehicle. At a 100ms rate (10 packets per second) the display ECU shows a ten-second message with a short pause, this behaviour appeared reasonable. Furthermore, stopping packet transmissions stops the display of messages. Knowing these factors aids with potential attacks aimed at manipulating the operation of the ECU display.

9.11.4 CAN packet 752, byte value setting tests

The byte value settings test was tried for CAN packet 752 to confirm the same behaviour. CAN packet 752 had the value 6 set in the first byte, the remaining bytes being set to the value of zero. The

Table 9.17 The CAN packet transmission rate affects how the messages are displayed on the ECU, in this example the CAN id was 793, and the eight bytes of data were `03 00 00 00 00 00 00 00`

Packets per second	*Message display effect*
0.5	The message *Press brake to start* is displayed for one second, the screen clears, the message is displayed again, occasionally the message *Press brake and clutch to start* is shown. The progress bar does not get a chance to fill.
1	Displays the message *Press brake to start* for ten seconds as the progress bar fills. The screen clears for six seconds. This process repeats. At the start of the transmission, the message *Press brake and clutch to start* is shown once.
10	Displays the message *Press brake to start* for ten seconds as the progress bar fills. The screen clears for four seconds. This process repeats.
100	Displays the message *Press brake to start* for ten seconds as the progress bar fills. The screen blanks for twelve seconds. The process repeats. At the start of the transmission, the message *Press brake and clutch to start* is shown once for ten seconds.

value of 6 corresponds to the 2nd and 3rd bits of byte 1 being set and the messages *Transmission overtemperature Stop safely* and *Transmission malfunction Service now* for display on the ECU's screen (as shown in Figure 9.16).

Transmitting CAN packet 752 with byte 1 set to value 6 did result in those two messages being display. Thus, the value behaviour is as for CAN packet 793. Again, varying the packet transmission rate did affect whether or not either message were displayed or only one of the messages was displayed.

9.12 Testing packet length variation

The two CAN packets that showed messages on the display ECU were discovered via fuzz testing. During that fuzz testing, the CAN fuzzer was configured to vary the CAN id and data values. The fixed variable was the data length, set to 8, the value observed on the lab vehicle's CAN networks. Next, an experiment was performed to vary the data length of the CAN packet for the two found packets, to see if it influenced the ECU.

9.12.1 Method for the packet length variation

The following method is used to test the variation in the data length for CAN packets 793 and 752.

1. The CAN fuzzer's single shot function was configured to transmit CAN packets 793 and 752 to the display ECU.

2. The data length of the CAN packet was varied from a maximum of 8 to a minimum of 0.

Table 9.18 Unexpected messages seen when decreasing packet data length for CAN ids 793 and 752, zeroed eight byte data values generate no messages and is used as the baseline, when more than one message was seen they were viewed with consecutively sent packets and/or seen one after another from one sent packet, however, the byte values that would normally be used to generate the seen messages would be present beyond the end of the data in the sent packet, as indicated by the underlines, this indicates a buffer overflow problem with the ECU's software.

Id	*Data length*	*Message(s) seen*	*Sent vs decoded data (hex)*
793	8	none	00 00 00 00 00 00 00 00
793	7	42	00 00 00 00 00 00 00 80
793	6	42	00 00 00 00 00 00 01 00
793	5	42	00 00 00 00 00 00 01 00
793	4	*Selector lever unlocked*, 42	00 00 00 00 00 80 01 00
793	3	77, 23	00 00 00 00 01 40 00 00
793	2	40	00 00 00 01 00 00 00 00
793	1	34 and *Engine on OK*	00 09 00 00 00 00 00 00
793	0	64, 3 and *Engine on OK*	90 01 00 00 00 00 00 00
752	8	none	00 00 00 00 00 00 00 00
752	7	6	00 00 00 00 00 00 00 0F
752	6	6	00 00 00 00 00 00 00 0F
752	5	62	00 00 00 00 00 01 00 00
752	4	62, 61, 57	00 00 00 00 00 18 40 00
752	3	11, 62	00 00 00 40 00 01 00 00
752	2	12, 11, 13	00 00 01 48 00 00 00 00
752	1	28, 10	00 04 00 20 00 00 00 00
752	0	67, 17, 28, 29, 10	06 06 00 20 00 00 00 00

3. All the data values were fixed at zero.

4. The display ECU was observed for a reaction when the configured CAN packets were transmitted.

9.12.2 Results for the packet length variation

The results from vary the data length for CAN packets 793 and 752 are shown in Table 9.18. When the data length is set to 8, with all bytes set to a value of zero, the display ECU does not show any messages. This is expected and was seen in the bit setting experiments (see Section 9.11.1). However, reducing the length of the data sent in the packet, whilst keeping the data values set to zero, does result in messages being displayed.

For example, transmitting the CAN packet 793 with the eight bytes of data 00 00 00 00 00 00 00 80 (hex) results in the message *Park brake applied* on the display ECU. However, this message is displayed when CAN packet 793 is sent with the 7 bytes of data 00 00 00 00 00 00 00 (hex). Similar results occur at all data lengths. This indicates that the software in the display ECU is probably

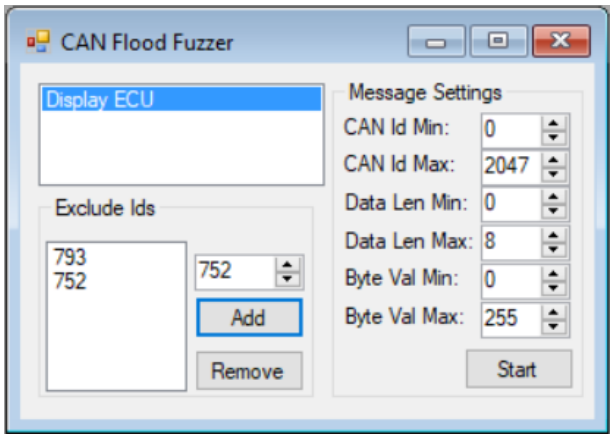


Fig. 9.19 Fuzz testing CAN with an id exclusion list

programmed to always read eight bytes from the data buffer. If this buffer overflow error was found prior to production, it would require a bug being raised. The software in the ECU should ignore any message that was not of the correct length.

9.13 Exclusions lists for fuzz testing and CAN packet 753

The use of the CAN fuzzer to determine component functionality has shown to be useful. Though, as previously noted, the discovery of functionality is dependent upon the ability to obtain a cyber-physical reaction from the component. In the case of the display ECU, it is a visible response being observed. Further functional discovery requires repeating the fuzz testing whilst excluding the CAN packets that have been determined. This required a modification to the CAN fuzzer to prevent the transmission of packets with a known id. In Figure 9.19 the CAN fuzzer can be seen configured with packet ids to exclude from the fuzz testing.

Having excluded the generation of CAN packets with ids 752 and 793 the next fuzz testing observed four additional messages:

- *OFF Traction control off*, see message 43 in Table 9.9;
- *SYNC connection error*, not explicitly listed in the vehicles user manual;
- `HH:MM DD:MM:YYYY`, a time and data template;
- `00:09 01:01:2000`, a time and date display.

The search for the CAN packet that produces the message *OFF Traction control off* is shown in Table 9.19. As for the previous searches, once the search narrows to the last four packets the results become inconsistent, variable based on the transmission frequency of the CAN packets. However, as for the other searches, the CAN fuzzer's single packet transmission functionality identified the CAN

Table 9.19 Results on a search on a fuzz testing log file for another display ECU CAN packet

Playback start line	*Playback end line*	*No. messages seen*
1	1343	3
1	671	0
672	1343	2
672	1007	0
1008	1343	2
1008	1174	3
1008	1091	0
1092	1174	2
1133	1174	2
1133	1153	0
1154	1174	2
1154	1164	2
1154	1159	0
1160	1164	2
1160	1162	Inconsistent
1163	1164	Inconsistent

packet responsible for the shown message, packet id 753 with the data `1C 39 32 CF DA 92 FE 00` (hex).

For the CAN packets with ids 752 and 793, an experiment to set individual bits in the data bytes revealed many of the messages that the ECU display shows, see Tables 9.13 to 9.15. However, for the next discovered message, id 753, the setting of individual bits did not work in the same manner:

1. For data `00 00 00 00 00 00 00 00` (hex) to `08 00 00 00 00 00 00 00`, all zeros and the first five bits, showed the message *OFF Traction control off*.
2. For data `10 00 00 00 00 00 00 00`, the sixth-bit set, the message changed to *Interior scan To deactivate press OK* with *OK* and *Cancel* also displayed.
3. All subsequent bit settings were as for 2., including resetting the data values as for 1.
4. Turning the power to the display ECU off, waiting two minutes, and turning the power back on cleared the message in 2.
5. When the display ECU was reset via 4. then the first values worked again, as in 1.

The hypothesis for the above results is that the ECU display requires the message to be acknowledged by a button press. However, this cannot be tested because the bench testing does not have the vehicles buttons connected.

It was further discovered that almost all the bits for CAN packet 753 display the same message, the message *OFF Traction control off*, except for:

- Byte 1 bit 6 (value 16) displays *Interior scan To deactivate press OK* with *OK* and *Cancel*;
- Byte 2 bit 7 (value 64) displays nothing;
- Byte 2 bit 8 (value 128) displays *OFF Electronic stability control off*;
- Byte 4 bit 6 (value 32) displays *Hold to switch ESC off*

Thus, the behaviour of packet 753 is different to packets 752 and 793. It is not possible to determine why this is the case. However, it can be speculated that the code processing 753 must be structured differently to code processing 752 and 793.

9.14 Reverse engineering confidential functionality

On first appearances, the display ECU appears to be a simple device, a screen for communicating messages to the vehicle occupants. However, there is a degree of complexity:

- the vehicle's user manual shows 78 possible messages, many of them would only be seen if a fault occurs or certain vehicle options are fitted (e.g. a manual transmission compared to an automatic transmission);
- the transmission rate of CAN packets influences the message display;
- different CAN packets are responsible for its operation;
- different bit settings, singular or in combination, invoke different messages;
- the vehicles buttons affect the operation of the display.

This information may not normally be available to those outside of a vehicle manufacturer's engineering functions and maintenance network. However, in Section 9.7 a list of entities who would find knowledge on the operation of an ECU useful was provided. The experiments against the display ECU have shown that the prototype CAN fuzzer is useful for revealing functionality, more than that which can be determined from operating the vehicle alone. However, the main aim of this research was not to fully reverse engineer a component. The primary aim was to determine the usefulness of fuzz testing to improve automotive systems assurance levels. To that end, the results from testing against various ECUs have provided useful evidence to support the use of automotive fuzz testing. Further support is added by using such results, and for the display ECU, it is to inject unexpected messages into the lab vehicle.

9.15 Injecting display ECU messages

One of the results from the experiments in this chapter is learning enough about a component in order to use it to compromise the vehicle. In doing so it illustrates the need to improve component and system design to make such compromising difficult.

For the display ECU, the knowledge to control what is shown to vehicle occupants is useful for any potential adversary. In this case, the knowledge allows for the display of incorrect messages to the vehicle users, including messages that may not be understood because they are not in the user manual. Displaying those messages would be disturbing in a variety of situations.

In Figure 9.20 the message injection can be seen in action. It should also be noted that the messages cause a loud beeping sound within the vehicle which is not heard during the bench based experiments, and would heighten anxiety for vehicle users.

Fig. 9.20 CAN packets discovered from the fuzz testing are injected into the lab vehicle's internal network, note that the lab vehicle required the brake to be pressed to start, the bonnet was closed, the brake fluid level was not low, and the car had not been used as was therefore cold

It had already been determined that the display ECU is not connected to the OBD port, see Section 9.4.2. Thus, the CAN packets need to be sent via a connection to the lab vehicles internal CAN bus. This is achieved using the same man-in-the-middle connection as for the display ECU investigated in Chapter 8, see Figure 8.3.

This means that the message injection is not as straightforward as it could be. However, access to the network is not difficult and certainly possible by someone knowledgeable. There is the argument that if physical access is required then some other attack would be performed. However, the principles behind the discovery of the display ECU functionality and the message injection, are applicable to future security testing. Furthermore, there are multi-stage scenarios in which a similar, or the same, type of attack would be useful to certain adversaries, for example, state actors or unscrupulous repair businesses planting controllable devices in a vehicle, in order to facilitate a vehicle stop or false repair. Although an attack is possible it may not necessarily be used, however, there are always lessons to learn, and knowledge that can be applied elsewhere.

9.16 Concluding the display ECU fuzz testing

This chapter began with an overview of the display ECU engineering and how it fits within the vehicle's systems. It was chosen as a ToE because of its cyber-physical characteristics, in other words it has a visual aspect that is enabled via the CAN communications. Unless access to the internal designs of vehicles is provided, it may not be immediately possible to determine if a CAN packet will result in a system reaction. This can make fuzz testing vehicle systems difficult in comparison to normal IT systems, where calling an API or sending a communications data packet often results in a response. This is one of the challenges automotive security researchers need to address (see Section 10.4.4).

The first stage in fuzz testing the display ECU was to get it communicating and working with the prototype CAN fuzzer. The work with the gateway ECU in Chapter 6 and the media ECU in Chapter 7 provided valuable knowledge in achieving a reliable bench experiment. However, the vehicle's wiring diagram did not fully match the display's design. This showed that documentation is not always accurate and can hamper experimentation.

Fuzz testing provided immediate results. The combinatorial explosion problem of the CAN bus protocol was not encountered in this instance, due to ECUs operational design. The internal operation of the display ECU's software comes across as fairly straightforward. It would not be difficult to strengthen the resilience of the ECU with some additional checks on the messages that the ECU needs to process (also see Section 10.4.2).

The use of a simple file halving search technique to discover functionality proved useful in finding the CAN ids used to control the display. A further granularity of functionality was achieved by setting individual bits within the discovered packets. Varying the length of the data packets also revealed potential issues with the ECU's software which suggests that only functional testing was performed.

If fuzz testing had been available when the display ECU was developed and was performed prior to production, then beneficial system design changes and bug fixes would have been made. The fuzz testing is also applicable to the post-production vehicle life cycle, where security testing can be applied to design iterations.

Finally, the techniques revealed by the experiments performed in this chapter have helped to improve the functionality of the prototype CAN fuzzer and suggest additional functionality. The results on the display ECU provide useful knowledge for automotive CAN fuzz testing and demonstrating that it is a useful technique for automotive systems testing.

Chapter 10

Discussion and conclusion

We hope you enjoyed the ride!

Johnnycab, Total Recall (Film, 1990)

In this chapter, the knowledge uncovered is summarised. This includes how the contributed knowledge addresses the original objective and research question. There is a discussion on the challenges revealed when fuzz testing within the automotive field. Taking this research further to solve discovered problems is covered, along with other future directions for the work, which includes improvements to the prototype CAN fuzzer.

10.1 On determining the research aim

In the introduction, in Chapter 1, the point was made that the two technologies of software and wireless connectivity provides a conduit for cyber attacks. Researchers have proven that CAVs can be susceptible to hacking. The hacking can be as easy as relaying key fob signals in order to steal a vehicle, or, require many man-years of work to find multi-stage attacks that can be used to take over a car remotely via the Internet or cellular networks.

The point was also made that cyber threats are not a new phenomenon, and much of the security knowledge learnt from other domains can be applied to vehicular systems. Indeed, in the literature review in Chapter 2, long established concepts of software assurance, the CIA triad, threats, threat agents, and ToEs are the foundations for viewing the cyber-security issues faced by CAVs.

Building upon these security foundations are the specific threats against technologies relevant to the automotive field. The history and background to the attacks on vehicle systems were covered, particularly the problems of the ubiquitous CAN bus as the favoured in-vehicle communications network.

Manufacturers must now be fully aware of the possible threats to their products. The vehicle is no longer just about its physical engineering, it is a malleable and connected software platform. As such it brings new dimensions to the engineering operations of the vehicle manufacturers and

their suppliers. Connectivity increases the potential exposure to malicious agents, and therefore, system designs need to be made cyber resilient. This requirement for cyber resilience, along with the publishing of security guidelines and standards by various organisations, means cyber-security testing of vehicles should be considered, if not a requirement, in order for consumers to maintain confidence in new vehicle models.

It is the security testing of connected vehicles which was the high-level objective at the beginning of this research. In Section 2.10 vulnerability, fuzz, and penetration testing were provided as three classes of testing considered suitable for improving cyber-security resilience in vehicles. From these three classes of security testing, fuzz testing was identified as having little meaningful evidence in the literature on its application to the CAN bus, or the automotive field in general. The published work on fuzz testing the CAN bus has been more concerned with the experience of using commercial fuzzers (see Table 2.6) for use in automotive testing, rather than publishing the results that directly examine the use of fuzz testing to tackle CIA weaknesses in automotive systems. It was the lack of CAN fuzz testing knowledge that provided the focused research question, namely to see how fuzz testing can contribute to vehicle cyber-security assurance.

Whereas the standards and guidelines say that automotive fuzz testing *should* be performed, this research adds the evidence as to *why* it should be performed, and for CAN, the *how*. This applied research adds to the knowledge by demonstrating what CAN fuzz testing means in the automotive lab and workshop. This research is useful to several stakeholders:

- Academic vehicle cyber-security researchers will find new information to which they can refer.
- Academic researchers entering the field will find the knowledge useful in guiding the direction of their work. The practical knowledge from the experiments will be useful.
- Automotive firms which offer vehicle security consultancy can incorporate this knowledge from this academic research into their testing procedures and processes.
- This research within in the area of vehicle systems security is ongoing within Coventry University and HORIBA MIRA Limited. In addition, HORIBA MIRA offers commercial testing services, which now includes fuzz testing. The outputs of this research are integrated into HORIBA MIRA's internal knowledge base and processes for access by relevant staff. The prototype software will be used for current and future engineering and research at both organisations.

10.2 Experimental outputs from this research

The outputs begin with the literature review, before following the DSRM process (Chapter 3) to perform practical experiments and software development, building the knowledge required to answer the research question. The research program experimental outputs are:

1. The establishment of an automotive cyber-security testbed in Chapter 4. It was validated by performing an attack via an aftermarket Bluetooth enabled device, known as an *OBD dongle*.

2. The construction of a CAN fuzzer in Chapter 5, which used the testbed for its development and validation.

3. In Chapter 6 the prototype fuzzer was used against several targets. Using the CAN fuzzer against a lab vehicle and its components presents the possibility of component damage. Therefore, a bench-based CAN network was constructed to continue the CAN fuzz testing research, demonstrating the ability to discover data packets in a CAN controlled cyber-physical system.

4. Testing the prototype fuzzer against a vehicle gateway ECU in Chapter 7. The experiment provided another ToE, its design revealed additional functional issues in using the fuzzer, contributing to improvements in its design to support multiple CAN interfaces.

5. Using the prototype fuzzer against a media ECU in Chapter 8. Again, experimentation against another ToE reinforced the knowledge being learnt and confirmed the problems with testing an ECU outside of a full vehicle system.

6. The fuzz testing of a display ECU, which has a cyber-physical aspect, in Chapter 9 revealed confidential operational information, software problems and integrity design issues.

7. An additional experiment, not core to this research was performed. The verification of a bit rate attack, against the CAN bus is recorded in Appendix C. It documents a known engineering issue with CAN that could be deployed as a cyber attack.

The experiments contributed to addressing the research question and raised challenging issues that are discussed in Section 10.4.

10.2.1 Summary of the results from the security testbed experiment

This research began by examining the use of HIL/SIL systems for not only R&D and functional testing, but for use as a cyber-security testbed. The use of HIL/SIL test equipment as a proxy for a vehicle during cyber-security testing not only allows for the development and refinement of new tools and techniques, it also allows for manufacturers to address cyber-security testing earlier in the vehicle development lifecycle, prior to manufacturing. This will require automotive engineers to design security in from the beginning, meeting the principle of secure-by-design, and allows for more time to find any potential security issues.

A further advantage of an automotive cyber-security testbed is the alleviation of the costs involved with automotive security research. Reducing the need to access vehicles, components, garages and proving grounds.

The main output from the establishment of the testbed was the conclusion that HIL/SIL equipment, that is normally used for functional design and testing, can be used for non-functional security testing. A view that other researchers have now taken [64]. A further output is the testbed itself, a useful platform for automotive security testing that can be augmented with additional technologies as research progresses.

10.2.2 Summary of the results from testing a vehicle gateway

In applying the prototype fuzzer against a vehicle gateway ECU, the main output was how ECU design can impact upon testing. Firstly, it interesting to see how the hardware design from a later model vehicle, compared to the older model lab car, afforded improved protection of the CAN bus. It is not known if this was by intention or as a by-product of using the low power features of the CAN transceiver chip used in the gateway ECU.

Secondly, the number of CAN busses in the gateway ECU adds an interesting dimension to the fuzz testing. Ideally, the prototype fuzzer would be able to inject a packet into all four CAN interfaces and monitor all four busses for responses. Whilst the CAN fuzzer was enhanced to handle multiple busses, cross-network data analysis would be a beneficial enhancement to the fuzzer.

Finally, the symbiotic operation of the gateway ECU adds an interesting dimension to fuzz testing components. It has been seen with other components, for example, the instrument cluster, the media ECU and the display ECU, that vehicle components are designed to operate as part of a whole system. Removing them from that system and attempting to test them on their own provides additional considerations to the testing environment.

10.2.3 Summary of the results from testing a media ECU

The media ECU exhibited similar issues as the gateway ECU. Again it had more than one CAN interface (two), and, therefore, would benefit for simultaneous monitoring of multiple ports. It was an easier device to work with in terms of being able to communicate with it on the bench. However, it also needs to be part of the system in which it is designed to make progress with the fuzz testing.

The output taken from using the media ECU was that fuzz testing via CAN will require the tooling to support multiple boundaries in the configuration of the CAN testing parameters. The initial parameters calculated from monitoring the lab vehicle's CAN busses still had too large a spread of CAN packet ids. Therefore, a useful addition to the prototype CAN fuzzer would be the capability of entering multiple ranges of fuzzing variables, to reduce the spread of randomised data that needs to be generated.

Another finding from testing the media ECU is the need for the ability to dynamically narrow in on a CAN packet causing an ECU reaction. That additional functionality is to be added to the fuzzer in future work. A process that was partially automated in the display ECU testing.

10.2.4 Summary of the results from testing the display ECU

The fuzzer was used against a display ECU and it revealed problems with the ECU's software (buffer overflow, undocumented messages, repeated messages, and incorrect messages), confidential operational information (how to display messages to vehicle occupants), and system integrity issues (how to inject incorrect packets into the system). If the fuzz testing was performed prior to production, then design changes could have been made to improve vehicle assurance levels:

- correcting the software issues;
- adding extra checks in the software, an example of a packet length check as in Chapter 6, which increased the time to find a CAN packet;
- securing access to the wiring beneath the dashboard.

As for the media ECU, functionality within the fuzzer to dynamically find a CAN packet causing an ECU action would be beneficial. It was partially achieved by adding the ability to playback specific lines from a CAN log file, which was supplemented by a manual search process with the single shot message transmission functionality.

Another useful addition to the fuzzer was the ability to configure the exclusion of packets with certain ids. It enabled easier reverse engineering of functionality. Further work will investigate if it would be worth extending this functionality to include values within the packet payload.

Another observation to make is that technical information is not always accurate. Although the display ECU wiring diagram indicated two CAN busses, there was only one, and the indicated wiring for the second bus was not present in the vehicle.

10.3 Contributions from the research outputs

Having summarised the experimental outputs, the contributions to knowledge are discussed, derived from the literature review, experimental outputs, and the methods and tooling development from this research.

10.3.1 Contribution from the literature review

The contributions begin with the literature review itself, which had identified the gap in the knowledge in relation to fuzz testing automotive systems, particularly the CAN bus, finding that fuzz testing has not been rigorously studied in the automotive field. Not only did the literature review show how existing cyber-security terms are applicable to vehicle systems, but it also provides a useful summary of the history and context of the automotive cyber-security field, supplemented with a brief history of the computerised vehicle in Appendix A.

10.3.2 Contribution from the development of the fuzzer tool

The contribution of a specialised prototype CAN fuzzer is of benefit to researchers and security testers performing automotive fuzz testing research. Chapter 5 discussed the rationale for a new CAN fuzzer tool and outlined its desired aspects. The resultant prototype fuzzer has benefits over the existing commercial and open source tools.

- The prototype CAN fuzzer meets the need for a dedicated CAN fuzzer tool for the automotive field. It enables automotive systems testers and security researchers to concentrate on the

process of fuzz testing, removing the need to develop knowledge of a complex general-purpose fuzzer tool.

- The minimal learning required for using the CAN fuzzer extends to its deployment. It is simple to deploy with the binaries being copied to an empty directory on a Windows machine, and the drivers for the PCAN USB interfaces installed. Unlike some of the commercial and open source fuzzers, there is no need to use a dedicated Linux computer and install a complex software stack.
- Set-up of the prototype CAN fuzzer is via a simple GUI interface, unlike some of the general-purpose fuzzers, which can require in-depth editing of a variety of text based configuration files.
- The use of a single modern high level language (C#) and modular software design (see Chapter 5) means it is adaptable in the future to other CAN hardware interfaces and is usable for implementing new testing algorithms. Commercial fuzzers are closed source and difficult to extend to new automotive technologies. Some open source fuzzers are based upon scripting languages and are not performant enough for commercial use.

The engineers at the commercial partner for the research program, HORIBA MIRA Limited (see Acknowledgements), are implementing the prototype CAN fuzzer due to its advantages for automotive use over the existing commercial and open source fuzzers.

10.3.3 Contribution from fuzz testing the CAN bus

When the prototype fuzzer was deployed against a vehicle (Chapter 6), the immediate impact upon the vehicle systems was surprising. Even though the number of possible CAN packet combinations are inordinately high, the effect of fuzz testing CAN, in terms of a DoS effect, is immediately noticeable. Further detailed research is required into mitigating the effects of fuzz testing.

The main contribution from the experiments is to demonstrate that fuzz testing does have a useful role to play in the security testing of automotive CAN systems. It provided evidence that vehicle subsystems require segmentation, isolation and protection from outside interference (see Section 10.4.2 later). The ECUs in this research were found not to be resilient to fuzz testing. If ECUs are made resilient to fuzz testing, e.g. through improved software checks on CAN packet data contents, further research is required to address potential issues around combinatorial explosion (see Section 10.4.5).

10.3.4 Contribution from identifying new automotive testing challenges

The several obstacles that presented themselves during the research enabled observations on challenges in the automotive security testing field. These challenges are discussed in 10.4, below. The contribution

of identifying these challenges allows future researchers to use them as an opportunity to inform their research.

10.3.5 Contribution of a method for developing automotive cyber-security tests

The process of researching CAN fuzz testing, developing and improving the prototype CAN fuzzer tooling, and the experimental use of the new fuzzer, resulted in a new methodology that can be applied to develop future automotive cyber-security test methods and tooling. The seven-step automotive security test development methodology is discussed in detail in Section 3.5 in Chapter 3.

10.3.6 Contribution in identifying combinatorial explosion in CAN fuzz testing

The problem of a combinatorial explosion is well established in the computer science and testing fields. However, it has not been addressed to any depth within the automotive security testing literature. In Section 3.6 in Chapter 3 a method is proposed that would limit the state-space to be searched during CAN fuzz testing and its use is discussed in Section 10.4.5. However, a combinatorial explosion has been identified as an issue with automotive fuzz testing and further research work to investigate the issue will be performed.

10.3.7 Contribution from the bit rate attack experiment

Whilst not part of the main research, the CAN bit rate attack was as a result of operational observations during the development of the practical experiments. It was interesting to discover that what many considered to be a mere configuration error has the potential to be weaponised. The contribution from the bit rate experiment is to show that the bit rate attack is a possibility, whilst acknowledging that further research is required to fully deploy, and counter, the attack.

10.4 Discovered challenges in automotive security testing

Having summarised the main contributions from this research the lessons learnt can be gathered. This section lists the issues identified during this research, and the possible solutions are outlined.

Whilst CAN technology is straightforward, applying CAN fuzz testing to vehicles and components is challenging. However, the experimental results did confirm previously made researchers statements [2], [21], [38], namely:

- Fuzz testing can be used to reverse engineer vehicle packets.
- Disruption of a vehicle's communication network is not difficult.
- The fuzz test can be used as a form of cyber attack.
- Cyber-security testing vehicles and their ECUs can lead to vehicle component damage.

10.4.1 Challenge 1: The risk of damage versus obtaining trustworthy results

The work outlined in Chapter 6, when the prototype fuzzer was tried against physical systems and components, reinforced the fact that damage to the ToE is a possibility, however slight. To mitigate this issue, and avoid the potential expense of repairing or replacing components, the systems and components should reject invalid sequences of inappropriate data (see the next challenge). In this research the potential for damage caused a restriction on the use of the prototype fuzzer against the vehicle, instead, targeting bench-based ToEs. However, rejecting invalid sequences of inappropriate data still does not guarantee the fuzzer will not generate a CAN data packet, or sequence of data packets, that will not damage the internal state of the ECUs. However, the fuzz testing must provide meaningful results, and means it must be used against valid ToEs, including full vehicle systems.

10.4.2 Challenge 2: Design of suitable protection mechanisms

Since fuzz testing can have a detrimental effect on a running vehicle, it follows that to be safe from attackers, vehicle systems connected to a CAN bus should have additional validation checks to enable them to ignore nonsensical CAN data packet values. These could be at the level of the CAN bus or part of the internal logic of the ECUs.

Typical protection mechanisms include network segmentation, packet encryption, firewalls and intrusion detection. Although functional segmentation of vehicle networks is common (see Figure A.3 in Appendix A), exploits [23]) have shown that cross-network protection is not always achieved. This is in part due to the need for data to flow across network boundaries, for example updating vehicle displays to inform the driver of system statuses (e.g. what gear the vehicle is in).

Proposals for CAN packet encryption have been made (e.g. [33]), however, the bandwidth restrictions and timing requirements imposed by the CAN protocol mean that practical schemes have not been produced. Vehicle firewalls have long been proposed [37], and commercial companies exist that sell firewalls and intrusion detection devices[1]. It is not known whether such protection devices are to be deployed in forthcoming vehicle models, or even if they are effective. However, their protection properties will need to be assessed, and security testing, including fuzz testing, of such products will be required. Yet, complex solutions are not the only protection method, our research suggests that simple protection mechanisms may be effective:

- The use of the fuzzer against the bench-based network in Chapter 6 showed that a simple modification to the packet increased the search time required for the fuzzer to find a packet. This suggests that increasing the complexity of the packet will require attackers to take more time designing their attacks.

- The gateway investigated had a hardware protection mechanism that prevented its examination outside of the vehicle. The grounded CAN lines and packet filtering hindered the use of the

[1] https://tekeye.uk/automotive/cyber-security/automotive-cyber-security-companies

gateway in the development of the fuzzer, but those features would also hold back anyone trying to learn more about the internal operation of the vehicle.

10.4.3 Challenge 3: Vehicle components function as part of a CPS

The point to make here is that the examination of the gateway ECU, media ECU and display ECU in Chapters 7 to 9 indicates that components, which are designed to be part of a network, would not function correctly as a stand-alone device. Further work here could consider some form of network emulation within the fuzzer. CAN traffic playback functionality from a log file, as discussed in Section 7.6 in Chapter 7, is one way of emulating a vehicle's CAN traffic and has been partially implemented (see Figure 5.8b in Chapter 5).

10.4.4 Challenge 4: Observing CPSs

Correct or incorrect responses to CAN fuzz testing are rarely possible through an examination of the network data. The CAN bus operates within a physical thing (the vehicle), and the incorrect response to a packet may be a physical reaction from a component.

A tool such as OpenCV[2] (which contains video processing software for real-time monitoring) would help in capturing incorrect physical responses from the vehicle. Tackling the observance and operation of a CPS is a challenge, and will require a variety of skills, covering both hardware and software.

10.4.5 Challenge 5: State-space explosion

Combinatorial explosion is not uncommon when dealing with computational systems. The lab vehicle uses a standard CAN packet with an eight-byte payload. Even though this is a simple protocol it produces an inordinately large number of possible CAN packet combinations. Visual examination of source code, formal methods and functional testing can provide a high degree of assurance for ECU programs, but they cannot allow for bugs caused by human errors, design errors, or the tools that compile and assemble the code. Could an unknown CAN packet in the large state space cause an issue with the ECU code?

The method outlined in Section 3.6 in Chapter 3 was applied in Chapter 8 to the media ECU. However, the CAN fuzzer requires further refinement around the state space issue. Testing the media ECU identified that multiple CAN packet ranges should be definable when fuzz testing.

In Chapter 9 the state space was restricted by retaining the full eight-byte payload length. The discovery of a probable buffer overflow issue suggests that restriction was not required as the ECU incorrectly responded to packets of different lengths. What the designers of ECU software should do is use the large state space to their advantage. As in Chapter 6, using a length check increased the time to find a CAN packet by fuzz testing. Simple checks on invalid data combinations would

[2]https://opencv.org/

have increased the time dramatically. Instead, the fuzz testing was effective due to the lack of data checking mechanisms.

The functionality for finer-grained configuration within the prototype fuzzer will enhance its usefulness. The refinement is one of the improvements that will be made to the fuzzer in future work. Furthermore, research is needed to explore the benefits of running more targeted fuzz tests, particularly with regards to finding unconsidered code paths in ECUs. The state-space problem remains an issue in automotive fuzz testing.

10.4.6 Challenge 6: Granularity of control

Currently, the fuzzer operates by altering byte values in the id and data fields of a CAN packet. This is at the same software level as the ECU code. Reducing the fuzz testing to the protocol bit level would allow an investigation into how systems respond to corrupt packets, such as an incorrect cyclic redundancy check in the CRC field. However, this additional control would need to be set against the previous challenge, the state space explosion. It would also require specialised interfaces capable of manipulating the CAN protocol at the bit level.

10.4.7 Challenge 7: Other vehicle networks and technology

CAN is only one of the possible data networks in a vehicle (see Table 2.4 in Chapter 2), and fuzz testing should be applied across them all. The first to consider will be CAN-FD, the Flexible Data-rate (FD) version of CAN. This is derived from CAN, but has a larger range of id values and allows for a larger data field (up to 64 bytes). It is anticipated that it will be relatively straightforward to support CAN-FD in the prototype fuzzer.

The technology does not stop at data networks. There is a variety of digital technologies within vehicles providing a large attack surface. Newer and more complex technologies are coming with autonomous driving. There is likely to be a need for a range of different fuzz testing tools and methods which are dependent upon the system are under test. For example, the tools to perform fuzz testing against Bluetooth will be different from the tools used against ultrasonic parking sensors.

10.5 Summary of future research

In the previous sections recommendations for further research were made, summarised here:

1. **Expansion of the security testbed:** Adding additional vehicle systems technologies to the fuzzer. The lists of possible technology additions are shown in Tables 2.4 and 2.5. Adding support for extended CAN (29-bit ids) and CAN FD will be considered first.

2. **Addressing combinatorial explosion:** In common with many computational systems, methods to handle the search for possible software issues within the large state spaces of vehicle systems need investigation. Input from the general software testing research community would be

beneficial. To help address this problem for CAN based systems the fuzzer will be modified to enable finer control of the range of generated packet data.

3. **Symbiotic testing:** Vehicle ECUs are reliant upon the systems in which they operate. Bench testing, whilst possible, will be more effective if the network traffic and ECU I/O fully represented the whole system interactions. This is where the use of virtual systems on the testbed could be beneficial. Additions to the fuzzer to include network and ECU emulation can be considered. Furthermore, monitoring of the physical world with sensor and vision systems will be required for future testing requirements. Partial symbiotic testing may be possible with the implemented CAN log file playback facility, and this will be tested in future work.

4. **Protocol fuzz testing:** This research tested CAN at the field level. Fuzzing at the CAN protocol bit level is an area for future work.

5. **Dynamic packet searching:** The ability to automate the finding of a CAN packet that evokes an ECU operation requires further work. It is currently performed in a semi-automated manner with log file playback and single shot packet transmission. Such functionality needs to consider the monitoring of physical actions, see point 3. The use of packet id exclusion lists aids the search for functional ECU CAN packets, consideration of extending the exclusion functionality to include data values is required.

In addition, this research can be extended into other areas:

1. Fuzz testing is not only new to the automotive field, the science of fuzz testing, itself, requires further knowledge. The immaturity of the tools in terms of systems monitoring, analysis of results, and obtaining useful metrics provides research opportunities.

2. Whilst the developed software will be used and extended for further studies in fuzz testing vehicle systems, it could also be adapted for fuzz testing engineering tools themselves. For example, HIL/SIL interfaces and CAN interfacing devices. This is to ensure assurance in the tooling that is used to design and test vehicle systems.

10.6 On answering the research question

The literature review, Chapter 2, covered the concept of software assurance. Violating the CIA triad reduces software assurance, and ultimately customers confidence in products that rely on software for functionality. For CPSs, such as vehicles, lower assurance reduces the physical safety of people and property. In Section 1.2.5 the question was posed as to whether the results from fuzz testing, which is used successfully in traditional IT systems testing, can contribute to vehicle system security assurance. To do this it needed to be demonstrated that fuzz testing had an effect on vehicle systems with regard to the CIA triad.

The developed CAN fuzzer was used against a variety of targets. It did reveal system weaknesses, the experiments demonstrated that the CIA security properties of a CAN based system can be compromised. Whilst the fuzzer could not be fully deployed against the lab vehicle, it did reveal problems with component software (see Chapter 9), confidential component operational information (Chapters 6 and 9), vehicle system integrity issues (Chapters 6 and 9), and affected system availability (Chapters 5 and 6).

Straightforward engineering changes would mitigate the security issues (as discussed in Sections 9.16 and 10.4.2). Therefore, CAN fuzz testing does improve a vehicle's security assurance, provided the results are acted upon. Furthermore, it demonstrates that the development of security tests, and appropriate tooling, is a requirement in automotive engineering, and supplements traditional functional testing.

However, a further demonstration of the ability to compromise confidentiality is required. This research was not against a full vehicle system, a preferred scenario to research the confidentiality of the personal data that is increasingly stored within vehicle ECUs.

Fuzz testing has been demonstrated to be viable and practical in identifying security design flaws in the automotive field. However, to move from fuzz testing in its wider sense, i.e. not just CAN but including other automotive technologies, additional research is required. This is to gather more knowledge in applying fuzz testing to those technologies and to invent, enhance and mature the tooling support, methods and experimental techniques.

10.7 Research impact considerations

Section 10.3 discussed the direct contributions from this research program. In this section consideration of the wider impact of the research is covered, by discussing how fuzz testing as an attack can be mitigated, looking at how the research applies to wider stakeholders and considering how manufacturers approach securing the connected car.

10.7.1 Mitigating fuzz testing as an attack

This research verified the usefulness of fuzz testing as an attack or reverse engineering technique for potential threat agents. The use of fuzzing as a DoS attack, or for reverse engineering vehicle functionality, requires consideration of mitigation techniques. Such techniques can include the protection mechanisms discussed in Section 10.4.2, i.e. network segmentation, firewalls, IDPSs, data encryption, physical protection of the CAN bus, and additional checks on the sanity of the data within a CAN packet. However, there are trade-offs, for example, the use of various types of encryption (discussed in Section 2.8 in the literature review in Chapter 2) has not resulted in a commonly accepted scheme. The uptake of CAN protection methods is limited due to various non-technical factors [33] that make them unsuitable for production deployment, which can be summarised as stating that new technology must have a cost-benefit and, importantly, be maintainable over the full lifecycle of a

vehicle by the dealer network. Any encryption system must not prevent a vehicle from being repaired years after it has been produced. The automotive industries need a solution to long term management of encryption keys.

One of the mitigation techniques not yet widely considered is the move away from the standard CAN bus itself. The natural evolution of the technology used within vehicles may see the improved protection required for in-vehicle data networks. There is a recent two-wire point-to-point automotive Ethernet standard, IEEE 802.3bw (also called 100BASE-T1). Using automotive Ethernet as a high speed link between ECUs may eventually see the end of the CAN bus within vehicle systems. However, that is some time in the future, in the short term, manufacturers should, where possible consider isolating certain ECUs from those that are connected to external communications networks to mitigate risk.

10.7.2 Impact of the research on additional stakeholders

In Section 10.1, at the beginning of this chapter, some of the stakeholders that would benefit from the knowledge from this research were listed. In addition, consideration is given to what manufacturing organisations gain from the research, and the final users of a vehicle, i.e. the car owners and their passengers.

Vehicular cyber-security testing is still in its infancy, even compared to the recent car hacking research field, which started in the early 2000s. This research has contributed new knowledge to demonstrate the value of security testing, particularly fuzz testing. It provides some new evidence for deploying security testing earlier in the design cycle for automotive systems. Early security testing can help manufacturers uncover potential issues, including any related safety issues that can occur as a result of a cyber attack. This increases the confidence in the resilience of the vehicle systems before the vehicles reach the customers. Fixing issues post vehicle delivery not only has a financial cost but can also impact upon an organisation's reputation.

The use of security testing to maintain system resilience will help retain the confidence of customers in the cars that they drive and use. Cyber-security issues are reported with regularity in the media, and some vehicle users will be aware of the threats to computerised systems. When they ask questions about cyber-attacks, the ability of manufacturers to communicate to customers about potential threats, and how they test for resilience to resist and prevent them, helps with the confidence argument. In Chapter 2, Section 2.3 the concept of software assurance was stated and how assurance promotes confidence, and it is the role of security testing to aid software assurance.

10.7.3 Securing the connected car

Testing for resilience aids with the discovery of software issues and unconsidered use cases of systems. However, designing in security mechanisms to help secure the connected car must be considered. Whilst data encryption of the CAN bus has not been practically implemented, the use of encryption at the vehicle's boundary is already present, built into the Bluetooth and WiFi protocols. However, this

encryption needs to be implemented correctly, and, as seen in Chapter 4, poor PINs, or passwords, are not helpful to security. The lessons learnt from the many years of securing traditional IT systems must be applied. This includes not only strong passwords but regular software updates as issues become known. These updates will be over the vehicle's connected channels, or in the absence of wireless connections, via dealer or user administered updates. Furthermore, vehicle users must be informed of good security practices, ideally, when a vehicle is handed over, or via online resources or the user manual. Users are a potential security weak spot in any system.

10.8 Conclusion

Automotive cyber-security is a relatively new cross-disciplinary field of study. It requires the deployment of a wide variety of security skill sets and tooling to deal with the complexity and range of technologies in use, particularly since a CPS differs from traditional information systems. Developing those skills and the applicable tooling is challenging for researchers and security engineers. It is easier to achieve for those working within the automotive industry who have access to confidential design information, compared to those working in a purely academic capacity. As discussed in the literature review (Section 2.14.1) researchers have barriers to overcome. The cost of components, vehicles and facilities is restrictive, and the issue of commercial confidentiality often means that there is no detailed information available on how systems under test operate internally.

To aid automotive security research a method has been implemented that is suitable for developing testbeds, tooling and security tests (Section 3.5). The method has been applied to automotive fuzz testing, starting with the insecure CAN, which is present within all mass-manufactured vehicles. However, the method not only applies to fuzz testing. It is designed to apply to the two other types of security testing, penetration and vulnerability (see Section 2.10).

The SAE J3061 guidelines and forthcoming ISO/SAE 21434 standard state that fuzz testing is desirable for vehicle systems engineering. To achieve widespread use of fuzz testing there is a need for on-going research. This is to cover all the digital technologies in a vehicle and to fully develop a range of tools and techniques.

Applying the method from this research to fuzz testing has resulted in an easy to use and easy to deploy prototype CAN fuzzer. The fuzzer has been used to increase the knowledge in automotive CAN fuzz testing. However, there is still more research work to be done on fuzz testing techniques. The fuzz testing of CAN is only one piece of the connected car's complex security engineering process. The contributions here only address part of the picture within the automotive security field, but they provide a base for making fuzz testing a common automotive security test.

The results from this research will be useful to anyone interested in how software assurance applies to the automotive field, and how the security properties of systems can be tested to improve assurance levels. There are challenges that need to be addressed, however, meeting those challenges will be useful to design, test and security engineers, not only in the automotive field but also other industries that use CPS and CAN based technologies.

Research on vehicle security assurance will not end since cyber-security is not a solvable problem. There is a need for an automotive security testing knowledge base to be continually built upon. The security test development method from this research will be used to add to that knowledge by developing other security tests and tools. This on-going research is aimed at expanding testing methods within the automotive cyber-security field.

References

[1] W. H. Ware, "Security and Privacy in Computer Systems," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, Atlantic City, New Jersey: ACM, 1967, pp. 279–282. DOI: 10.1145/1465482.1465523.

Covers security concepts related to computer systems, showing that computer system security is a long standing issue. Subsequent cyber-security issues were foreseen as possibilities in this report.

[2] 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*, 2010, pp. 447–462.

The authors practically demonstrated previously theoretical possibilities on controlling a vehicle via a cyber attack. Numerous attacks on a running car were performed. Security implications were discussed. This research was reported widely in the media and probably accelerated interest in the automotive cyber-security field.

[3] M. Meng and W. Khoo, "An Analysis of Secure Software Development Lifecycle from an Automotive Development Perspective," SAE, Warrendale, Tech. Rep., 2016, p. 4. DOI: 10.4271/2016-01-0040.

A brief look at the importance of the SDLC process for automotive systems. Provides an overview of the vehicle development process, the V-model, SDLC and applicability to component development.

[4] ISO, *ISO 26262-2:2018 Road vehicles - Functional safety - Part 2: Management of functional safety*, Geneva, 2018.

ISO 26262 is the international standard used by vehicle manufacturers and their supplies to analyse and reduce risks in the functional operation of cars and their components. The latest 2018 version acknowledges the intersection with cyber-security.

[5] H. Schuette and M. Ploeger, *Hardware-in-the-Loop Testing of Engine Control Units - A Technical Survey*, Warrendale, 2007. DOI: 10.4271/2007-01-0500.

ECU testing has a long tradition of using test rigs for R&D. Manufacturers need to consider the use of test rigs for vehicle cyber security R&D.

[6] D. S. Fowler, M. Cheah, S. A. Shaikh, and J. Bryans, "Towards a Testbed for Automotive Cybersecurity," in *Software Testing, Verification, and Validation, ICST, International Conference on*, Tokyo: IEEE Computer Society, 2017, pp. 540–541, ISBN: 9781509060313. DOI: 10.1109/ICST.2017.62.

The use of a HIL/SIL testbed for cyber-security testing vehicular systems was presented at the 2017 IEEE International Conference on Software Testing, Verification and Validation.

[7] A. Zeller, "Search-Based Testing and System Testing: A Marriage in Heaven," in *2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST)*, 2017, pp. 49–50. DOI: 10.1109/SBST.2017.3.

This paper summarises a keynote that was given at the 2017 IEEE International Conference on Software Testing, Verification and Validation. The keynote of Professor A. Zeller provided impetus to investigating dynamic testing as a vehicle system test method. He provides the argument that search based inputs at the system level can complement, and improve, the results from static analysis, unit testing and integration testing.

[8] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox Fuzzing for Security Testing," *Queue*, vol. 10, no. 1, 20:20–20:27, 2012, ISSN: 1542-7730. DOI: 10.1145/2090147.2094081.

Microsoft have successfully used fuzzing to test their Windows OS for many years.

[9] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

DRSM is a method that can be used for systems engineering where the details of a process, system, program, design, etc. (an artefact) is not fully known from the outset. It is good for covering the various aspects of the design process that need to be considered, from the initial problem to the final communication of results.

[10] K. M. Goertzel, T. Winograd, H. L. McKinley, L. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software Security Assurance State-of-the-Art Report," DTIC, Fort Belvoir, Tech. Rep., 2007, p. 396. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a472363.pdf.

A report by the US military, covers security risks, engineering and formal methods, testing, initiatives, standards, resources and observations. Provides a useful definition for the meaning of software assurance.

[11] NIST, "FIPS PUB 199: Standards for Security Categorization of Federal Information and Information Systems," *Fips*, vol. 199, no. February 2004, p. 13, 2004.

Provides a definition of the CIA security properties triad and how those properties can be classified in term of the impact (low, moderate, and high) upon operations, assets and individuals.

[12] US Govt, *E-Government Act of 2002*, 2002. [Online]. Available: https://www.gpo.gov/fdsys/pkg/STATUTE-116/pdf/STATUTE-116-Pg2899.pdf.

This act required the US Government to implement Internet based services to improve efficiency and access to government services. It included coverage of security and a definition for the CIA triad.

[13] European Communities, "Information Technology Security Evaluation Criteria (ITSEC) Provisional Harmonised Criteria," Luxembourg, Tech. Rep., 1991, p. 171.

ITSEC was an early definition of a certification system for IT systems security and assurance. It provides a definition of the CIA triad. It subsequently contributed to the ISO/IEC 15408 international security standard, a.k.a. Common Criteria.

[14] K. Lemke, C. Paar, and M. Wolf, Eds., *Embedded Security in Cars*. Berlin, Heidelberg: Springer-Verlag, 2006, p. 273, ISBN: 9783540283843.

This is an early book covering security issues in cars. Written by Germans who led the way in early vehicular security research. The same researchers are amongst the first contributors to the long running commercial conference of the same name, commonly known as "escar".

[15] D. K. Nilsson and U. E. Larson, "Simulated attacks on CAN buses: vehicle virus," *Fifth IASTED International Conference on Communication Systems and Networks (AsiaCSN 2008)*, pp. 66–72, 2008.

An early look at how the security properties of a vehicle can be attacked. Defines building blocks for a vehicle attacking "virus" and the need for preventative measures.

[16] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach, A. Fuchs, S. Gürgens, O. Henniger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacalet, and G. Pedroza, "Security requirements for automotive on-board networks based on dark-side scenarios," Tech. Rep. 1.1, 2009, p. 149.

The European E-safety vehicle intrusion protected applications project ran from 2008 to 2011 and investigated protection of in-vehicle networks. This output looked at security engineering, threats and security requirements.

[17] C. Metz, "AAA protocols: authentication, authorization, and accounting for the Internet," *IEEE Internet Computing*, vol. 3, no. 6, pp. 75–79, 1999, ISSN: 10897801. DOI: 10.1109/4236.807015.

The CIA triad defines the properties of computer security. Triple-A security is a protocol used to implement a security mechanism to preserve CIA.

[18] ISO, *ISO/IEC 15408-1:2009(E) Information technology - Security techniques - Evaluation criteria for IT Security*, 2014.

This introduces concepts and terminology for the evaluation of security techniques in systems, software and components. Much of the work is derived from Common Criteria. It defines terms such as Target of Evaluation and threat agent.

[19] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*, 1st. Doubleday, 1989, ISBN: 9780385249461.

One of the earliest writings on tracing a remote cyber attack on computer systems.

[20] M. Wolf, A. Weimerskirch, and C. Paar, "Security in Automotive Bus Systems," *Proceedings of the Workshop on Embedded Security in Cars*, pp. 1–13, 2004.

Provides an early discussion on a vehicle's digital systems and networks, and the security implications, this includes attacks, encryption defences and firewalls.

[21] C. Smith, *The Car Hacker's Handbook : A Guide for the Penetration Tester.* No Starch Press, 2016, ISBN: 9781593277031.

A comprehensive look at techniques for manipulating, breaking and controlling vehicle systems.

[22] 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 *20th USENIX Security Symposium*, San Francisco, 2011, p. 6.

A comprehensive view of how a vehicle can be exploited remotely through software. This paper expanded upon previous work in performing cyber attacks on a vehicle. The focus in particular was to research the possibility of a true cyber attack, i.e. from a remote distance from the vehicle over existing wide area transmission networks (the Internet and cell phone communications). This remote compromise was to address concerns that prior work had unrealistic physical access to vehicles, closing a gap in the knowledge that remote compromise is possible.

[23] C. Valasek and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, vol. 2015, pp. 1–91, 2015. [Online]. Available: https://ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf.

The aim of this paper was to take further the previous work of Kocher (2010) and Checkoway (2011). The authors were able to engineer an attack that could be deployed over a cell phone connection and gain physical control of a vehicle.

[24] isits AG International School of IT Security, *escar Europe conference history*, 2019. [Online]. Available: https://www.escar.info/escar-europe/history.html (visited on 03/27/2019).

The commercial Embedded Security in Cars conference was established in 2003, attracting many researchers and businesses involved in automotive cyber-security.

[25] D. Ward, I. Ibarra, and A. Ruddle, "Threat Analysis and Risk Assessment in Automotive Cyber Security," *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 6, no. 2, 2013, ISSN: 19464614.

Examines how a security risk analysis of computer based systems can be performed similarly to the hazard based analysis used for functional safety, whilst acknowledging and accounting for the differences.

[26] D. Klinedinst and C. King, "On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle," Pittsburgh, 2016, p. 21.

Funded by the US Government this report is a summary of CAN and the OBD port, and their vulnerabilities, evaluated under the STRIDE and EVITA criteria.

[27] SAE International, *J3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*, Warrendale, 2016. [Online]. Available: http://standards.sae.org/wip/j3061/.

Provides detailed considerations for the management of a cyber-security process for a product's development and its lifecycle. It mirrors the functional safety V-model and draws influence from the ISO26262 functional safety standard. Additional cyber-security resources are referenced.

[28] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and Vulnerable: A Story of Telematic Failures," in *Proceedings of the USENIX Workshop On Offensive Technologies (WOOT)*, Washington, D.C.: USENIX, 2015.

A comprehensive security review of a vehicle telematics device that plugs into the OBD port was performed. Several security flaws allowed the researchers to compromise the safety of a vehicle remotely.

[29] S. Woo, H. J. Jo, and D. H. Lee, "A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015, ISSN: 15249050. DOI: 10.1109/TITS.2014.2351612.

Korean researchers reverse engineered CAN packets and programmed a phone app to send them over a Bluetooth dongle connected to a car. The phone could maliciously affect ECU functions and inject CAN messages to impact the vehicle safety. A security protocol was designed to mitigate the attack.

[30] D. K. Oka, T. Furue, L. Langenhop, and T. Nishimura, "Survey of Vehicle IoT Bluetooth Devices," in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, IEEE, Nov. 2014, pp. 260–264. DOI: 10.1109/SOCA.2014.20.

Bluetooth is one of the multiple interfaces to a vehicle. Supplemental devices, e.g. OBD dongles, may also use Bluetooth for communications. Bluetooth does provide reasonable security if used correctly. This survey shows that poor choice in Bluetooth pairing schemes lowers the security threshold for vehicle connectivity.

[31] R. Bejtlich, "Strategy, Not Speed: What Today's Digital Defenders Must Learn From Cybersecurity's Early Thinkers," *Brookings*, no. May, pp. 1–18, 2014. [Online]. Available: http://www.brookings.edu/research/papers/2014/05/07-strategy-not-speed-digital-defenders-early-cybersecurity-thinkers-bejtlich.

Infiltration of systems is a long standing issue. The need to keep abreast of on-going cyber-security problems today means learning from the past. Experience from previous decades can still be applied.

[32] Q. Hu and F. Luo, "Review of Secure Communication Approaches for In-Vehicle Network," *International Journal of Automotive Technology*, vol. 19, no. 5, pp. 879–894, Oct. 2018, ISSN: 1976-3832. DOI: 10.1007/s12239-018-0085-1.

Provides a good summary of the technology used for in-vehicle networks and the approaches that have been investigated for securing communications within the vehicle.

[33] N. Nowdehi, A. Lautenbach, and T. Olovsson, "In-Vehicle CAN Message Authentication: An Evaluation Based on Industrial Criteria," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto: IEEE, 2017, pp. 1–7. DOI: 10.1109/VTCFall.2017.8288327.

CAN's lack of security features means that several schemes have been devised to increase CAN's security resilience. None of these schemes have been adopted. This paper looks at the reasons behind this lack of adoption and concludes that schemes do not take into all required factors.

[34] J. A. Bruton, "Securing CAN Bus Communication: An Analysis of Cryptographic Approaches," PhD thesis, NUI Galway, 2014, p. 205.

Cryptography can be used to added security to a system and many cryptography methods have been proposed to add security to CAN. However, this causes performance issues.

[35] M. Wolf and T. Gendrullis, "Design, Implementation, and Evaluation of a Vehicular Hardware Security Module," in *International Conference on Information Security and Cryptology*, 2011. [Online]. Available: http://www.evita-project.org/Publications/WG11.pdf.

Hardware based cryptography is seen as one solution to overcoming performance issues in adding a security layer to vehicle networks. Here, the design, implementation and evaluation of a hardware security module on an FPGA is provided.

[36] T. Sugashima, D. K. Oka, and C. Vuillaume, *Approaches for Secure and Efficient In-Vehicle Key Management*, Apr. 2016. DOI: 10.4271/2016-01-0070.

Encryption of data for in-vehicle networks has yet to be widely established. One reason is the handling of cryptographics keys. Here, two theorectical in-vehicle proposals to the problem are examined. However, practical research results on key management solutions are still required.

[37] R. J. Chutorash, *Firewall for vehicle communication bus*, 2000. [Online]. Available: https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2000009363.

The possibility of a firewall in vehicles has been considered for some time. Here, the firewall blocks communication to the vehicle network bus if the correct access code is not provided.

[38] C. Miller and C. Valasek, "Adventures in Automotive Networks and Control Units," *Technical White Paper*, p. 99, 2013. [Online]. Available: http://www.ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf.

Provides a commentary on the methods and procedures used to reverse engineer vehicles CAN messages and use the results maliciously. Problems encountered are discussed and ways to detect the attacks are indicated.

[39] National Highway Traffic Safety Administration, "Cybersecurity Best Practices for Modern Vehicles," U. S. Department of Transportation, Washington, DC, Tech. Rep., 2016, p. 22. [Online]. Available: https://www.nhtsa.gov/document/cybersecurity-best-practices-modern-vehicles.

Provides guidelines for vehicle engineering while keeping cybersecurity considerations in mind.

[40] HM Government, "The Key Principles of Cyber Security for Connected and Automated Vehicles," HM Government, Tech. Rep., 2017, pp. 1–17. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/624302/cyber-security-connected-automated-vehicles-key-principles.pdf.

Lays down some over-arching security principles that need to be addressed by parties involved in providing CAVs and ITSs. It covers lifecycle, management and system design. Contains a list of useful standards.

[41] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd. Indianapolis: Wiley Publishing Inc., 2008, p. 1080, ISBN: 9780470068526. [Online]. Available: http://www.cl.cam.ac.uk/%7B~%7Drja14/book.html.

An essential read to understand the whole security picture. Cyber-security is only part of the security picture. The best security requires thoughtful engineering to fully reduce risks.

[42] BSI, *PAS 1885:2018 The fundamental principles of automotive cyber security - Specification*, 2018.

Following on from the UK Government's Key Principles of Cyber Security for Connected and Automated Vehicles, providing guidance on managing risks, security and the related safety aspects. Encourages a secure-by-design approach and vehicle systems that are resilient to attack.

[43] S. Bayer, T. Enderle, D.-K. Oka, and M. Wolf, "Automotive Security Testing-The Digital Crash Test," in *Energy Consumption and Autonomous Driving Proceedings of the 3rd CESA Automotive Electronics Congress, Paris, 2014*, J. Langheim, Ed., Paris: Springer, 2016, pp. 13–22.

A brief look at the need to provide security testing in the the automotive field. Acknowledges that security testing is not widely used in the industry. As for J3061, beyond the known functional tests the security testing needs to address vulnerability scanning, fuzz testing and penetration tests.

[44] H. H. Thompson, "Why security testing is hard," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 83–86, 2003, ISSN: 15407993. DOI: 10.1109/MSECP.2003.1219078.

Summarises security testing as the need to find functionality that exists beyond what has been specified. Contains a diagram often replicated in other papers, succinctly showing the difference between the required functionality and the implemented system.

[45] P. Wooderson and D. Ward, "Cybersecurity Testing and Validation," in *SAE Technical Paper*, SAE International, 2017. DOI: 10.4271/2017-01-1655.

Provides a concise view of the need and challenge in integrating cyber-security testing into the vehicle systems testing process. Identifies the contributions from SAE J3061 that need to be considered.

[46] K. Strandberg, T. Olovsson, and E. Jonsson, "Securing the Connected Car: A Security-Enhancement Methodology," *IEEE Vehicular Technology Magazine*, vol. 13, no. 1, pp. 56–65, Mar. 2018, ISSN: 1556-6072. DOI: 10.1109/MVT.2017.2758179.

Drawing on previous European automotive security projects (EVITA and HEAVENS), Microsoft secure development practices (STRIDE and DREAD), J3061, and attacking Volvo cars, they suggest a Start, Predict, Mitigate, Test (SMPT) methodology for systematic security testing of vehicles.

[47] S. Talebi, "A Security Evaluation and Internal Penetration Testing Of the CAN-bus," PhD thesis, 2014.

Dicusses CAN and possible attacks against CAN, implemented as a penetration test against the Vector CANoe vehicle simulator.

[48] J. Wurzinger, P. Priller, A. Kolar, and M. Nager, "Real world evaluation of a novel security testing environment for vehicular control units via CAN networks," in *Informatik 2016*, Gesellschaft für Informatik eV, 2016, pp. 1509–1521.

Describes CAN Communication Tester (CAN-CT), a software program used to run attacks against the CAN bus and ECUs attached to it.

[49] M. Cheah, S. A. Shaikh, O. Haas, and A. Ruddle, "Towards a systematic security evaluation of the automotive Bluetooth interface," *Vehicular Communications*, vol. 9, pp. 8–18, 2017, ISSN: 2214-2096. DOI: 10.1016/j.vehcom.2017.02.008.

Examines the Bluetooth vehicle interface and attacks against it. Provides a method for enumerating Bluetooth weaknesses with an Attack Tree based Threat Modeling technique.

[50] X. Zheng, L. Pan, H. Chen, R. D. Pietro, and L. Batten, "A Testbed for Security Analysis of Modern Vehicle Systems," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 1090–1095. DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.357.

Provides information on a bench based CAN testbed based upon a National Instruments scientific computer. An example of security testing CAN is given.

[51] Tencent Keen Security Lab, "Experimental Security Assessment of BMW Cars: A Summary Report," Keen Security Lab, Tech. Rep., 2018, p. 26.

A large team of security researchers found multiple vulnerabilities in components and ECUs fitted to several models of BMW vehicles. This allowed for an attack chain to be used to control ECUs on the CAN bus.

[52] N. Rathaus and G. Evron, *Open Source Fuzzing Tools*. Burlington: Syngress, 2007, p. 210, ISBN: 9781597491952.

Provides a foundation on what is fuzz testing. Whilst primarily targeted at traditional IT the knowledge provides a foundation for fuzz testing other domains. Contributed to by Charlie Miller who went on to compromise vehicles.

[53] A. Takanen, J. DeMott, and C. Miller, *Fuzzing for Software Security Testing and Quality Assurance*, ser. Artech House information security and privacy series Fuzzing for software security testing and quality assurance. Norwood: Artech House, 2008, ISBN: 978-1-59693-214-2.

Introduces fuzz testing and its use for software quality and security testing. Co-authored by Miller who later produced seminal work on car hacking.

[54] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: results from a survey," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing - JAMAICA 2014*, San Jose, California: ACM, 2014, pp. 1–6, ISBN: 9781450329330. DOI: 10.1145/2631890.2631891.

A survey of testing methods used in the automotive domain. The results show that functional testing dominates.

[55] S. Bayer and A. Ptok, "Don't Fuss about Fuzzing: Fuzzing In-Vehicular Networks," in *escar Europe 2015*, Cologne: isits AG International School of IT Security AG, 2015, pp. 1–10.

Covers fuzzing in an automotive domain. Provides an account of fuzzing a UDS implementation though results are only given at a high level.

[56] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The Oracle Problem in Software Testing: A Survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015, ISSN: 0098-5589. DOI: 10.1109/TSE.2014.2372785.

Surveys the issues related to test oracles (determining whether systems outputs are correct or not) and provides commentry on the challenges faced in software testing due to a lack of oracle automation.

[57] P. Lapczynski, H. Heinemann, T. Schöneberger, and E. Metzker, "Automatically Generating Fuzz Tests from Automotive Communication Databases," isits AG International School of IT Security, Detroit, Tech. Rep., 2017, p. 12. [Online]. Available: https://www.escar.info/escar-usa/history.html.

A commercial report that describes the integration of the Open Source booFuzz Python tool with the commercial Vector CANoe vehicle network simulator.

[58] R. Nishimura, R. Kurachi, K. Ito, T. Miyasaka, M. Yamamoto, and M. Mishima, "Implementation of the CAN-FD protocol in the fuzzing tool beSTORM," in *2016 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2016, pp. 1–6. DOI: 10.1109/ICVES.2016.7548161.

A commercial fuzzing tool is configured to fuzz a device with a CAN FD interface. Covers issues with processing time and provides summarised results but little in-depth results data.

[59] D. K. Oka, A. Yvard, S. Bayer, and T. Kreuzinger, "Enabling Cyber Security Testing of Automotive ECUs by Adding Monitoring Capabilities," in *Embedded Security in Cars Conference, 15th escar Europe*, Berlin: isits AG, 2016, pp. 1–13.

An ECU security test system is described, a comparison module acts as a test oracle to determine correct operation of the ECU when a commercial fuzzer (Defensics) sends fuzzed messages onto the CAN bus.

[60] R. Kurachi and T. Fujikura, "Shift Left: Fuzzing Earlier in the Automotive Software Development Lifecycle using HIL Systems," in *16th escar Europe*, Brussels: International School of IT Security AG, 2018, p. 9.

Proposes fuzz testing a SUT alongside an identical (not subjected to fuzz testing) system as a reference, comparing the results from the two SUTs to spot issues. Uses a dSpace HIL system with the commercial Defensics fuzzing tool. Describes a high-level view of the system but only provides a brief summary table and no detailed results.

[61] C. Miller and C. Valasek, "Car Hacking: For Poories a.k.a. Car Hacking Too: Electric Boogaloo," p. 27, 2014. [Online]. Available: http://illmatics.com/car_hacking_poories.pdf.

Automotive cyber-security researchers write about their practical experience. They give tips that can lower the practical costs required to perform research in the domain.

[62] R. K. Dutta, "A Framework for Software Security Testing and Evaluation," PhD thesis, Linköping University, 2015, p. 89.

An output from the HEAVENS project, used white box testing and malformed data injection (termed fuzzing). The testing found several issues with software. Referred to the DSRM as their methodology.

[63] M. Cheah, J. Bryans, D. S. Fowler, and S. A. Shaikh, "Threat Intelligence for Bluetooth-enabled Systems with Automotive Applications: An Empirical Study," in *3rd Workshop on Safety and Security of Intelligent Vehicles (SSIV 2017)*, Denver, 2017, p. 8.

A study on weaknesses in the use of Bluetooth technology in vehicles. Notably the use of older versions of Bluetooth which expose vehicles to lower security.

[64] P. S. Oruganti, M. Appel, and Q. Ahmed, "Hardware-in-loop Based Automotive Embedded Systems Cybersecurity Evaluation Testbed," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, ser. AutoSec '19, New York, NY, USA: ACM, 2019, pp. 41–44, ISBN: 978-1-4503-6180-4. DOI: 10.1145/3309171.3309173.

Describes the ongoing development of a multi-modal automotive cyber-security testbed. The aim of the testbed is to allow for security testing of several in-vehicle and out of vehicle systems. It references the work published from this research.

[65] ELM Electronics, *ELM Electronics: OBD*, 2017. [Online]. Available: https://www.elmelectronics.com/products/ics/obd/ (visited on 04/11/2017).

ELM is a common format used in consumer devices to connect to the OBD port.

[66] J. P. Dunning, "Taming the Blue Beast: A Survey of Bluetooth Based Threats," *IEEE Security and Privacy*, vol. 8, no. 2, 2010.

Covers Bluetooth technology, its threats and mitigating them.

[67] SAE International, *SAE J1979 E/E Diagnostic Test Modes*, 2014. [Online]. Available: http://standards.sae.org/j1979_201408/.

The specification for the minimum OBD diagnostics messages to be supported by test equipment.

[68] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W. R. Harris, S. Jha, T. Peyrin, A. Poschmann, and S. Chakraborty, "Security Challenges in Automotive Hardware/Software Architecture Design," in *Proceedings of the Conference on Design, Automation and Test in Europe*, Grenoble: EDA Consortium, 2013, pp. 458–463, ISBN: 9781450321532. [Online]. Available: https://dl.acm.org/citation.cfm?id=2485398.

Summarises some of the security threats to vehicle electronics. Coverage of threats to electric vehicles and their battery systems is provided.

[69] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks - Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011, ISSN: 09518320. DOI: 10.1016/j.ress.2010.06.026.

Early bench based CAN bus attacks by German researchers. They apply well-known IT security concepts and properties to automotive systems. They recommend that cyber security be improved in vehicles and discuss countermeasures.

[70] EPA, "The Clean Air Act in a Nutshell: How it Works," EPA, Washington, D.C., Tech. Rep., 2013, p. 23. [Online]. Available: https://www.epa.gov/sites/production/files/2015-05/documents/caa_nutshell.pdf.

The US Government's Clean Air Act was introduced to address environment pollution. It include provisions to reduce vehicle emissions, which accelerated the adoption of computer controlled systems in vehicles, as a means of achieving a cleaner fuel burn in the engine.

[71] D. F. Moyer and S. M. Mangrulkar, "Engine Control by an On-Board Computer," in *1975 Automotive Engineering Congress and Exposition*, SAE International, Feb. 1975. DOI: 10.4271/750433.

The introduction of computer control to engines increased fuel efficiencies. This early example shows improvement in efficiency from 10% to 20%.

[72] G. H. Czadzeck and R. A. Reid, "Ford's 1980 Central Fuel Injection System," in *Passenger Car Meeting & Exposition*, SAE International, Feb. 1979. DOI: 10.4271/790742.

A 1979 description of Ford's first digitally controlled fuel delivery system using an Engine Control Unit (ECU).

[73] F. P. Caiati and J. F. Thompson, "The Feasibility of a Car Central Computer," in *1973 International Automotive Engineering Congress and Exposition*, Detroit: SAE International, Feb. 1973, pp. 125–145. DOI: 10.4271/730126.

In the early 1970's General Motors used a PDP computer to control several vehicle functions.

[74] BMW, "BMW Technical Training - F30 General Vehicle Electronics," BMW Group, Munich, Tech. Rep., 2012, p. 78.

A vehicle's maintenance and repair is covered by workshop manuals. Usually provided by a manufacturer via an online portal. The manuals will include wiring diagrams that cover ECUs and the relevant fault diagnosis procedures.

[75] M. Broy, "Challenges in automotive software engineering," in *Proceeding of the 28th international conference on Software Engineering - ICSE '06*, Shanghai: ACM, 2006, pp. 33–42, ISBN: 1595933751. DOI: 10.1145/1134285.1134292.

Discussed are the issues that a highly computerised vehicle presents due to the complexity of the hardware and software platform. There is discussion on how the vehicle has a whole has become a programmable platform as the sensors and computers reach into all parts of the vehicle and are connected by communication busses. This raises issues that have been seen in other industries, but other issues exist due to the historical operation of vehicle manufacture.

[76] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008, pp. 363–369.

Covers the challenges presented through the emergence of the CPS. The interface of discrete computational components with analogue physical components in large complex systems needs new abstractions to understand and design the resultant complexity.

[77] R. Coppola, M. Morisio, and P. Torino, "Connected Car: Technologies, Issues, Future Trends," *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–36, 2016, ISSN: 15577341. DOI: 10.1145/2971482.

A comprehensive examination of what a connected car means, the services provided and an overview of the technologies involved. Includes in the discussion autonomous vehicles, software engineering and security issues.

[78] European Commission, "Cybercars, the ecological future of automobile technology," Tech. Rep., 2004, p. 2. [Online]. Available: http://cordis.europa.eu/result/rcn/45177_en.html.

The European CYBERCARS project (CYBERnetic CARS for a new transportation system in the cities) ran from 2001 to 2004. It was interested in how small electrically powered autonomous vehicles could help improve transportation in the urban environment and thus reduce issues associated with traffic congestion.

[79] F. Li and Y. Wang, "Routing in vehicular ad hoc networks: A survey," *IEEE Vehicular Technology Magazine*, vol. 2, no. 2, pp. 12–22, 2007, ISSN: 1556-6072. DOI: 10.1109/MVT.2007.912927.

A survey on Vehicular Ad Hoc Networks, VANETs are required to implement Intelligent Transportation Systems. Provides an overview of the key concepts and technology for V2V and V2I.

[80] KMPG, "Connected and Autonomous Vehicles – The UK Economic Opportunity," SMMT, Tech. Rep. March, 2015, pp. 1–24.

This report for the Society of Motor Manufacturers and Traders looks at the positive potential impact of CAVs on the UK economy.

[81] NHTSA, "Vehicle-to-vehicle communications: Readiness of V2V technology for application," National Highway Traffic Safety Administration, Washington, DC, Tech. Rep., 2014, p. 305.

The US National Highway Traffic Safety Administration looks at the benefits of introducing V2V technology, particular with regards to road safety. Covers privacy and security issues.

[82] V. S. Zeimpekis, C. D. Tarantilis, G. M. Giaglis, and I. E. Minis, *Dynamic fleet management: concepts, systems, algorithms & case studies*. Springer Science & Business Media, 2007, vol. 38.

Logistics management has a long history of using routing algorithms to improve efficiency. The latest vehicle connectivity provides for dynamic situational management. As illustrated by papers in this book.

[83] Bosch, "CAN Specification Version 2.0," Tech. Rep., 1991, p. 72.

Bosch defined the first specification for the Controller Area Network. It subsequently evolved to become ISO 11898.

[84] ARB, *Modifications to Malfunction and Diagnostic System Requirements for 2004 and Subsequent Model-Year Passenger Cars, Light-Duty Trucks, and Medium-Duty Vehicles and Engines (OBD II), Section 1968.2, Title 13, California Code Regulations*, Sacramento, 2002. [Online]. Available: https://www.arb.ca.gov/regact/obd02/att2.doc.

The Californian Air Resources Board issued requirements for emissions testing of vehicles. This includes the location and access to the OBD port.

[85] J. Wetzels, "Broken keys to the kingdom: Security and privacy aspects of RFID-based car keys," *CoRR*, vol. abs/1405.7, p. 20, 2014. arXiv: 1405.7424. [Online]. Available: http://arxiv.org/abs/1405.7424.

A comprehensive review of the security of vehicle wireless key fobs. Reveals the security mechanisms used in such keys and the weaknesses in the security implementations.

[86] J. D. Howard and T. A. Longstaff, "A common language for computer security incidents," *Sandia National Laboratories*, vol. 10, p. 32, 1998.

Establishes a high level taxonomy for computer security.

[87] C. Miller and C. Valasek, "CAN Message Injection - OG Dynamite Edition," Tech. Rep., 2016. [Online]. Available: http://illmatics.com/can%20message%20injection.pdf.

Discusses the techniques used for ECU reverse engineering for penetrating vehicle systems, taking their Jeep hack as the example.

[88] K.-T. Cho and K. G. Shin, "Error Handling of In-vehicle Networks Makes Them Vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna: ACM, 2016, pp. 1044–1055, ISBN: 9781450341394. DOI: 10.1145/2976749.2978302.

Discusses in detail attacks on the CAN bus via bus-off states. Shows how a bus-off attack can be used to disable ECUs and CAN networks. Demonstrated against vehicles and bench based networks.

[89] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A Stealth, Selective, Link-Layer Denial-of-Service Attack Against Automotive Networks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings*, M. Polychronakis and M. Meier, Eds., Cham: Springer International Publishing, 2017, pp. 185–206, ISBN: 978-3-319-60876-1. DOI: 10.1007/978-3-319-60876-1_9.

Attacking the CAN protocol at the bit level can induce errors to cause a CAN node to enter a bus-off state. This is demonstrated as an attack against the parking sensors functionality of a car.

[90] S. Abbott-McCune and L. A. Shay, "Intrusion prevention system of automotive network CAN bus," in *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*, 2016. DOI: 10.1109/CCST.2016.7815711.

Discusses security issues with CAN based systems and investigates a mechanism to detected replay attacks.

Appendix A

The computerised vehicle

I was originally supposed to become an engineer but the thought of having to expend my creative energy on things that make practical everyday life even more refined, with a loathsome capital gain as the goal, was unbearable to me.

Albert Einstein

Small, computationally powerful, and energy-efficient embedded computers are hidden beneath the surface of all the cars manufactured today. In this appendix, a brief discussion on the origination of the computerised vehicle is given, plus the emergence of the connected car. This was originally part of the literature review (Chapter 2), however, length consideration has moved some of the historical content out of the main thesis to here.

A.1 The rise of the ECU

Computers in vehicles began in the last quarter of the 20th century, there was an increase in the number of new vehicle designs that included electronic and then computational systems. The addition of microprocessor systems to vehicles was initially spurred on by the Clean Air Act in the United States [70]. The act was introduced to reverse the increasing levels of smog in US cities. To meet the requirements of the new pollution limitation regulations, i.e. emissions targets, manufacturers fitted computers, called Electronic Control Units (ECUs), to vehicle engines. The ECUs improved fuel efficiency (and hence reduced pollutants) by improving fuel burn in combustion engines [71][72].

Since the first ECUs were used to control engine functions, they were sometimes referred to a Engine Control Units. However, ECU is a general term for any vehicular embedded computing device. Another term occasionally used for an ECU is On-board Unit (OBU).

In the 1970s automotive engineers began to experiment with digital electronics and computer control of other vehicular functions [73], Figure A.1 showing the dashboard of a General Motors

(GM) early 1970's *Alpha* experimental car. By the first decade of the 21st century new vehicle designs had deployed digital computers to simplify wiring and add enhanced functionality.

Fig. A.1 Early 1970's General Motors Alpha experiments, used with permission from the GM Heritage Center.

Multiple ECUs have become commonplace, a mid-range executive car can have 30 to 40 ECUs [74] covering the entire range of vehicle functions. In a luxury limousine, the number of computers can approach 100. An entry-level car may have less than 10 computers.

All the computers in vehicles are interconnected using data networks. Typically the systems are functionally segmented, see Figure A.3 and Table A.1. A network of computers for the power-train (engine, fuel, transmission). A subsystem focused entirely on the safety of the vehicle and occupants (anti-lock braking systems and airbag deployment). A network for cabin functions (often called body control). A network for entertainment systems (sometimes referred to as infotainment), and wireless connectivity and telematics. Finally, a network for external functions (lights and parking sensors). A central gateway computer will provide cross-segment functionality, such as turning on rear lights when reverse is engaged, or allowing diagnostics information from all computers to be accessed.

In essence, an ECU is a specialist computer hardened for use in a vehicle, often referred to as an embedded computer. The central processing unit (CPU) in an ECU is referred to as a microcontroller or MCU. A MCU is normally a highly integrated device with on-board memory and input and output ports. This reduces component counts in embedded applications. The ECU runs custom software written in C code (produced by developers or generated from models designed by engineers).

Fig. A.2 Emergency assistance, navigation and cell phone integration are examples of computer-based features.

Increasingly the power and sophistication of the ECUs have allowed for full operating systems to be used (including Linux, Android, QNX and versions of Microsoft Windows). The development of autonomous vehicles (a.k.a. driverless cars) is adding more sensors and computers, turning cars into mobile supercomputers.

An issue noted by Broy [75] is that the highly computerised vehicle presents challenges due to the complexity of the hardware and software platform. It impacts traditional vehicle engineering due to the management challenges of software-based projects. The established automotive mechanical engineering, being based around a parts supply chain, has to contend with the differences between the computer and mechanical domains. The car as a whole has become a programmable platform as the sensors and computers reach into all parts of the vehicle and are connected by data networks. The road vehicle has moved into the CPS arena. The challenges presented are not limited to the automotive field, they apply to other types of CPS [76].

These new programmable platforms provide new modes of functionality. Previously software was initially a solution to an engineering requirement. Now the software in these programmable CPSs drives the engineering as new features are offered. A point made by Broy is how to ensure the increasing software complexity can be handled without loss of control of safety and costs. Modelling is proposed as a solution, how systems can be modelled from the top down. However, Broy does not discuss other factors that have driven computerisation of vehicles; this includes market forces (economics and consumer requirements) and governmental, environmental and safety legislation.

In becoming CPS entities our cars, previously entirely dependent upon driver actions, now require the electronics, sensors, computers and software to enable operation. Furthermore, the car is no longer an isolated machine. Manufacturers have added wireless connectivity to vehicles in order to provide additional services to the drivers [77]. The vehicle's wireless connectivity uses high speed data (3G and later) technology from telecommunications companies. The connectivity is supplemented by the devices the driver and any passengers carry (e.g. cell phones), thus even older vehicles can

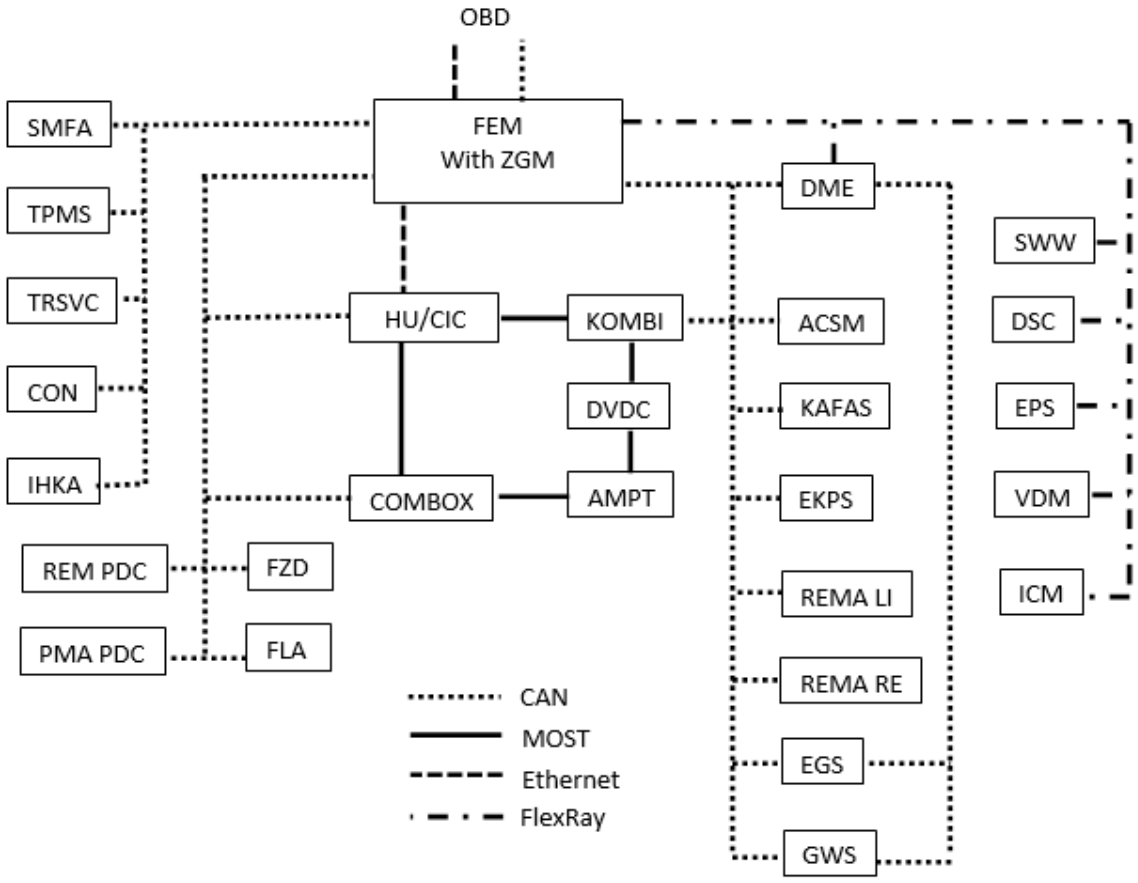


Fig. A.3 In-vehicle networks in a medium-sized executive car [74], see table for abbreviations

become *connected cars*. This wireless connectivity allows the connected car to access multiple remote information exchanging services, and this sees changes in how vehicles are used.

A.1.1 The growth in connected and autonomous vehicles

CAVs would not be possible without advances in computer-based vehicular functions. New and improved sensors, electronics, microprocessors and software allow for continual innovation by vehicle manufacturers and their component suppliers. These innovations add to the computerised vehicle to provide value-added functions for the driver and passengers. These functional additions are for improved safety, comfort, convenience and entertainment. Some functions provide product differentials or unique selling points for manufacturers. Innovation in the wider transportation infrastructure sees the computerised vehicle as an entity within city-wide and countrywide Intelligent Transportation Systems (ITS). ITS and CAV are seen as technologies to aid in accident, traffic and pollution reduction for the benefit of future societies [78].

Table A.1 ECUs in a German executive car

Reference	*Function*	*Reference*	*Function*
ACSM	Crash Safety Module	ICM	Chassis Management
AMPT	Hi-Fi Amplifier	IHKA	Air Conditioning
COMBOX	Emergency Call and Media	KAFAS	Driver Camera Systems
CON	Controller (For Driver Input)	KOMBI	Instrument Cluster
DME/DDE	Engine Electronics	PMA PDC	Parking Assistant
DSC	Dynamic Stability Control	REM PDC	Rear Parking Module
DVDC	DVD Changer	REMA LI	Left Automatic Reel
EGS	Transmission Control	REMA RE	Right Automatic Reel
EKPS	Fuel Pump Control	SMFA	Driver Seat Module
EPS	Power Steering	SWW	Lane Change Warning
FLA	High Beam Assistant	TPMS	Tire Pressure Monitoring
FEM	Front Electronic Module	TRSVC	Vision Camera Control Unit
FZD	Roof Function Centre	VDM	Vertical Dynamics
GWS	Gear Selector Lever	ZGM	Central Gateway Module
HU/CIC	Head unit/Car Computer		

The number of computerised systems in a vehicle is extensive, see Section A.1.2 below. The computerisation of vehicular systems continues to increase in complexity as further technological advances are made. The highly advanced CAVs use radar and vision systems, machine learning and machine intelligence software to perform the driving functions of a human. Furthermore, each new version of a system has lower costs and weight yet can do more, facilitating new engineering applications, however, complexity increases.

As vehicular systems continue to increase in functionality the car connections multiply to cover vehicle to vehicle (V2V), vehicle to infrastructure (V2I) and vehicle to device (V2D) applications, collectively V2X. V2V and V2I are an essential feature of Intelligent Transportation Systems (ITS) [79]. The number of cars fitted with connectivity continually increases as the vehicles on the road age and are replaced with new models. The Society of Motor Manufacturers and Traders (SMMT) estimate that by 2025 all cars sold will be connected vehicles and the connected cars on the road will outnumber unconnected vehicles [80]. The advantages of V2V safety systems are regarded as particularly beneficial with the potential to decrease vehicle accident rates and reduce the loss of life, serious injury, impact on emergency services, infrastructure repairs, traffic flows and insurance costs [81].

A connected car allows for the provision of many value-added services to a vehicle driver and the passengers (Table A.2). Any such service is based on the exchange of information between the vehicle and the remotely located service provider. The information exchange is in the form of digitally encoded data (e.g. for route guidance: vehicle location via GPS, vehicle identification, vehicle speed).

Table A.2 List of connected vehicle services

Description of Service
Crash detection and response
Route guidance (navigation, traffic reports and avoidance, local facilities)
Weather reports
Breakdown assistance
Vehicle diagnostics
Stolen vehicle tracking and recovery
Personal safety alerts
Internet connectivity (Wi-Fi hotspot)
Safer driving guidance (behavioural footprinting, insurance telematics)
Fleet monitoring (fleet management telematics services, efficient logistics routing)
Vehicle to infrastructure (V2I) information and safety notices
Vehicle to vehicle (V2V) sensing
Vehicle apps, vehicle to device (V2D) connectivity
Vehicle functionality via smartphone apps (e.g. locate, lock/unlock, preheat, servicing)

The use of vehicle connectivity is not limited to private cars. The efficient delivery of goods and maximising resources is a concern for the logistics industry [82]. In that case data from and to connected vehicles is used by algorithms in fleet management systems to maximise resources (vehicles and drivers) and minimise costs (time, fuel and mileage). However, the new-found functionality achieved via vehicle connectivity has downsides. As well as the increase in engineering complexity, there is the perennial IT problem of cyber-security, which has now entered the automotive field. Indeed, vehicle connectivity, combined with having nearly all vehicle subsystems using some form of computer control, means there is now a permanent link between cyber-security and physical safety in the cyber-physical systems that are modern vehicles.

A.1.2 List of computerised vehicle functions

The following is a (non-exhaustive) list of vehicular functions that are or can be computer-controlled. It is non-exhaustive because computerised vehicular functions are under constant development by vehicle manufacturers and their suppliers. These computerised functions are developed in response to technological advances, innovations, technology cost reductions, consumer demands and product differentiation:

- controlling the engine and regulating power and emissions, via controlled fuel and air delivery (for fossil-fuelled engines), and controlling energy charging and discharging in hybrid and electric engines;
- controlling gear changes and power delivery to the driving wheels;
- monitoring wheel traction and tyre pressure and providing handling and skid control;

- monitoring and operating actuators for windows, wiper blades, washers, doors, locks and sunroofs;
- controlling radio, media and navigation playback (infotainment);
- running the security systems, including arming and sounding the security alarm and disabling the engine for theft deterrence, providing keyless entry and starting, and central locking;
- providing advanced driver assistance services, for example, cruise control, parking sensors and park assist, collision avoidance (automatic braking and steering), lane-keeping assistance, blind spot warning;
- linking with driver and passenger devices (smartphones, tablets and computers) via Bluetooth and Wi-Fi, and providing software applications (apps) through these devices or via a manufacturer's portal;
- updating the cars displays (dashboard) and responding to driver and passengers operation of the vehicle's switchgear and controls;
- controlling the vehicle's interior climate (air circulation via fans, air conditioning, heating, heated seats, window demisting);
- de-icing windows and mirrors.

Computerised vehicle functions can originate from electronic control of previously mechanical and hydraulic systems (e.g. braking and steering), or functions that result from the availability of new sensor technologies and improved computational power (e.g. blind spot warning, lane-keeping assist). The development of a computerised function does not necessarily lead to its presence in a series production vehicle. The adoption is driven by factors that include production cost, energy requirements (the load on the vehicle's electrical system), size, weight, return on investment, market requirements and local laws.

A.2 Summary

This appendix was derived from the work performed for the literature review in Chapter 2. The literature review provides enough information to support the background and motivation for this research, however, this appendix provided some additional historic and background information that may be useful to other readers. Covered was the rise of the computerised car from early work in 1970's and the introduction of ECUs to improve emissions from combustion engines, to the rise of the connected car. The list of computerised vehicle functions and remotely provisioned services are now extensive.

Appendix B

More on the CAN bus and OBD port

One of the biggest risks for autonomous vehicles is somebody achieving a fleet wide hack.

Elon Musk

The literature review, Chapter 2, introduced the CAN bus and the OBD port, two pieces of technology common to most cars, and both are examined in this research. Whilst the literature review provided introductory technical information, additional detail is provided here for readers who want to further their understanding of these vehicle technologies. Although this appendix expands upon the overview information provided in the literature review, it does not provide extensive depth. For in-depth technical information use the references provided in this appendix.

B.1 A brief history of CAN

CAN has evolved since its initial release. There are now three types of CAN protocol, here is a short history.

- In 1983 the German company Robert Bosch GmbH starts development on a new serial bus for vehicles. They had failed to find a suitable serial bus for use in vehicles to support new functionality.
- The first Automotive Serial Controller Area Network was announced in 1986 [83]. Thus it precedes the World Wide Web and was designed in the era prior to widespread connectivity.
- In 1987 Intel delivers the first CAN integrated circuits.
- Standard CAN, the first CAN protocol, has a data packet with an 11-bit packet identifier (id) field, a 4-bit data length field, and up to eight bytes (64 bits) of data. (Standard CAN was observed in all vehicles and components used in this research.)

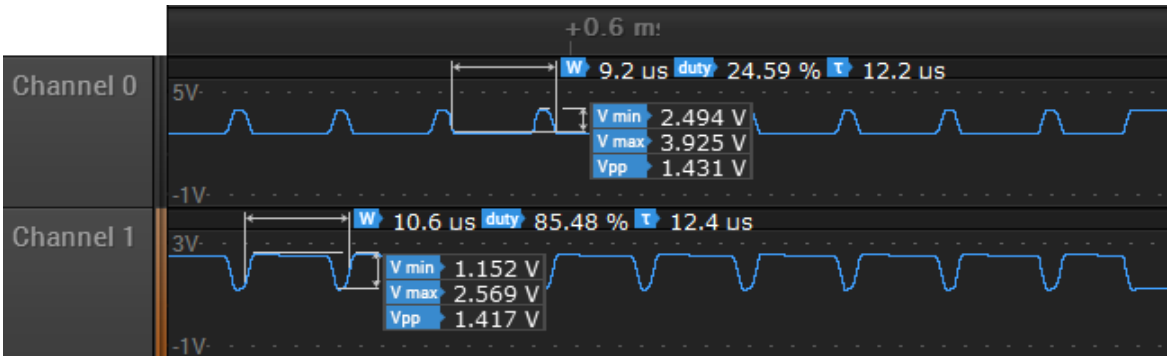


Fig. B.1 Part of a CAN signal from a bench measurement showing voltage levels, the 2.5v recessive (logic 1) level is taken to 1.2v for CAN low and 3.9v for CAN high to create the dominant (logic 0) level, a 2.7v differential (and 0.7v above a nominal differential of 2 volts)

- 1991 sees an updated CAN specification and the release of Extended CAN. The second version of the CAN specification was split into part A for *Standard CAN* and part B for *Extended CAN* and hence Extended CAN is sometimes referred to as CAN 2.0B. The main change in Extended CAN was the introduction of a 29-bit id field.
- The increase in data transmission volumes in vehicles (including the time taken when *flashing* new firmware to vehicles) saw CAN bus utilisation design issues. The introduction of a new CAN with Flexible Data Rate, or simply CAN FD, in 2012 was in order to increase data throughput. There were two main changes. Firstly, an increase in the maximum size of the number of data bytes that a packet can contain, from 8 to a maximum of 64. Secondly, the data transmission speed can be increased during the transmission of the data bytes packet section (hence Flexible Data Rate). This is to keep the overall packet transmission time in line with normal CAN. CAN FD devices are able to co-exist with standard CAN devices.

For a more detailed history of CAN see https://www.can-cia.org/can-knowledge/can/can-history/. Although originating from Bosch, CAN is now specified in an international standard, ISO 11898.

B.2 An overview of CAN

CAN uses twisted-pair wire to connect devices in a bus topology (with 120 ohm termination resistors at each end of the bus), supporting physical (a grounded or cut wire) and electrical (electromagnetic interference) fault tolerance due to differential signalling (see Figure B.1).

B.2.1 Data transmission speed

Compared to a typical enterprise computing environment the transmission speed of CAN is modest. When CAN was originally designed data speeds were low, in the tens to hundreds of kilobits per second (kbps). Subsequent CAN specification releases have increased transmission speeds to a more

modest 2 megabits per second (mbps) maximum throughput. However, a common transmission speed used for in-vehicular networks is 500 kbps.

The data transmission on the CAN bus is continuous, as the vehicle's state is constantly updated for effective command and control, and to provide human feedback on vehicle status. This means a CAN bus has a high load, usually 50 to 70 percent. The complexity of modern cars means that there may be more than one CAN bus in a vehicle, several networks are common.

In the literature review in Chapter 2, a schematic of a CAN data packet was provided in Figure 2.5. The software in the ECUs using CAN as a communications network read and write the packet id, data length, and data bytes. The protocol control is handled automatically by the CAN interface hardware.

B.2.2 Packet id

Each device, or node, on the bus, can initiate data transmission, i.e it is a multi-master network, with only one node at a time transmitting a complete packet. The protocol is designed to allow a higher priority packet to continue transmission in the event of two to more nodes transmitting simultaneously. Packet priority is determined via the id, encoded into 11-bits (Standard CAN) or 29-bits (Extended CAN).

The lower the id the higher the priority of a packet. The id is referred to as the *Arbitration Field*. For example, a packet with an id of zero would take priority over a packet with an id of 16. In which case packet 16 would re-transmit once the data bus was quiet.

B.2.3 Data length and data bytes

Although the Standard CAN payload is only up to eight bytes in length, this is all that is required for the main application of CAN in a vehicle. CAN is used to transmit sensor, switches and actuator settings to and from ECUs located around a vehicle.

For example, a door open sensor only requires a single bit. A window position sensor can use a single byte (256 positions) or less (a half byte or nibble for 16 positions may suffice). Thus, the data packet size only needs to be small.

The number of data bytes in a frame is stored in the Data Length Code (DLC) field, encoded into 4-bits and transmitted immediately before the data. Data is transmitted as common 8-bit bytes.

B.2.4 Control bits

The CAN hardware adds control, CRC and other error handling bits to the transmitted packet. The CRC is sent immediately after the data bytes. The other bits sent in a complete data frame, e.g. End of Frame (EOF), are used by the CAN transmission protocol, implemented and handled automatically in CAN transceiver chips. The chips pass on the packet id, data length, data bytes and error states to the higher-level application (e.g. the software in an ECU).

B.3 CAN example

In Figure B.2 an ECU microcontroller is reading a switch, e.g. on a dashboard. The switch state (shown on) is encoded into the second bit of a data byte (`00000010` or `0x2` hex). This is sent out by the CAN transceiver as a data packet with id 8 and one byte of data. In the receiving ECU, handling the vehicle lighting, the MCU software sees that the bit for the headlight is set and turns it on. Likewise, when the switch is off the data byte is zero (`00000000` or `0x0` hex) and the light is turned off, Figure B.3.

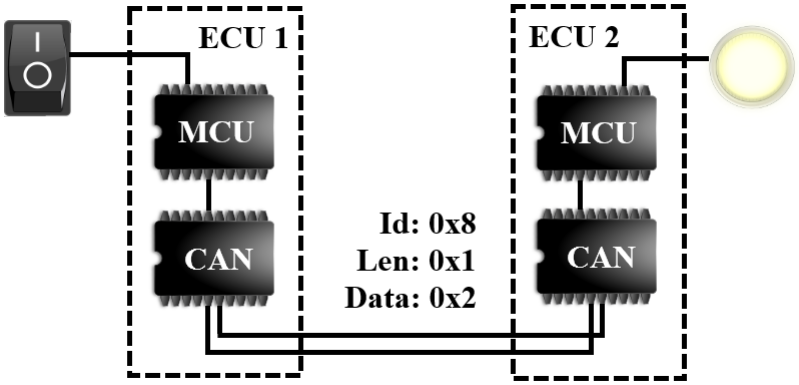


Fig. B.2 Transmitting vehicle sensor data on CAN, here packet id 8 is sending data to turn on a headlight

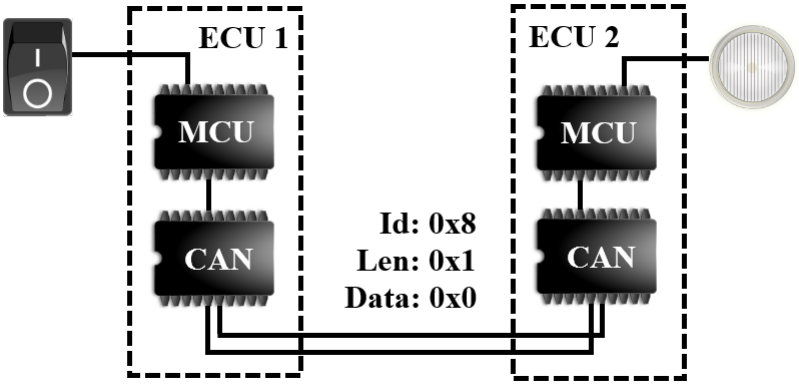


Fig. B.3 CAN transmissions are continuously updating vehicle state, here packet id 8 is turning off the headlight

A modern car has hundreds of sensors, lights, actuators, switches and motors. Therefore, many CAN packets are constantly sent around the vehicle's internal networks to operate and control the car, and to keep the driver informed of the vehicle's state. Table 2.3 shows a brief amount of CAN data read from a vehicle. Several CAN packets have been captured, each packet having 8 bytes of data.

Table B.1 Section of CAN data captured from a car

Time	*ID*	*Length*	*Data*
3.157253	190	8	00 00 00 01 93 01 00 00
3.157551	275	8	00 00 FF 00 00 00 00 00
3.157850	2C1	8	00 00 E0 00 00 00 00 00
3.158144	400	8	01 17 00 01 01 92 01 04
3.158430	405	8	01 66 4D 4E F2 49 FF FF
3.158708	430	8	51 64 01 35 59 14 78 C0
3.159010	432	8	28 00 00 00 00 00 00 40
3.159294	433	8	00 19 35 00 00 00 ED 08
3.159584	4E3	8	CC 00 00 02 10 4C 00 06

B.4 The On-Board Diagnostics port

The On-Board Diagnostics (OBD) port, Figure 2.6, is a legal requirement for the provision of emissions data. Beyond the legal requirement manufacturers use it for other maintenance purposes. This includes the ability to update ECU software (called firmware) and use spare pins on the OBD connector to add manufacturer-specific functionality, see Table B.2.

Table B.2 Usage of OBD port pins

Pin	*Use*
1	Manufacturer defined, e.g. VW/Audi/BMW ign. on
2	SAE J1850 PWM Bus+
3	Manufacturer defined, e.g. Chrysler CCD bus+
4	Chassis Ground
5	Signal Ground
6	CAN+ (High), ISO 15765-4, SAE J2284
7	K-Line (ISO 9141-2, ISO 14230-4)
8	Manufacturer defined, e.g. BMW second K-Line
9	Manufacturer defined, e.g. BMW engine RPM
10	SAE J1850 PWM Bus-
11	Manufacturer defined, e.g. Chrysler CCD bus-
12	Manufacturer defined
13	Manufacturer defined, e.g Ford PCM programming
14	CAN- (Low), ISO 15765-4, SAE J2284
15	L-Line (ISO 9141-2, ISO 14230-4)
16	Vehicle Battery +ve (live 12v or 24v)

Data packets known as Diagnostic Trouble Codes (DTCs) provided the emissions and maintenance values. DTCs are read from the vehicle's ECUs over CAN. This is done with test equipment and devices known as scan tools, or dongles, which plug into the OBD port. The tools provide a wired or wireless (via Bluetooth or Wi-Fi) connection to a diagnostics computer.

The CAN data connection is a universal standard, always on pins 6 and 14. Thus, the OBD port is providing a two-way communications connection to the nervous system of the modern vehicle. As such use of the port has spread beyond its initial function. It is utilised by independent service centres, car modifiers and enthusiasts. There are many inexpensive, easily obtainable and usually unregulated aftermarket devices which connect to the OBD port (Table B.3). These devices can be used to read vehicle performance data for custom displays and provide telematics services. Examples of obtainable data through the OBD port are engine speed, vehicle speed, throttle position and tyre pressures. Plus, OBD is used by home mechanics to read DTCs when fixing their vehicles. Finally, other devices and software using OBD allegedly improve fuel consumption and/or engine performance.

Table B.3 Devices that Connect to Vehicular Systems

Aftermarket Device Type	***Connection***
Technician diagnostic tools (scan tools)	OBD, Wi-Fi
Consumer targeted diagnostic tools (including dongles)	OBD
Fleet management telematics units	OBD, Wired
Insurance telematics units for driver monitoring	OBD, Wired
High-end infotainment (or head) units	Wired
Tire pressure monitoring system (TPMS)	OBD, Bluetooth
GPS vehicle tracking units	Wired, OBD
Cell phones	Bluetooth, USB, Wi-Fi
Navigation systems	Bluetooth, USB
Consumer targeted vehicle performance enhancing dongles	OBD

The vehicle data that is available via OBD is not generally encrypted nor access controlled. Though the Unified Diagnostic Services (UDS) protocol, ISO 14229, used in most cars to access advanced ECU diagnostics, does provide the facility to implement a seed and key service. The seed-key security access may be implemented by an ECU for higher-level functions other than reading ECU data values, for example to program the ECU with a new firmware.

B.4.1 Internal operation of OBD dongle style devices

Many OBD aftermarket devices work as an OBD to RS-232 interpreter (RS-232 is a well-established serial data communications interface for computers). The signal conversion from the vehicle CAN bus to serial date is performed via a CAN transceiver chip to a MCU with a USB interface. A computer can provide a virtual RS-232 port to the USB interface. The ELM[1] MCU is a popular choice for OBD to RS-232 interfacing. The ELM chip allows a serial connection to be created between data terminal equipment (DTE), such as the computer or a cell phone, and data communication equipment (DCE), in this case, the OBD devices. The devices expose a RS-232 port via the standard Serial Port Profile (SPP) of a Bluetooth Class 2 (low power 10 metre range) wireless interface. Attention modem commands, known as AT commands, can then be sent through this serial channel via a terminal

[1]https://www.elmelectronics.com/products/ics/obd/

program (the terminal program can run on a laptop or cell phone). AT commands are a longstanding method of configuring a RS-232 device, in this case, the ELM MCU.

Once a serial connection has been established between an OBD dongle and a computer (or mobile phone), then messages can be sent from the computer to the OBD device (see Chapter 4). The sent messages cause CAN packets to be transmitted on the in-vehicle network. A relatively unsophisticated attack, often using no more than a terminal program, can disable a sophisticated vehicle. This can compromise vehicle safety and security. Furthermore, this method has been been used to steal vehicles, leading to the issue of addressing the physical security of the OBD port.

B.4.2 OBD physical security

When US legislation mandated access to the OBD systems it stated [84]:

> *the connector shall be capable of being easily identified by a crouched technician entering the vehicle from the driver's side*

This allows for convenient access for the emissions tester and service technicians, but also everyone else, including criminals. Especially when it is coupled with poor engineering, as was the case of being able to reprogram a blank keyless entry fob for a luxury car via the OBD port [85], resulting in a spike in luxury vehicle thefts.

With the physical and the operational security of the OBD port generally weak, the marketplace provides a variety of lockable OBD covers[2] providing a physical deterrent to mitigate mechanical access to the OBD port. However, a knowledgeable agent will know that, due to the physical nature of CAN, a CAN tap is possible by accessing a vehicle's wiring elsewhere. Furthermore, no study of the physical resilience testing of such locks has been performed.

B.5 Summary

In this appendix CAN bus and OBD port information was provided to supplement the overview in the literature review. A short history of CAN was given along with a brief technical description. The use of the OBD port and its physical access was described.

[2]https://www.maplefleetservices.co.uk/product/obd-protector/

Appendix C

Bit rate attack on the CAN bus

When in doubt, use brute force.

Ken Thompson

In the literature review, Chapter 2, it was discussed how previous research has demonstrated the security weaknesses of the CAN bus. Therefore, CAN's use as a data transmission network for connected vehicles needs additional security considerations. Furthermore, a vehicle's vulnerability to attacks via CAN is not helped by CAN's ease of accessibility. Not only through the use of aftermarket devices, see Chapter 4, but other methods include compromised ECUs and network wiretaps. The security weaknesses in CAN was not an issue, or an even consideration, in the pre-connected car and pre-Internet era (when CAN was designed). However, due to vehicle connectivity, any weaknesses in the CAN protocol that could be exploited by a malicious adversary needs to be highlighted as a potential security issue. In this appendix, a configuration error with a CAN node is examined for its use as an attack against vehicle systems. Note, for this work, the experiments on the physical vehicles were was performed collaboratively with HORIBA MIRA researchers and engineers.

C.1 Introduction to the experiment

In the literature review (Chapter 2) the CAN communications protocol was introduced and described (with further technical material provided in Appendix B). In that description, the data transmission throughput of the CAN bus was described as modest, in the hundreds of kilobits per second (Kbps) range, compared to the modern home and business networks running at the megabits or gigabits range.

At the CAN hardware level any discrete bit rate is configurable, up to the protocols maximum of 1Mbps. The common rates found in cars are 500kbps, sometimes referred to as high speed CAN, and 125kbps, which is sometimes referred to as medium-speed CAN.

In performing this research program it was observed that it is possible to disable a CAN bus through a simple attack. The attack is a manifestation of a known problem with the CAN bus and had been experienced during this research. The problem manifests itself as a consequence of setting

the incorrect bit rate, i.e. the CAN protocol bit rate, on a CAN bus node. The incorrect bit rate was set on several occasions during the development of the CAN fuzzer, and when developing the OBD experiment in Chapter 4.

When the incorrect bit rate was set on a device connected to the CAN node it had the effect of either disabling the connecting device or disrupting some or all of the communications on the CAN bus. The latter effectively halts all functionality for the affected network, in security terms this is a DoS attack. For vehicles, this DoS attack can result in diagnostic messages being displayed on screens, erratic engine idling and actuator operation, and a possibility of causing damage to ECUs. This problem raises a safety concern for vehicle users.

The cause of the CAN communications failure is a consequence of the protocol's design. When CAN nodes on a network detect errors at the bit level it will cause internal error counters to increase. Once the error count gets too high within a CAN node, the node will shut down, known as a *bus off* state [83]. This bus off state is intended to protect a CAN bus from faulty nodes, however, this protection mechanism is triggered by a node with an incorrect bit rate configuration.

The bit rate CAN configuration issue was discussed with automotive engineers on the MIRA Technology Park. The problem of an incorrect bit rate is encountered during vehicle systems R&D and testing, it is considered simply as a configuration error. Configuration, however, has long been recognised as a vulnerability in an attack process [86]. A search in the literature on using the bit rate configuration as a possible attack against in-vehicle systems did not uncover any published research. Therefore, presented here are experiments to examine the problem. The motivation is to determine the kinds of errors that an incorrect bit rate can cause, and whether it is justified as a threat vector.

The CAN bit rate attack threat could be via a compromised ECU or aftermarket device connected to the OBD port, or a device attached to the CAN bus elsewhere in the vehicle. Depending upon the attacker's intentions the attack could be timed to be immediate, delayed, intermittent or controlled remotely via wireless communications.

C.2 Method

In security terms, the DoS from an incorrect bit rate setting affects a systems *availability* security property. To gather data on this problem experiments are performed, the objective is to test CAN nodes and CAN packets at different bit rates and investigate the effect. Experiments are executed to meet that objective and demonstrate the use of an incorrect CAN bit rate setting as an attack against in-vehicle networks. The results are evaluated to determine the security implications.

The following experiments test the effect of incorrectly configured CAN nodes being connected to a running CAN bus. The incorrect configuration is that the bit rate on the connecting node is different from the running CAN bus. The effects that the incorrectly configured CAN node have is observed in three types of ToEs; firstly, against a simulation of a CAN bus running on a bench based testbed (Section C.3), secondly, against a vehicle component taken from a car (Section C.4), and finally against real-world vehicles (Section C.5). The details of the vehicles, including the manufacturer

names, are withheld for ethical reasons of commercial sensitivity and responsible disclosure. The use of different ToEs aids the development of the experiment and the determination of the effectiveness of the attack, both as a security testing method in itself, as well as its usefulness to a threat agent.

The USB to CAN interface used for the development of the fuzzer, see Figure 5.3 in Chapter 5, is connected to a PC to act as the attacking CAN node. It is a PEAK-System PCAN-USB device that supports fourteen different bit rates and is thus suitable for testing the bit rate attack. The supported bit rates (in Kbps) are 5, 10, 20, 33.333, 47.619, 50, 83.333, 95.238, 100, 125, 250, 500, 800, 1000 (1 Mbps).

For the physical connections to the ToE, the PCAN-USB node uses a common DB9 (9-pin D-Sub) connector, cabled according to the CAN in Automation (CiA) 303-1 specification[1]. When testing against the testbed, i.e. the CAN bus HIL simulator, the connector is directly plugged in. For both the bench based vehicle cluster component and the real vehicles, additional cabling is required. The custom cabling either connects directly to the CAN bus under test, or via an OBD port. To interface to an OBD port a DB9 to OBD cable was constructed, the wiring for the custom cable is shown in Figure C.1.

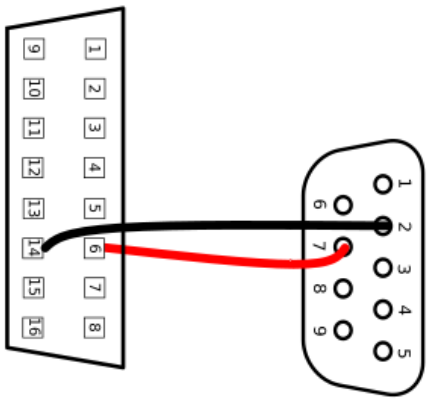


Fig. C.1 Connecting a PEAK-USB CAN interface to a vehicle OBD port

With the PCAN-USB attached to the ToE being tested different bit rate configurations are tried. For each of the PCAN-USB supported bit rates the following is performed:

1. The ToE is initialised and starts.
2. If the ToE is observed to be operating normally, continue.
3. The bit rate for the attacking CAN node (PCAN-USB device) is set.
4. In the case of the simulated vehicle, a CAN packet transmission was necessary to start the test. This is because the PCAN-USB device was not recognised as a node on the simulated network until a CAN transmission was sent. Against the real-world ToEs, all that was needed was for the PCAN device to be initialised.

[1]https://www.can-cia.org/groups/specifications/

5. Observations on the system responses were made.

6. If all bit rates have been tried then finish, otherwise choose the next bit rate, reset the ToE and go to 1.

C.3 Experiments against a simulated CAN bus

It was observed that the attacking CAN node, the PCAN-USB device, was not recognised by the testbed's HIL equipment unless a CAN data packet was transmitted. Therefore, a fixed CAN packet was defined within the fuzzer tool for transmission during the bit rate testing. Any packet could have been chosen. In this case, one of the packets observed on the simulated vehicle's CAN bus, running on the testbed is used. The packet used is shown in Table C.1. To reduce the variability of the experimental parameters all the data bytes are set to a fixed value, in this case, zero.

Table C.1 CAN data packet for the bit rate attack

	Id (Arbitration)	*Data Length Code*	*Payload*
Dec	100	8	all 0
Hex	0x64	0x08	all 0x0

The simulated vehicle's CAN bus is executed on one of the testbed's physical CAN busses. The CAN bus bit rate for the simulated car is 83.333 Kbps. The attacking node was configured with a packet cycle time 0.05s (50ms). The results from the first bit rate attack against a simulated vehicle are shown in Table C.2.

Table C.2 Bit rate attack at 50ms, results against a simulated vehicle

Attacking bit rate (Kbps)	*Observation*
5, 10, 20, 33.333, 50, 125, 250, 500, 1000	Stopped the simulation traffic
47.619, 95.238, 100, 800	Attacker entered bus off state
83.333	Partial attack success then attacker enters bus off

The same experiment was performed with a different packet cycle time, doubled to 0.1s (100ms). This was in order to test the packet cycle time as a variable in the experiment. The second attack results are shown in Table C.3.

The experimentation against the simulated vehicle on the testbed produced different results. The attacker either stopped the simulation CAN data traffic, caused the attacker to enter a bus off state, or was partially successful (with the attack functional for a short time before the attacker reports transmission errors or enters a bus off state). The CAN packet transmission rate (not the bit rate but the frequency of the packets) did cause a variation in the results of the attack. Between the 50ms and 100ms packet rates the results of the 33.333, 47.619, 83.333 and 95.238 kbps rates varied. Thus, at these rates the combination of the bit rate and packet rate determined whether the CAN bus traffic

Table C.3 Bit rate attack at 100ms, results against a simulated vehicle

Attacking bit rate (Kbps)	*Observation*
5, 10, 20, 50, 47.619, 125, 250, 500, 1000	Stopped the simulation traffic
33.333, 100, 800	Attacker entered bus off state
83.333	Runs normally but an occasional bus off observed
95.238	Attacker reports bus errors or enters bus off state

was halted or the attacker entered bus off. At the attack bit rate configuration of 83.333kbps the bus off state was unexpected. This is because the CAN bus is configured to operate at 833.333kbps and thus an attacker node configured at the same rate should not have any affect. This was the case for the 100ms packet transmission setting, but for that setting, an occasional bus off state was seen. Further experimentation is required to investigate in detail the operation at a bit level.

The experiments against the simulated vehicle on the testbed show that a bit rate attack is a possibility. The next step was to investigate the bit rate attack against physical hardware, aiming to check the attack validity on a non-simulated ToE.

C.4 Experiments against a component

An instrument cluster component that matches the one fitted to the lab car is driven from an Arduino SBC, together, the SBC and vehicle component are a small CAN bus. The bit rate for the small CAN bus is 500 Kbps. The results from a bit rate attack against the vehicle components CAN bus is shown in Table C.4.

Table C.4 Bit rate attack at 50ms, results against a physical vehicle component (instrument cluster)

Attacking bit rate (Kbps)	*Observation*
1000	Stopped the CAN traffic and thus the component
5, 10, 20, 33.333, 50, 47.619, 95.238, 100, 125, 250, 500, 800	Attacker enters bus off state
500	Attacker and component runs normally

At the instrument cluster's network correct operational bit rate (500 Kbps) the attacker did not have any impact on the system. The attack was only successful at one bit rate, at 1 Mbps, where it halted the instrument cluster network. Although the attack was successful at the one bit rate (in this case), it was a demonstration of the ability to disable a CAN bus using this tactic. The next step was to execute the attack against a running vehicle.

C.5 Experiments against vehicles

Attacks on vehicles are difficult due to their complexity and CPS nature, furthermore, vehicles can be regarded as black boxes with information on their internal systems being proprietary. This means it is not always possible to predetermine the effect of attacking the vehicle's systems. Attacks on vehicle systems via the CAN bus can result in damage to those systems [38], [87], and for a driven vehicle it has safety implications. The possible damage to vehicles must be considered due to their cost, their possible use as a shared resource, and the not insignificant repair costs (hence the value of a testbed for experimental development). To mitigate that possibility of damage in this case, the testing was limited to two vehicles, and constrained only to short bursts of exposure to the incorrect bit rates.

C.5.1 Vehicle A

The first vehicle, Vehicle A, is a small hatchback from a major manufacturer manufactured in 2013. The attacking node was attached via the OBD port. Firstly, a quick test was performed to determine if a bit rate attack would have an effect. Based upon the attack against the instrument cluster the attacking node would run a 1 Mbps with a 50ms packet transmission rate. When the attack was started the following was immediately observed:

- Engine idling disrupted with engine revolutions dropping below normal idle speed.
- Buzzers and warning lights were activated.
- Vehicle dials behaved erratically.
- Warning and malfunction messages appearing on the displays.

The attack was brief, at around five seconds, and then stopped because of the reaction of the vehicle. If the attack continued the possible effect on the vehicle is unknown.

Knowing that the attacking node has an effect, the range of bit rates supported by the attacking device (PCAN-USB) could be used. The attacking node was run at its various supported bit rates using the design procedure (Section C.2). The vehicle's reactions were observed. The vehicle's displays showed warning messages and warning lights for a variety of bit rates, these are shown in Table C.5. The vehicle's general response, as for the initial test above, was disruption of the engine speed and erratic behaviour of dials. As for the simulation, Section C.3, at the correct operational rate for the vehicle's CAN bus (500 Kbps), there was no perceived effect.

C.5.2 Vehicle B

The second test vehicle, B, is a premium five-door hatchback, a model produced from 2012 to 2018. This particular vehicle contains a gateway ECU between the diagnostics (OBD) port and in-vehicle networks. To eliminate any effects of the gateway the attacking node (PCAN-USB device) was wired

Table C.5 Vehicle A: Warning and malfunction messages at various supported bit rates (Y=appeared, N=no reaction)

bit rate (Kbps)	*Engine Malfunction*	*Immobiliser Malfunction*	*Engine Coolant*	*Cruise Control*	*ABS*	*Powertrain Warning*	*Stability Control*	*Temperature Reset*
5	Y	N	Y	Y	N	Y	N	Y
10	Y	N	Y	Y	N	Y	N	Y
20	N	N	N	N	N	N	Y	N
33.33	N							
47.619	N							
50	N							
95.238	N							
100	N							
125	N							
250	N							
500	N							
800	N							
1000	Y							

directly into the in-vehicle network (effectively bypassing the gateway). As with Vehicle A, there were marked reactions when the attacking node was attached to the network. This is summarised in Table C.6.

There were two effects observed in addition to the warning lights:

- Firstly, each time the attacking node was connected, the vehicle would beep. To prevent possible permanent vehicle damage, as with Vehicle A, the node was connected only for a short time (maximum of three beeps).
- Secondly, there was a small movement of the steering wheel, this may be due to engagement or disengagement of power steering.

All the reported effects were permanent as long as the attacking node was connected, and some effects remained even after the attacking node was disconnected. Cycling the ignition (turning the car off and back on) one or more times removed the effects. As with Vehicle A, at the correct operational rate of 500 Kbps the vehicle operates normally.

C.6 Vehicle considerations

It was not unexpected to find differences in specific reactions (such as the type of warning light), as the vehicles are different in construction. Vehicle A did not respond to as many bit rates but displayed an increased array of errors. Vehicle B was much more sensitive to the effects of the various bit rates,

Table C.6 Vehicle B: Warning and malfunction messages at various supported bit rates (Y=appeared, N=no reaction)

bit rate (Kbps)	*Inoperative*	*Restrain System Malfunction*	*Power Steering Malfunction*	*Parking Brakes*	*Ignition Coil light*
5	Y				
10	Y				
20	Y	N	Y	Y	Y
33.33	Y	N	Y	Y	Y
50	Y	N	N	N	Y
47.619	N				
95.238	Y	N	Y	N	Y
100	Y	N	Y	N	Y
125	Y	N	N	Y	Y
250	Y	N	N	Y	N
500	N				
800	Y	N	Y	Y	Y
1000	Y	N	Y	Y	Y

but had fewer error types. There are several factors that are likely to be the cause for the different reactions to incorrect bit rate settings. These include:

- The physical design of the vehicle systems; whilst each vehicle model of the same age from a manufacturer has the same internal systems, the systems used between manufacturers will vary. Furthermore, the same model of car from a manufacturer may have different options specified and hence have more or less ECUs fitted. These differences will change the characteristics of the vehicle's internal systems. Matching the exact specification of the vehicle with the errors is a non-trivial effort since these exact specifications are usually commercially sensitive and thus closely guarded.
- The CAN transceiver hardware; there are a number of different manufacturers of CAN transceiver chips. Whilst all chips are developed to pass the CAN certification tests, there will be variations in implementations that will affect CAN bus characteristics.
- The ECU hardware; many of the MCUs used within ECUs have built-in support for CAN. As for the CAN transceiver chips, there are many types of MCUs in use by different manufacturers. This will affect the characteristics of the on-board systems.
- The ECU software; the programming of the ECUs will vary greatly depending upon their functionality and design. How the error handling code is written will also be a factor. Thus, the vehicle's internal software will have an effect on the characteristics of the on-board systems and would, therefore, be likely to respond differently to induced errors.

Due to safety reasons, it was not possible to perform the bit rate attack experiments on a moving vehicle, but this would be desirable for future work (Section C.10). This would be to determine the physical threat to the vehicle and other road users. The nature of the errors appear to be severe and would be worth cataloguing in case a threat agent produces a malicious node (or compromises an existing node) to launch an attack. Thus, the tests performed here would be desirable as a defined security test. This is particularly important because of the observed physical reactions, such as the movement of the steering wheel and problems with the engine, which confirms the safety dimensions of the errors caused by the attack.

C.7 Weaponizing the attack

Further experimentation is required to enable full deployment of the attack and to provide additional data to determine, in detail, the required conditions for a successful attack.

The testing here was limited to scenarios where physical proximity was required. However, it is not inconceivable that a malicious bit rate manipulating device, with wireless communications for remote control, could be fitted to a car. Other researchers and the work on the testbed in Chapter 4 have demonstrated the ability for rogue devices to be connected to a CAN bus.

The assumption of the tests was that a malicious device had been attached, or an existing ECU compromised. Furthermore, such a device must be able to control the configuration of the bit rate. With such conditions, the bit rate attack appears viable for deployment as a real-world weapon.

C.8 Additional attacking nodes

There has been a preliminary investigation into which particular kinds of CAN nodes cause an effect. This is an initial exploration to help determine how an attacking node could be constructed and to further understand the threat. The main experimentation used the PCAN-USB device, and it was demonstrated that a bit rate attack is successful with such a device. Following this, two other devices were tried. The HIL equipment used as the testbed for the vehicle simulation (see Section C.3 and Chapter 4 for details) has Vehicle-In-The-Loop capabilities. The testbed acted as an attacking node on Vehicle A. This was also successful in performing the bit rate attack, resulting in the same error messages and physical effects appearing.

A device called CANtact[2] was then used as the attacking node (Figure C.2), which is an open source tool, a low-cost PC to USB interface, of a similar nature as the PCAN-USB device. The use of CANtact proved unsuccessful in executing the bit rate attack. This may be due to physical hardware considerations. The CANtact device uses a generic CAN transceiver (an MCP2551 manufactured by Microchip) that remains passive until initialised directly from the microcontroller. The PCAN-USB device (see Figure C.3) is implemented with a powerful ARM Cortex-M3 32-bit MCU supported by

[2]http://linklayer.github.io/cantact/

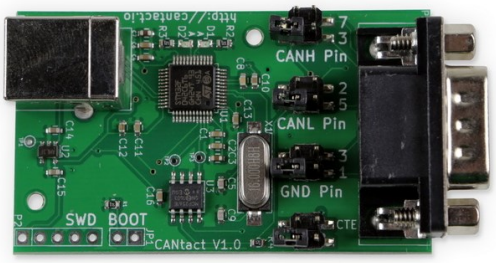


Fig. C.2 The open source CANtact device

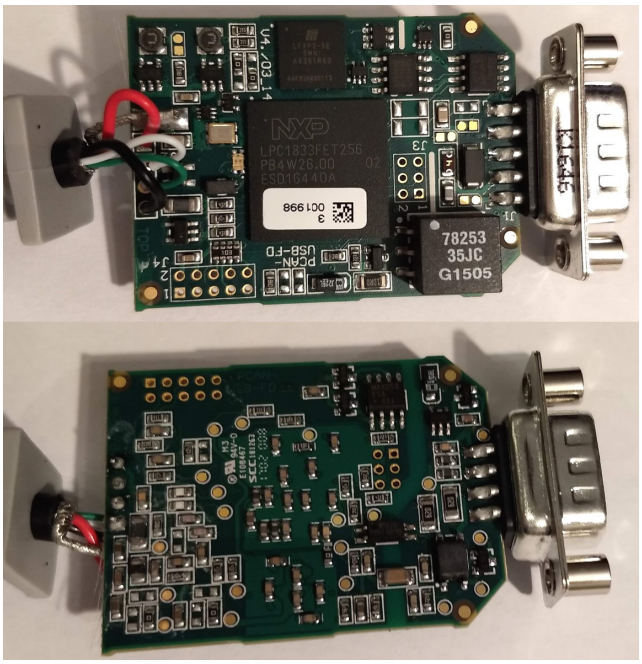


Fig. C.3 The top and bottom printed circuit board of the PCAN-USB interface, compared to the CANtact device it is a more complex design

custom hardware and is actively monitoring the CAN bus when connected. The software controlling CANtact may also be the reason it was not successful in performing the attack. Unlike the PCAN-USB device which uses the developed GUI fuzzer tool on a Windows PC, the CANtact device is driven from a Linux command line terminal. At this stage no further investigations have been performed to determine why the CANtact device did not execute the bit rate attack, it would be a topic for further work.

The weaponizing of the bit rate-attack will require further exploration of using different nodes, and the toleration of different types of vehicles to such nodes. Finally, performing the attack via an externally connected device, for example, with a wirelessly enabled rogue node, possibly attached to the ODB port, would allow for further understanding of the risks from the bit rate attack method.

C.9 Discussion and attack mitigation

These bit rate attack experiments have confirmed another weakness of the CAN bus. CAN is susceptible to a DoS attack by a malicious device that is able to manipulate the bit rate of the packet transmissions. The simplicity of the CAN protocol is the root cause of the issue. At any given bit rate the CAN hardware is sampling the CAN signal to determine the bit values to decode into the data elements (see Table 2.2 in Chapter 2 for a summary of those elements). If CAN nodes are configured with different bit rates then their sampling frequencies will differ. For a given CAN signal the different sampling frequencies will result in different interpretations of the bitstream. The nodes configured with the same bit rate as the CAN bus bitstream will decode the data correctly. However, a mismatch between the configured bit rate and the bit rate of the CAN bus bitstream causes nodes to indicate errors with the transmitted data packets. At the bit level, too many errors will cause a CAN node to shut down, known as a *bus off* state [83].

A more concrete example is illustrated in Figure C.4. In this schematic, the lower bitstream is twice the bit rate of the upper bitstream, and the upper bitstream fits into the first half of the lower bitstream. In CAN, a high differential signal is a zero, therefore, the signal peaks are sampled as a binary 0, and binary 1 for the signal valleys. A node configured to match the upper bitstream's bit rate will interpret the signal values from the faster bit rate differently. This is the cause of protocol errors and, thus, nodes entering a bus off state. Getting a CAN node to enter this state has been used as an attack method by other researchers [88], [89], by directly manipulating the bits in the CAN protocol.

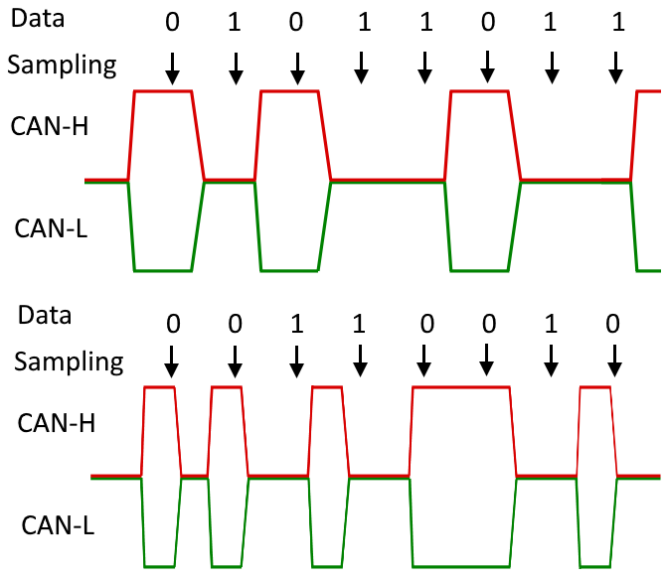


Fig. C.4 Schematic comparing bit rates and why it results in errors, for the same sampling frequency the faster signal results in a different interpreted bit values

At a more macro systems level, the CAN bus errors usually manifest within the vehicle as error messages, warning lights and diagnostic indicators on the vehicle's displays, or physical effects on vehicle controls and engine.

There are not likely to be many mitigations to the bit rate attack. This is due to it being an attack directly affecting the bit level physical characteristics of the data transmissions. Cryptography solutions to protect CAN [33], or a CAN Intrusion Detection Systems (IDS) [90] will not work because of the physical disabling (bus off) effect of the attack. For devices attached directly to the OBD port, a firewall or gateway device would be able to detect incorrect bit rates and not allow those signals to propagate through to the vehicle networks.

The emergence of alternative protocols, such as automotive Ethernet (100BASE-T1), as another in-vehicle communications bus is beneficial for future vehicle systems design. This is because vehicle systems that use automotive Ethernet would not be subjected to this attack because of its single transmission speed of 100 megabits per second (Mbps) and point-to-point topology.

C.10 Possible further work

These experiments were conducted to confirm observations during this research, and by others, on CAN bus failures due to incorrect bit rate settings. However, these experiments need to be expanded further too fully quantify the threat. There are several variables that affect how successful a particular CAN bit attack is at affecting a vehicle, these include:

1. CAN bus bit rate
2. Attacker bit rate
3. Attacker data packet transmission rate
4. The functionality of existing nodes
5. The functionality of the attacking node
6. The design of the in-vehicle systems

These variables need to be studied further in order to gather more data to fully quantify the threat, and aid methods to mitigate the attack. It will require further detailed experimental work to determine how the above variables individually affect the attack process. The experimental work can address the following:

- Perform the CAN bit rate attack against other targets, particularly vehicles, to study the variety of responses to the attack.
- An additional variation on the packet cycle time variable.

- Investigations to determine if the CAN bandwidth utilisation affects the efficiency of the attack. This would be to determine if a CAN bus with a higher or lower traffic load has an effect on executing the attack.
- Examine the attacks in detail at the hardware level to find protocol factors that determine attack success or failure.
- Demonstrations of weaponizing the attack, for example via an OBD interface, and wirelessly, to demonstrate the attack in action against a running vehicle in a safe environment.
- Examine how the attack could apply elsewhere, for example, CAN is used in other transportation systems, IoT and factories. Performing this attack on systems in other domains that also use CAN would enable further understanding of the generalised implications of the bit rate attack.

These experiments looked at standard CAN as this is the most widely used CAN protocol found within vehicles. Studying the effect of the configuration attack on the higher speed CAN with Flexible Data-rate (CAN FD) would be one extension to this work. It would likely yield interesting results due to the higher data transmission rates of CAN FD. These results could form a point of comparison with the current widespread use of standard CAN.

C.11 Conclusion

The examination of a bit rate attack against CAN was not part of the original research aims but a consequence of observations made during this research. However, it provided an initial use, and hence operational validation, of the testbed and the fuzzer tool. It was interesting to note that the bit rate attack had not been previously considered as a possible threat in the literature, despite acknowledgement by automotive engineers that bit rate configuration errors can affect CAN functionality.

In summary, the experiments performed do confirm that deliberate manipulation of the bit rate may disrupt a CAN bus and thus, its use as a possible attack against vehicle systems is feasible. For this attack, the security property compromised is *availability*, as it is a DoS attack.

It was important to examine the attack from an automotive cyber-security viewpoint, even though it was peripheral to the main research. The experiment provides a starting point for further investigations by researchers into the bit rate attack. The investigations can include weaponizing and packaging of the attack for real-world deployment, examining the detail of its operation at the bit level, protection and mitigation mechanisms to prevent such an attack, and use of the attack in other domains.

The experiment provided further evidence on the weakness of the CAN protocol, particularly in the age of the connected car, and thus the importance of security testing systems. This experiment further validated the testbed for automotive cyber-security testing, and provide a first application for the developed fuzzer, although limited to controlling the bit rate of the USB adaptor as the attack node and transmitting a single packet.

Appendix D

Image permissions

Just as iron rusts unless it is used, and water putrefies or, in cold, turns to ice, so our intellect spoils unless it is kept in use.

Leonardo da Vinci

Content removed on data protection grounds

Content removed on data protection grounds

Appendix E

Safety considerations

Engineering is too important to wait for science.

Benoit Mandelbrot

This research program was conducted in accordance with Coventry University's and HORIBA MIRA Ltd's ethical guidelines and health and safety policies. For readers of this thesis who are embarking on similar research it is important, for ethical and health and safety reasons, to understand the risks involved in working with vehicles and vehicle systems, particularly for those who are not familiar with automotive or electrical and electronic engineering.

This research required interaction with the electrical systems of vehicles and vehicle components that require voltages of 12 to 13 volts. It is important to be aware of safety issues when dealing with electrical voltages. If replicating the experiments in this research please be aware of the following considerations:

- Read the health and safety procedures of your organisation.
- Turn off power supplies, equipment and vehicles when they are not being used.
- Never leave the positive and negative and power leads touching.
- Always unplug power cabling when not in use.
- Ensure power supplies, equipment and vehicles are turned off prior to connecting cabling.
- Double-check all voltage connections to ensure components and equipment are safely connected.
- Never leave powered equipment or running vehicles unattended or in an unsafe state.
- Always check that voltage levels from power supplies are set correctly. Voltage levels are important, too high a voltage may damage a component, test equipment, an ECU, or cause a fire.

- If necessary seek help from qualified technicians.

Appendix F

Log File Searching

There's a humorous side to every situation.
The challenge is to find it.

George Carlin

The cyber-physical nature of the modern vehicle means that there is a transition from the digital signals to the physical world and vice versa. A CAN packet transmitted on the vehicle network will generally not result in a CAN packet in response. Instead, a physical action will occur. If a log file of CAN data is available for playback in a system it is possible to use it to find the CAN packet responsible for the physical action using a search. This was done with the display ECU in Chapter 9. A divided-by-half search is well understood, however, the following is a formal definition of the process used in this research.

1. The CAN data being played back by the fuzzer is taken from each recorded line in a log file. The data line being played back is given as L_x, where x is the x^{th} line in the log file. After the search, the packet stored at L_x should contain data that causes a reaction from the ECU.

2. The number of log file lines to playback is given as t, initially set to the total number of lines read from the log file.

3. The starting line for the playback, L_s, begins at the first line, $L_s = L_1$.

4. The number of CAN packets to playback, n, is set to half (rounded down) of the total lines, $n = \lfloor t/2 \rfloor$, this halving is the important search part.

5. The ending line for the play back, L_e, is set at the n^{th} line from, and including the starting line, $L_e = L_{s+n-1}$.

6. The set of packets being played back, S_p, is $S_p = \{L_s \ldots L_e\}$.

7. The fuzzer plays backs each packet, L_x, in set S_p.

8. If the ECU does not react to the playback then the starting and end lines are set to the next half, $L_s = L_{e+1}$ and $L_e = L_{e+t-n}$, return to step 6.

9. If a packet causing the ECU to respond is within the played back set, the search space is reduced to the count of the packets in the set, $t = |S_p|$.

10. If $t = 1$ then all packets have been played back and the search is stopped, jump to step 12.

11. The search returns to step 4.

12. The log file line L_x will (or should) have the CAN bus packet data that caused an ECU reaction.

Appendix G

Ethics documentation

And remember... don't be evil, and if you see something that you think isn't right – speak up!

Google Code of Conduct

This following pages contain the supporting document for the ethics process, reference number P6333.

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Content removed on data protection grounds

Appendix H

Additional information on the CAN fuzzer software implementation

Truth is ever to be found in simplicity, and not in the multiplicity and confusion of things.

Isaac Newton

This appendix provides additional information on the software that was written during the development of the CAN fuzzer tool. It adds to the high-level information provided in Chapter 5, which discussed the rationale for a new CAN fuzzer tool, its required characteristics, its interface to the CAN bus, the user interface, and its initial operational validation. Whilst the primary focus of the research was to apply fuzz testing to automotive systems, the CAN fuzzer was an output from the research and will be of interest to others working in the automotive systems field. This Appendix provides some additional information on accessing and obtaining the source code for the CAN fuzzer tool, it also includes an example of the log file output for the CAN fuzzer.

H.1 Limitations on access to the CAN fuzzer code

This research was conducted under a partnership agreement between HORIBA MIRA Limited and Coventry University (see the Acknowledgements at the beginning of the thesis). The agreement gives HORIBA MIRA the Intellectual Property Rights (IPR) to the output of the research, and Coventry University a no-cost licence to use the output. To protect their commercial IPR HORIBA MIRA have requested that information on the internal's of the CAN fuzzer software tool is not disclosed. Specifically, there is a restriction on the amount of detail that can be revealed about the software's source code and its operation. The outcome on a discussion to open source the entire project was made, the decision was for the project to remain close sourced, as confirmed in the email shown in listing H.1.

Listing H.1 Email confirmation on keeping the CAN fuzzer tool closed source

```
From: Paul Wooderson <paul.wooderson@horiba-mira.com>
Sent: 08 August 2019 12:38
To: Fowler, Dan <fowlerd3@coventry.ac.uk>
Subject: RE: The CAN fuzzer

Hi Dan,

We've discussed here and our preference is to keep it closed source
for the time being, although keeping an open mind for the future.

Best regards,
Paul
```

The IPR restrictions have prevented the source code from being included, and only the descriptive overviews in Chapter 5 and this appendix are provided. However, two components have been made open source as they are not covered by the IPR. Links to a publicly available source code repository, on GitHub, [1] are provided for those two components in Section H.3.

H.2 The code's project file and components

The project file for the CAN fuzzer tool is shown opened in the Microsoft Visual Studio environment in Figure H.1, the main start-up and UI project, `FuzzMainMDI`, is highlighted.

The `Program.cs` file contains a stub class that is called by Windows to start the CAN fuzzer and load the main UI interface (`FrmFuzzMDI`). In the main project each C# class with a name beginning **Frm**, for Form, is a UI component.

- `FrmInterfaces` and `FrmCANInterface` are used to select, start, stop and configure the CAN bus interfaces and data packet logging.
- `FrmFuzzTest`, `FrmFuzzFlood` and `FrmRunFuzzer` configure the parameters for the CAN fuzz testing and executes the testing.
- `FrmFuzzMDI` is the UI container for all other interfaces.
- `FrmMonitor` reads data packets from the CAN bus and then logs data that matches defined criteria.
- `FrmSingleShot` allows the tester to send a single packet of a specific format to the CAN bus, on a button press or at a regular interval.
- `FrmStatus` displays status and error messages generated by the tool.

[1] https://github.com/

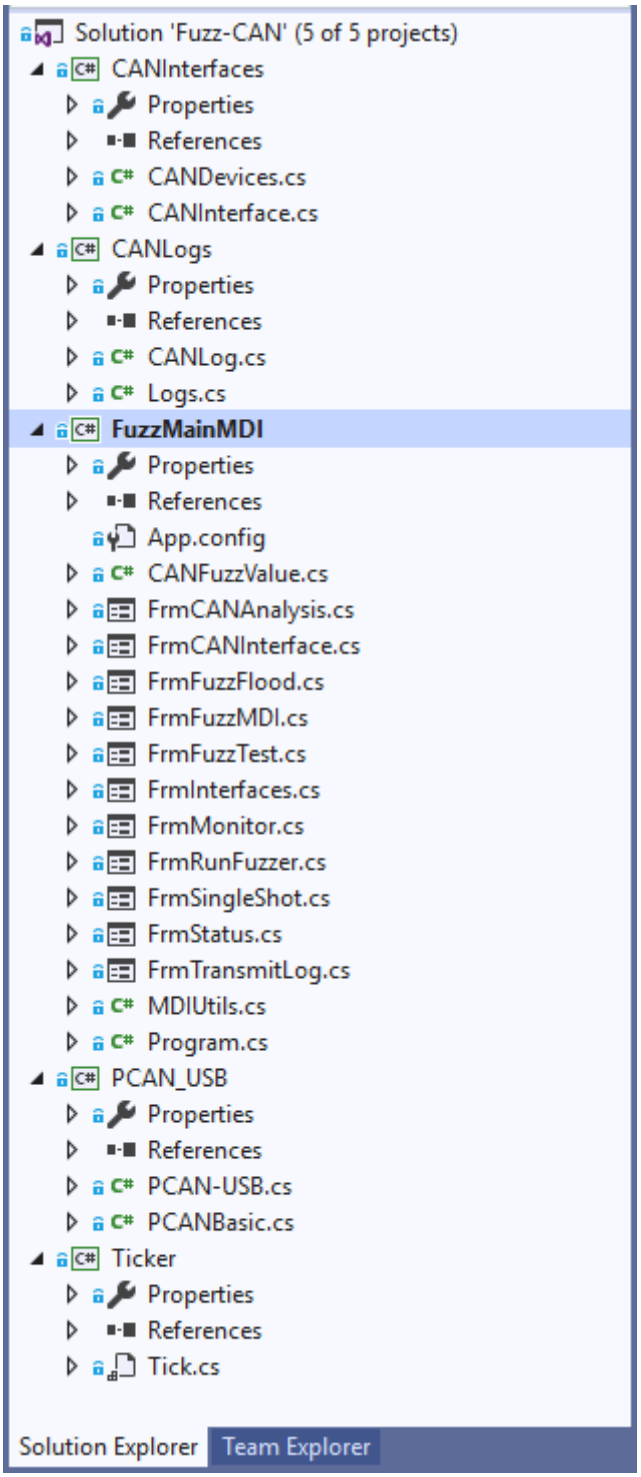


Fig. H.1 The CAN fuzzer software project in the Microsoft Visual Studio IDE, the main program and UI interfaces project is highlighted

- `FrmTransmitLog` allows for a previously recorded log file (see listing H.2 for an example) to be sent to the CAN bus.

The remaining class files in the project are used to store CAN data packet fuzz testing values, CANFuzzValue.cs, and provide some UI utility functions, MDIUtils.cs. The main project is supported by four C# Windows Dynamic Link Library (DLL) components.

- The `CANInterfaces` library contains the definitions and configuration for each CAN bus interface being used, it exposes a reference for each defined interface that can be used by each UI component.
- The `CANLogs` library contains the functionality to handle CAN bus log files and provides a reference to be used by other components to access a log file .
- The `PCAN_USB` library is the code that is used to talk to the CAN to USB interface adaptor. This is one of two open source components used with the project, see Section H.3.2.
- The `Ticker` library is the second open source component in the project. It is used to provide a timing signal for CAN data packet transmission, see Section H.3.1.

H.3 Open source components

The CAN fuzzer tool has two software components that have been made open source. Firstly, the Ticker component is a novel one millisecond timer for Windows programs. Whilst code to wrap the Windows high-resolution timer is commonly available, the implementation here has the advantage of being very simple and short. Secondly, the code to interface to the PCAN-USB interface device is based upon its device driver interface sample code. That code was extended into a useful general-purpose library for Windows C# programs, and not being core to the logic of the CAN fuzzer it was made open source.

H.3.1 Ticker is a one millisecond timer

One of the first restrictions encountered when developing the CAN fuzzer was the default timer control provided by the programming framework. The general timer has a minimum interval of 15 milliseconds. To get an interval timer below 15ms the Windows high-resolution timer must be used. Example code to use the high-resolution timer from C# is available by doing an Internet search, however, it often relies on complex interaction with the Windows API. A simpler solution was implemented using the programming frameworks *Stopwatch* component and a component to handle a background thread called a *Background worker*. The final solution for a 1ms timer is a small standalone class file that is used by the software to control CAN data packet transmission, see listing H.3 for the class `Tick.cs`.

Listing H.2 Example log file captured by the CAN fuzzer, the fuzzer's output begins at message 117

```
;$FILEVERSION=1.1
;$STARTTIME=43497.4380077083
;
;    C:\Fuzz-message.trc
;
;    Start time: 01/02/2019 10:30:43.866.0
;
;    Message Number
;    |          Time Offset (ms)
;    |          |         Type
;    |          |         |         ID (hex)
;    |          |         |         |      Data Length
;    |          |         |         |      |    Data Bytes (hex) ...
;    |          |         |         |      |    |
;----+--   -----+-----  --+--  -----+---  +  -+ -- -- -- -- -- -- --
      1)       6372.8  Rx          055B   8   5B 01 00 00 00 00 00 00
      2)       6472.9  Rx          055B   8   5B 02 00 00 00 00 00 00
       .
       .
       .
    106)      11509.7  Rx          0290   8   00 7E 00 00 00 00 00 00
    107)      11510.0  Rx          02A3   8   00 00 00 70 00 00 00 10
    108)      11510.3  Rx          02E3   8   00 00 00 00 00 00 00 00
    109)      11510.6  Rx          0321   8   00 00 00 00 00 00 00 00
    110)      11510.9  Rx          0322   8   00 04 00 01 00 00 00 00
    111)      11511.2  Rx          04D3   8   31 50 54 28 00 00 00 00
    112)      11709.1  Rx          0287   8   00 00 00 00 00 00 00 00
    113)      11709.4  Rx          0290   8   00 7E 00 00 00 00 00 00
    114)      11812.9  Rx          055B   8   5B 14 00 00 00 00 00 00
    115)      11908.9  Rx          0287   8   00 00 00 00 00 00 00 00
    116)      11909.2  Rx          0290   8   00 7E 00 00 00 00 00 00
    117)      96882.9  Rx          03EE   8   44 3B 99 FF 5E 7F C6 97
    118)      96883.1  Rx          01C1   8   5E 19 A5 96 56 2F 81 C6
    119)      96884.6  Rx          0270   8   BD 9F 42 9D D1 10 0D 35
    120)      96886.3  Rx          0723   8   94 7F 0C C1 0D CA 2C 36
    121)      96887.5  Rx          03EC   8   90 D4 11 CA C9 16 BA 1C
    122)      96889.4  Rx          0169   8   5E 48 AC 5A 21 A5 F8 FC
    123)      96890.5  Rx          0174   8   71 91 B2 2E 6B DE 0D E3
    124)      96892.6  Rx          0448   8   02 4A 49 71 92 E8 8D 78
    125)      96894.4  Rx          06A0   8   56 F1 07 40 AA 4C D2 B1
    126)      96896.4  Rx          0503   8   D1 30 08 2F 6B FF 22 E2
    127)      96897.5  Rx          02F9   8   08 D7 93 A1 A0 13 FF 3B
    128)      96899.4  Rx          0642   8   18 80 27 74 F7 80 89 72
    129)      96899.7  Rx          055B   8   5B 01 00 00 00 00 00 00
    130)      96901.3  Rx          0421   8   0C 5C 43 21 B9 D2 42 19
    131)      96902.5  Rx          0031   8   C6 AA 19 E9 B1 7F 5D 66
    132)      96904.3  Rx          04FD   8   4C 96 E9 67 4F 51 08 94
    133)      96906.4  Rx          01BE   8   76 AF 5E 40 DE AC 94 72
    134)      96907.5  Rx          0741   8   D1 E5 35 35 71 6D 67 3E
    135)      96909.3  Rx          0612   8   22 22 D2 ED D2 75 73 DB
    136)      96910.5  Rx          068D   8   C9 AE E1 1B 1B 16 4C D9
    137)      96912.3  Rx          005C   8   EB EB 7D D1 C5 67 61 B3
```

Listing H.3 Tick.cs implements a 1ms timer

```
using System.ComponentModel;
using System.Diagnostics;
using System.Threading;

namespace Ticker {
    public class Tick : BackgroundWorker {
        Stopwatch sw = new Stopwatch();
        bool ticking;
        public bool Ticking {
            get { return ticking; }
        }
        public int Milliseconds { get; set; } = 100;
        public void Start() {
            if (IsBusy != true) {
                RunWorkerAsync();
            }
        }
        public void Stop() {
            ticking = false;
        }
        public Tick(ProgressChangedEventHandler CallOnTick) {
            WorkerReportsProgress = true;
            ProgressChanged += CallOnTick;
            DoWork += RunStopwatch;
        }
        private void RunStopwatch(object sender, DoWorkEventArgs e) {
            BackgroundWorker worker = sender as BackgroundWorker;

            long diff = 0;
            long nextTrigger = 0;
            int tick = 0;   //toggles from 1 to 0 and sent back via progress

            sw.Start();
            ticking = sw.IsRunning;
            do {
                nextTrigger = sw.ElapsedMilliseconds + Milliseconds;
                do {
                    Thread.Sleep(1);     //or Thread.Sleep(0)
                    diff = nextTrigger - sw.ElapsedMilliseconds;
                } while (diff > 0);
                //Display tick
                worker.ReportProgress(tick);
                if (tick == 0)
                    tick = 1;
                else
                    tick = 0;
            } while (ticking);
            sw.Stop();
            sw.Reset();
        }
    }
}
```

The `Tick` class extends the `BackgrounderWorker` class, of which an instance is declared when `RunStopWatch` is called. The stopwatch checks to see if the correct number of milliseconds has lapsed (default 100), and if they have, an event is raised to the main program via the call to `ReportProgress`. The use of `BackgroundWorker` prevents the Tick class from blocking the main UI thread.

The `Ticker` DLL, along with a demonstration program is available as an open source project that extends the timing capability from 1ms into the tens of microseconds. It can be found at **https://github.com/GR8DAN/C-Sharp-Microtimer**.

H.3.2 PCAN_USB is a CAN to USB interface library

The interface from the computer to the CAN bus is via the PCAN-USB device manufactured by PEAK-System Technik GmbH. When the device is supplied it comes with example software, `PCANBasic.cs`, to interface to the device driver DLL. The example software has been used in the `PCAN_USB` component DLL to allow the CAN fuzzer to communicate with multiple PCAN-USB devices. A program was developed to test and verify the operation and functionality of this DLL. The source code for the DLL is too large to include in the Appendix, however, it can be viewed on Github, along with the test program which doubles up as a demonstration of `PCAN_USB`. The open source project is located at **https://github.com/GR8DAN/PCAN_For_USB**.