A Method for Constructing Automotive Cybersecurity Tests, a CAN Fuzz Testing Example

Fowler, DS, Bryans, J, Cheah, M, Wooderson, P & Shaikh, S

DOI 10.1109/QRS-C.2019.00015

Publisher: IEEE

A Method for Constructing Automotive Cybersecurity Tests, a CAN Fuzz Testing Example

Daniel S. Fowler
Systems Security Group
Coventry University
Coventry, UK
fowlerd3@coventry.ac.uk

Jeremy Bryans
Systems Security Group
Coventry University
Coventry, UK
jeremy.bryans@coventry.ac.uk

Madeline Cheah
Horizon Scanning
HORIBA MIRA Limited
Nuneaton, UK
madeline.cheah@horiba-mira.com

Paul Wooderson
Vehicle Resilience Technology Centre
HORIBA MIRA Limited
Nuneaton, UK
paul.wooderson@horiba-mira.com

Siraj A. Shaikh
Systems Security Group
Coventry University
Coventry, UK
siraj.shaikh@coventry.ac.uk

***Abstract*—There is a need for new tools and techniques to aid automotive engineers performing cybersecurity testing on connected car systems. This is in order to support the principle of secure-by-design. Our research has produced a method to construct useful automotive security tooling and tests. It has been used to implement Controller Area Network (CAN) fuzz testing (a dynamic security test) via a prototype CAN fuzzer. The black-box fuzz testing of a laboratory vehicle's display ECU demonstrates the value of a fuzzer in the automotive field, revealing bugs in the ECU software, and weaknesses in the vehicle's systems design.**

***Index Terms*—cybersecurity testing, controller area network, fuzz testing, automotive engineering, systems security, embedded systems, dynamic software testing, black-box testing, SAE J3061**

I. Introduction

At Black Hat 2015, the premier convention for hackers, security researchers Miller and Valasek showed the first remote exploitation and control of an unmolested mass production car [1]. Their research provided a demonstration of cyber-physical hacking that could potentially cause harm to the vehicle's occupants, other road users, and pedestrians. Their widely reported research once again showed that any type of device that has connectivity could become a target for a cyber-attack and highlighted the potentially severe consequences of a compromised connected car. If car manufacturers were not already addressing the cybersecurity of their products, then their research showed the importance of doing so.

The automotive industry is responding to the cybersecurity threat to their products. In the SAE J3061 guidelines [2] the use of Threat Analysis and Risk Assessment (TARA) and designing security into systems is advocated. The goal of *secure-by-design* [2]–[4] is to raise assurance levels [4], [5] in the software and hardware within vehicle systems. High assurance is important in order to maintain confidence [5] in a connected car's operation and safety.

Jointly funded by Coventry University and HORIBA MIRA Limited.

The secure-by-design goal is verified with security testing [2], [4]. One of the goals of our research is to examine how a class of security testing, called fuzz testing, can have an impact on the automotive systems engineering field. Fuzz testing is a successful dynamic security testing technique used in traditional information systems [7]. However, there is a scarcity of detailed published works in the application of fuzz testing to vehicle systems [6].

Within vehicle systems several types of data networks are used, here, the Controller Area Network (CAN) is chosen as the target for fuzz testing, for several reasons [6]:

- it is a commonly used in-vehicle network;
- many of the computers in a vehicle have a CAN interface;
- the design of CAN makes it very susceptible to attack;
- tools and techniques for manipulating the CAN protocol are inexpensive and readily available.

In our research we examine how a new, easy to use, and easy to install prototype automotive fuzz testing tool, a CAN *fuzzer*, can be used to test the security properties of a vehicle's computers.

An in-vehicle computer is commonly called an Electronic Control Unit (ECU). In our research a laboratory vehicle's display ECU is used as the Target of Evaluation (ToE) [8], a.k.a. the system-under-test. The results from the fuzz testing found weaknesses in the display ECU and the associated lab vehicle's systems design. Additional engineering to address the discovered weaknesses would improve the assurance of the target vehicle against CAN based attacks.

The work presented here builds upon our previously published work, see Section II. In Section III a brief refresher on the CAN protocol is given. Section IV adds our methodology for constructing automotive security tests and its application to CAN fuzz testing. The new results from fuzz testing a display ECU are covered in Section V, this is followed by a discussion in Section VI and a conclusion in Section VII.

II. Previous work

In our previous work [6] we examined the use of fuzz testing within the automotive field. Fuzz testing is used by security researchers, software testers and attackers. The aim of fuzz testing is to discover software and hardware issues that may then be used to compromise a system's security properties, its confidentiality, integrity, and availability.

Fuzz testing is a dynamic software stress test (equivalent to injecting noise into computer electronics when performing hardware fault injection tests):

1) High volumes of random or malformed data are sent into the system's interfaces by a fuzzer.
2) The system's reaction to the fuzz testing is monitored.
3) Conditions at a point of failure or interest are recorded for analysis.
4) Fuzz testing is highly automated for efficiency, and to cover a wide value space.
5) Analysis of fuzz testing results may reveal a system's weaknesses.
6) An attacker will use weaknesses to try and compromise a system. A software tester will use the results to improve system design, and hence a system's assurance levels.

Automotive fuzzers are summarised in Table I. The beStorm [9] and Defensics [10], [11] fuzzers are general purpose commercial products. The booFuzz [12] fuzzer is open source but was configured to work with commercial software. The Peach fuzzer is advertised as supporting automotive use, but there are no published results or use cases available. There is an open source version of Peach available. Most of the approaches to fuzz testing are based upon protocol information, i.e. based upon knowledge derived from network or interface specifications. For automotive CAN this means using the format of the CAN data packets, see Section III. This is common for black box testing, performed in our research, where little is known about the system internals. However, a white box approach can also be used, for example, if there is pre-existing knowledge of CAN data packet contents acquired from source code running in an ECU. The complex features, functionalities, and algorithms of the existing commercial fuzzers will the subject of further research.

TABLE I
Automotive fuzzers

Fuzzer	License	Approach
beStorm [9]	Commercial	Protocol based
Defensics [10], [11]	Commercial	Protocol based
booFuzz [12]	Mixed	Design based
Peach (www.peach.tech)	Mixed	Protocol based
Prototype CAN Fuzzer [6]	Commercial	Protocol based

The addition to Table I from its previous publication is the interesting solution to the oracle problem provided by [11]. They propose fuzz testing a ToE alongside an identical system (not subjected to the fuzz testing) as a reference point, and comparing the output from the fuzzed ToE to the reference system. This proof-of-concept system uses two model cars, one as the ToE and one as the reference, with the models' systems controlled by CAN busses. A dSPACE Hardware-in-loop (HIL) system is linked to the Defensics fuzzer. (HIL, software-in-the-loop, and model-in-the-loop, are methods used to emulate systems and components during automotive development.) The dSPACE HIL can load, start and stop the fuzzer. When the fuzzer is active the measured analog and digital signals from the two models are compared and any differences are logged as errors.

Information on the performance of the system would be useful, for example, does the analysis of the outputs keep up with system operation? Further, there is no discussion of the acceptable variation between the ToE and the reference system. How much of a difference between the output of the fuzz tested ToE and the reference system is required before an error condition is considered? An interesting point is made about the reference systems. If a model of the system can run on the HIL, and the model is accurate, then the reference system could be virtual, used within the HIL system itself.

As we have previously noted in our own research [6], they also mention the lack of available knowledge on automotive fuzz testing. Most publications are overview experiences of implementing a commercial fuzzer, rather than detailed data on the effectiveness of the fuzz testing, the practical experimental methods used, and lessons to be 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 does not only apply to CAN. There are many technologies in use in a vehicle, the large attack surface provides plenty of scope for more research into applying fuzz testing to vehicle systems. This motivates our research to address the gaps.

III. The CAN bus

There is plenty of publicly available information on the technical details of the CAN bus, the following is a brief overview.

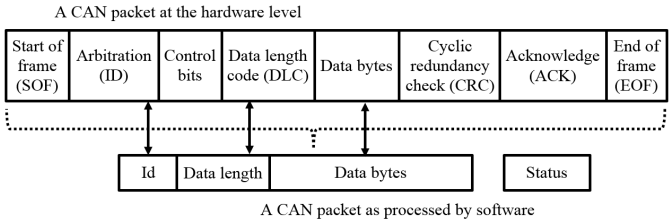

Fig. 1. Overview of a CAN data packet. The software in an ECU or fuzzer sees the CAN data as an id, data length, data byte values and status condition. The CAN network hardware handles the control bits during data transmission and reception.

CAN uses twisted-pair wires to provide a data network between ECUs in a car. The transmission speed of the standard CAN protocol is modest, designed to support up to 1Mb/s. A simplified CAN data packet is shown in Fig. 1. The CAN hardware in an ECU handles the details of the data protocol, including error handling. The ECU software views the CAN

TABLE II
EXAMPLES OF CAN PACKETS LOGGED FROM A CAR

Time (ms)	Id	Length	Data
5328.009	043A	8	1C 21 17 71 17 71 FF FF
5329.008	0296	8	00 00 00 00 00 00 00 60
5331.029	04F2	8	00 53 6C 00 00 00 00 00
5338.165	0215	7	00 1C 01 00 00 01 40

packet as an identifier, data length, payload bytes and network status information. Any ECU can initiate data transmission, however, only one ECU at a time is allowed to transmit. For a CAN packet the arbitration identifier (id), determines transmission priority. The lowest id continues transmission in the event of packet collision (two or more ECUs attempting simultaneous transmission). Examples of CAN packets logged from the lab vehicle are shown in Table II. CAN was designed prior to vehicle connectivity and without consideration for cybersecurity. It has vulnerabilities and is susceptible to several types of attack [15], including packet sniffing and replay, injecting false values, denial of service via flooding, jamming (known as bus-off) and spoofing data from ECUs.

IV. A METHOD FOR CONSTRUCTING AUTOMOTIVE SECURITY TESTS

Our research has identified seven phases in the development of an automotive security testing technique, shown in Fig. 2.

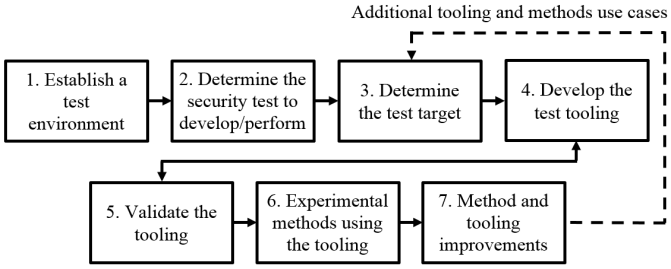

Fig. 2. Systematically constructing automotive security tests.

1) **Establishment and validation of a test environment** - The complexity and costs of the modern vehicle provides a challenging environment in which to develop new security testing methods. For a new vehicle design the access to the car's internal systems and components may 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 security testing methods. In our research a commercial vehicle design and development system is deployed as a security testing rig. Such systems are traditionally used for functional testing, but our research [14] confirms they can be used for the development of security tooling, a similar approach taken by other researchers [11]–[13].
2) **Determine a security test to develop or perform** - SAE J3061 identifies three classes of security test that can be executed against a vehicle system or component. 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 WiFi interface would differ in its details to one targeted at Bluetooth.
3) **Determine the technology/ToE to test** - Vehicles contain multiple digital technologies. There are many types of components, sensors, interconnections, interfaces, and communication protocols. For this research CAN was chosen for its ubiquity within vehicle systems. However, many vehicle technologies 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 systems this invariably means some form of software, but may include interfacing and measurement equipment. The software and equipment can be commercially sourced, or may require bespoke design and development, or both. This research required a simple CAN fuzzer. Simplicity here is in terms of ease of use and deployment. Our previous work [6] discusses the development of the prototype CAN fuzzer. 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 specialised tooling for other vehicle technologies will be required.
5) **Validation of the tooling** - Any security test tooling will need to be performant for its intended application. Therefore, it needs to be validated. In this research validation of the CAN fuzzer was performed during its development. A similar validation phase is required for security tooling used against other technologies. Issues with validation results in further tooling development.
6) **Experimental methods using the tooling** - In this phase the established test environment and the new test tooling is used, performing the determined security test and recording the results. 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. 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** - The final 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 Fig. 2. The near continuous development of new and improved technologies in the automotive field will, likewise, require continuous im-

provements in security testing tooling and techniques.

The method, summarised in Fig. 2, for the construction of automotive security tests and tooling, was derived from our work on producing a prototype CAN fuzzer tool. The method applied to CAN fuzz testing is summarised in Figure 3.

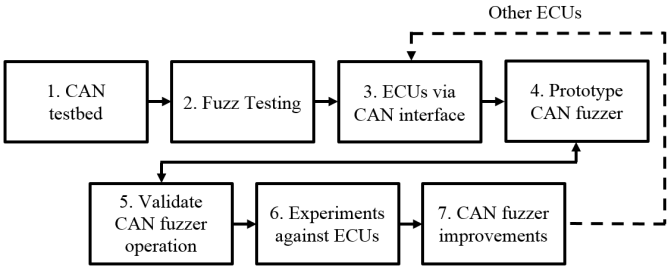

Fig. 3. CAN fuzz testing development method.

The implemented prototype CAN fuzzer is a simple to install Windows PC application. The fuzzer communicates with CAN using a PEAK System[1] USB to CAN interface. No other expensive software, special packages, or specific versions of libraries are required. It has several Graphical User Interface (GUI) screens for configuring the CAN interfaces, fuzz testing and logging. One of the screens in shown in Fig. 4.

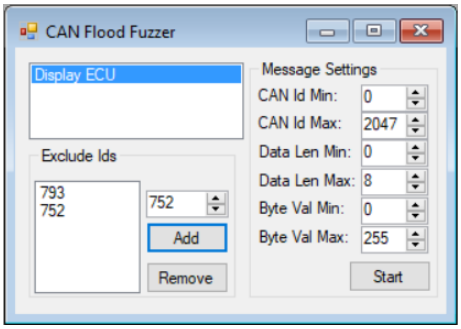

Fig. 4. One of the prototype CAN fuzzer's configuration screens.

Having used the prototype CAN fuzzer against ToEs in our previous work, here it was deployed against a vehicle's display ECU. As well as allowing for an examination of the ECUs security properties (Section II), it provides for further refinement of the fuzzer, i.e. stage 7. of the method.

Aside from the display ECU having a CAN interface, its screen was a factor in its choice as a ToE. It had been noted in our research that a problem with fuzz testing vehicle systems is the cyber-physical aspect – the software side of a ToE may induce changes on the physical side. A CAN data packet may not cause a detectable reaction from an ECU via a CAN bus reply. This is because the CAN packet is triggering an external, real world response. Experimenting on an ECU with a built-in real world, and visible, aspect allows for testing that is not reliant upon other vehicle parts, such as sensors, actuators or lights.

[1]https://www.peak-system.com/

V. Results from fuzz testing a display ECU

The target Display Interface (DI) ECU operating in the lab vehicle is shown in Fig. 5. It's position within the vehicles CAN networks is illustrated in Fig. 6, derived from the vehicle electrical service manual. The functions of the vehicle's ECUs are listed in Table III.

Fig. 5. The Display Interface (DI) ECU in the laboratory vehicle, a variety of messages are displayed in response to occupants operating the vehicle.

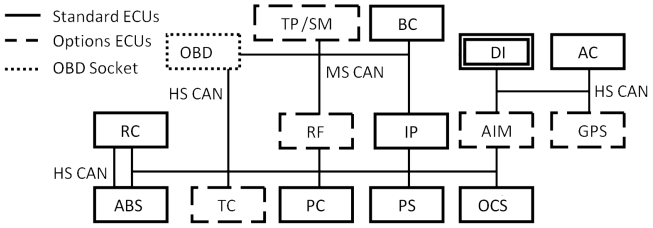

Fig. 6. Networks connecting ECUs in the lab vehicle, 3 High Speed (HS) CAN busses run at 500Kbps, and one Medium Speed (MS) CAN at 125Kbps.

TABLE III
ECUs in a small vehicle

ECU Reference	ECU Function[a]
OBD	On-Board Diagnostics connector
IP	Instrument panel
PS	Power steering control
AIM	Audio Interface Module
OCS	Occupant (Passenger) sensing
RF	Key fob functions
RC	Restraint control
ABS	Anti-lock brake system
TC	Transmission control
PC	Powertrain control
TP/SM	Tire pressure and security module
BC	Body control
DI	Display interface
GPS	Global positioning system
AC	Audio control

[a]Abbreviations changed for commercial confidentiality.

In our experience fuzz testing may damage components and so another display ECU was purchased for the experiment. The purchased ECU does not have the full color display but functions correctly within the vehicle, see Fig. 7. The ECU's internal CAN connection was confirmed to run at 500Kbps, and the observed CAN packets have an eight-byte payload.

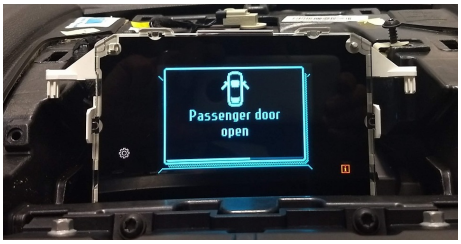

Fig. 7. The ECU obtained for bench testing operating in the lab vehicle. Despite ordering the correct used unit, it is a lower specification component (there is a small difference in the part numbers), it does functions correctly.

For the bench-based fuzz testing, custom cabling, correctly terminated[2], was used to connect to the PC based CAN fuzzer. A bench supply, to power the ECU, replicated the 12.4 volts that was measured at the vehicle.

The fuzzer was configured to randomise the standard CAN id range from `0` to `2047`. At the fuzzer's default transmission rate of one packet per millisecond, it would take less than three seconds to run through the standard range of id values ($2048 \cdot 0.001s = 2.048s$). The CAN packet payload was configured to be fixed at eight bytes, as observed on the car's CAN bus. The byte values were randomised over their full range of `0` to `255`.

It was observed that the display ECU, as with other ECUs that have been tested, enters a standby mode when no CAN communications are present on the CAN bus. This can cause the CAN fuzzer to report communications errors. This is because the initial CAN packet transmission from the fuzzer does not get acknowledged, due to the ECU waking from its standby mode. If the ECU does not wake up quickly enough then the fuzzer's CAN interface can enter what is known as a *bus-off* state. To overcome this problem a second CAN interface is configured within the CAN fuzzer and attached to the CAN bus. This second interface allows for the initial CAN packets from the fuzzer to be acknowledged, while the ECU awakens from standby. This prevents the possible bus-off state.

When power is applied to the ECU the screen is blank. The fuzz testing is started. After a short period (10s of seconds) the screen flashed a message (*Park brake applied*), see Fig. 8. This physical response to the fuzz testing demonstrated that the fuzzer had generated a packet that the ECU is programmed to process.

The use of the fuzzer to provoke a response from the ECU allowed for the opportunity to use the results for reverse engineering. Knowledge on the internal operation of vehicle systems and components is difficult to obtain for researchers (and attackers), 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.

[2]https://tekeye.uk/automotive/can-bus-cable-wiring

Fig. 8. Displayed message during fuzz testing.

- Functional safety engineers, to understand the operation of vehicle systems.
- Security engineers, to use system operational knowledge to aid penetration and vulnerability testing.
- Attackers who have an interest in compromising vehicle systems.

Having the log file from the fuzz testing, and knowing the ECU responded to the testing, it allowed for the CAN data that caused the response to be determined. 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 the packet that invokes an ECU reaction.

A. Binary search for a CAN packet

The fuzzer has a log file playback function. The 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 file was played back. This was a *binary search* for the packet affecting the ECU. This binary search worked until the last four lines of the log file remained, at which point, the display of the message was inconsistent.

Several attempts at playing back the last few log file search lines would, or would not, cause a message to be displayed on the ECU's screen. This made it difficult to zero in on the CAN packet responsible. One cause is the ECU entering the standby mode prior to a play back. Another cause is the rate of packet transmission. The ECU would ignore packets if they are sent at too high a rate. However, using the fuzzer's single shot mode to send each of the remaining search packets, a CAN packet with the id of `793` was found responsible for displaying the seen message.

B. Determining display ECU's functionality

Having found the packet that causes the ECU to react, the next test was to determine what functionality the individual

bytes in the CAN packet perform. The identified packet has eight bytes of data. Testing the effect of the byte values can be done in 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.
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.

For 2) and 3) there is a combinatorial explosion problem. For an eight byte data packet, with eight bits per byte, the 64 bits give 2^{64} value combinations, or nearly 18.5 Exa ($Exa = 10^{18}$) possible values. Thus, methods to simplify the search of the value space for the meaning of a CAN packet's byte values are required.

One method is to view the packet contents as individual bit flags. This vastly reduces possible combinations by testing each bit individually. This bit setting was chosen for its simplicity, testing the 64 packet bits using this method:

1) At the start, data bytes in the CAN packet are zeroed.
2) The CAN packet is transmitted to the display ECU.
3) The display ECU's screen 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.

To aid with the single bit testing the prototype CAN fuzzer's single shot functionality was modified to make it easy to increment the byte values, see Fig. 9.

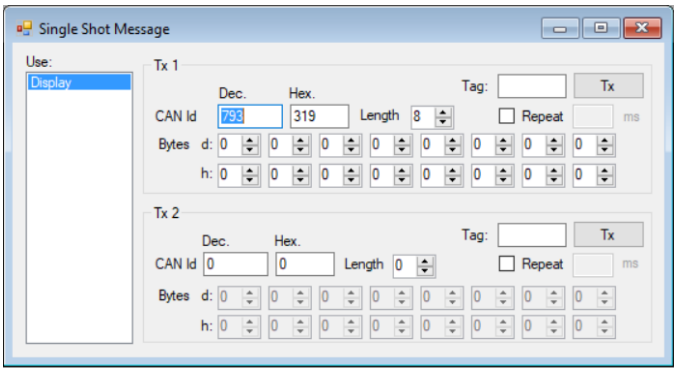

Fig. 9. The CAN fuzzer's single shot packet functionality was modified to aid with testing the display ECU. The arrows next to each byte increment or decrement the value by one (values can be entered directly).

The setting of individual bits within the CAN packet (id 793) did result in messages being displayed. The results for the first two bytes are shown in Table IV, space restrictions prevent all the results being included.

The messages found were compared to the messages listed in the user manual for the car, available from the manufacturer's customer website (the reference has been redacted due to commercial disclosure reasons). The user manual lists 78 possible messages that the display may show. Some of the possible messages relate to options and equipment that may not be present due to the vehicle model.

TABLE IV
DISPLAY ECU CAN PACKET ID 793 BIT SETTING RESULTS FOR BYTES 1 AND 2, ALL THE OTHER 5 BYTES WERE SET TO ZERO

Byte 1	Byte 2	Message
0000 0000	0000 0000	none (blank screen)
0000 0001	0000 0000	Press brake and clutch to start
0000 0010	0000 0000	Press brake to start
0000 0100	0000 0000	Press clutch to start
0000 1000	0000 0000	Active City Stop Auto braking
0001 0000	0000 0000	Active City Stop not available
0010 0000	0000 0000	Active City Stop Sensor blocked Cle…
0100 0000	0000 0000	Check tyre pressures
1000 0000	0000 0000	Tyre pressure sys malfunction Servic…
0000 0000	0000 0001	Engine on OK (not in manual)
0000 0000	0000 0010	MyKey active Drive safely
0000 0000	0000 0100	MyKey Speed limited to 160 km/h)
0000 0000	0000 1000	MyKey vehicle at top speed
0000 0000	0001 0000	none
0000 0000	0010 0000	none
0000 0000	0100 0000	none
0000 0000	1000 0000	none

For packet 793, out of the total 64 bit positions, there are 22 bits that result in a message being displayed. However, the message *Park brake applied* was seen twice. Furthermore, two messages are not present in the user manual (*Engine on OK* and *Selector lever unlocked*). Therefore, the bit testing for CAN id 793 revealed 20 of the 78 messages listed in the manual.

The CAN fuzz testing, packet discovery, binary search and bit setting was repeated to find another CAN packet, id 752, that controls the display ECU. For the second round of testing the previously found id, 793 was excluded from the fuzz testing, by entering it into an exclusion list in the fuzzer (see Fig. 4). Fig. 10 shows one of the messages displayed by packet 752.

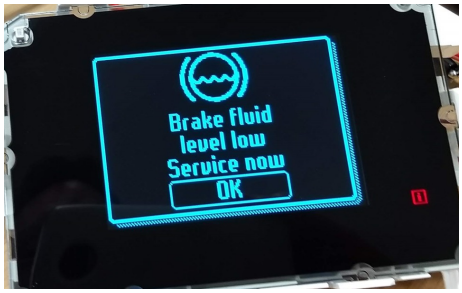

Fig. 10. The CAN data sent to the display ECU is manipulated to reverse engineer the ECU's functionality.

For packet 752, out of a total of 64 bit positions, there are 55 bits that result in a message. 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 a bit in different bytes. One message is not listed in the user manual, whilst in

three messages the text is not identical to the messages in the user manual. Two messages are repeated by several bits.

Taking the results from the bit testing, and excluding repeated messages, and the one unlisted message, then at least 44 messages in the user manual are displayed via CAN packet 752. This means that CAN packets 752 and 793 can display 64 of the 78 messages listed in the user manual, suggesting that one or more additional packets could be responsible for the remaining messages, see the Discussion in Section VI.

C. Varying CAN packet data byte length

The technique to discover the display ECU's functionality was proving successful. One variable that had not been altered was the length of the payload data. This was tried against the two discovered packet ids, 793 and 752. The following method is used to test the variation in the data length.

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 the maximum of 8 to the minimum of 0.
3) All the data values were fixed at zero. These Values were known not to cause the display of a message.
4) The display ECU was observed for a reaction when the configured CAN packets were transmitted.

Varying the data length had an unexpected effect, the results are shown in Table V. 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. However, reducing the payload data length does result in messages being displayed, even though the bytes values are all zero. As the number of bytes in the payload decreases the messages beginning displayed changed, in some cases several messages are seen. In Table V the values that would have resulted in the seen messages have been entered in bold red and underscored, with the actual zeroed and sent data entered in blue. What the results indicate is that the display ECU's program always reads eight full bytes from a data buffer. This appears to be a classic buffer overflow error. Such a finding, if found prior to production, would require a bug report to be raised. The software in the ECU should ignore any message not of the correct length.

VI. Discussion

On first appearances the display ECU appears to be simple in operation, a screen for communicating status messages to the vehicle occupants. However, there is a degree of complexity. It was observed that packet transmission rates influence its operation. Furthermore, the vehicle's user manual shows 78 possible messages, but 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). Experimental use of the CAN fuzzer was able to reveal the data required to display the messages, and other operational characteristics that would have otherwise remained hidden, this could be useful to an attacker.

TABLE V
Unexpected messages were displayed when the CAN packet data length was decreased from the observed 8 bytes.

Id	Data length	Msgs.	Sent vs decoded data (hex)
793	8	0	00 00 00 00 00 00 00 00
793	7	1	00 00 00 00 00 00 00 **80**
793	6	1	00 00 00 00 00 00 **01 00**
793	5	1	00 00 00 00 00 **00 01 00**
793	4	2	00 00 00 00 **00 80 01 00**
793	3	2	00 00 00 **00 01 40 00 00**
793	2	1	00 00 **00 01 00 00 00 00**
793	1	2	00 **09 00 00 00 00 00 00**
793	0	3	**90 01 00 00 00 00 00 00**
752	8	0	00 00 00 00 00 00 00 00
752	7	1	00 00 00 00 00 00 00 **0F**
752	6	1	00 00 00 00 00 00 **00 0F**
752	5	1	00 00 00 00 00 **01 00 00**
752	4	3	00 00 00 00 **00 18 40 00**
752	3	2	00 00 00 **40 00 01 00 00**
752	2	3	00 00 **01 48 00 00 00 00**
752	1	2	00 **04 00 20 00 00 00 00**
752	0	5	**06 06 00 20 00 00 00 00**

Reverse engineering of ECU functionality is needed to determine how systems work. System knowledge is useful to hackers, pen testers, functional safety engineers, rival manufacturers, and parts suppliers. The reverse engineering of the display ECU could be continued. Indeed, another run of the fuzz testing, with the found ids excluded, found a CAN packet with id 753 responsible for an additional 4 messages. However, the focus of our research is not to fully reverse engineer a component, as the primary aim is to determine the usefulness of fuzz testing to improve automotive systems assurance levels. To that end the results from testing against the ECU have provided useful evidence to support the use of automotive fuzz testing. Furthermore, the results were used to inject unexpected messages into the lab vehicle.

A. Injecting display ECU messages

One aim of security experiments is to learn enough about a component in order to use the knowledge to compromise a vehicle. In doing so it provides evidence needed to improve the component design, system design and mitigate the attack.

For the display ECU the knowledge to control what is shown to vehicle occupants is useful for attackers. In this case it 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.

It can be seen, Fig. 6, that the display ECU is not connected to the On-Board Diagnostics (OBD) port, a common entry when attacking vehicle CAN busses. Thus, the CAN packets needed to display a rogue message must be sent via a connection to the vehicle's internal CAN bus. This is achieved using a man-in-the-middle connection close to another ECU under the dashboard, see Figure 11.

Thus, the message injection is not as straightforward as it could be. However, access to the network is not difficult and certainly possible by someone who is knowledgeable. There is

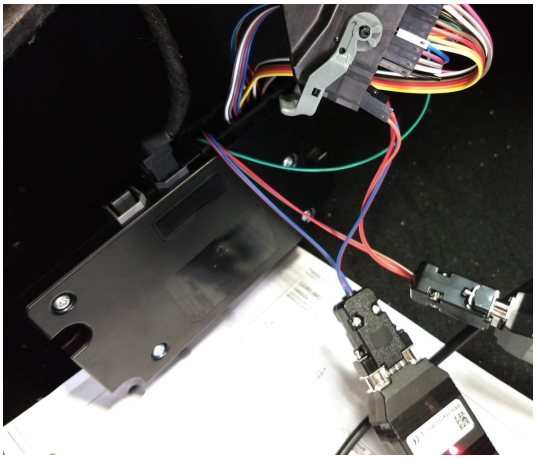

Fig. 11. Splicing into the lab's vehicle's internal CAN network via an Audio Interface Module (AIM) ECU located below the dashboard. This provided an access point for injecting invalid CAN data.

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 stopping a vehicle or a 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.

In Figure 12 the message injection can be seen in action. The injected 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. 12. CAN packets are injected into the lab vehicle's internal network to display false messages. The messages are incorrect because the vehicle requires the brake to be pressed to start, and the brake fluid level is correct.

VII. Conclusion

The full range of processes required for automotive security testing are not trivial due to the attack surface of vehicles. Furthermore, complexity is increasing with the advent of autonomous driving. We have developed a method that is suitable for developing testbeds, tooling and security tests. Our method has been applied to automotive fuzz testing, starting with insecure CAN, which is present within all mass manufactured vehicles. We have developed an easy to use and easy to deploy prototype CAN fuzzer to supplement TARA and secure-by-design with a dynamic test method.

The fuzzer was used against a display ECU and it revealed problems with the software, confidential operational information, and system integrity issues. If this security fuzz testing had been available, and was performed prior to production, then beneficial system design changes and bug fixes would have been made. It is also applicable to the post-production vehicle life cycle, where security testing can be applied to design iterations. Therefore, the development of security tests is a requirement in automotive engineering, and supplements traditional functional testing. A validated security test, as with the CAN fuzz testing, does improve system security assurance.

One area that requires further research, and noticeably absent from the literature, is how to gather useful metrics on fuzz testing. The fuzzer will be used to explore this issue within the automotive field.

Finally, our method and research will be used to develop other security tests and tooling, aimed at expanding the knowledge within the relatively new automotive cybersecurity field in the literature

References

[1] C. Miller, and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," Black Hat USA, 2015.
[2] SAE International, "J3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," Warrendale, 2016.
[3] BSI, "PAS 1885:2018 The fundamental principles of automotive cyber security - Specification," 2018.
[4] P. Wooderson and D. Ward, "Cybersecurity Testing and Validation," SAE Technical Paper, 2017.
[5] 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, Technical Report, 2007.
[6] D. S. Fowler, J. Bryans, S. A. Shaikh, P. Wooderson, "Fuzz Testing for Automotive Cyber-security," 4th Workshop on Safety and Security of Intelligent Vehicles (SSIV 2018), 2018.
[7] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, "A systematic review of fuzzing techniques," Computers and Security, 75, pp. 118-137, 2018.
[8] ISO, "ISO/IEC 15408-1:2009(E) Information technology - Security techniques - Evaluation criteria for IT Security," 2014
[9] R. Nishimura, R. Kurachi, K. Ito, T. Miyasaka, M. Yamamoto, and M. Mishima, "Implementation of the CAN-FD protocol in the fuzzing tool beSTORM," 2016 IEEE International Conference on Vehicular Electronics and Safety (ICVES), pp. 1–6, 2016.
[10] D. K. Oka, A. Yvard, S. Bayer, and T. Kreuzinger, "Enabling Cyber Security Testing of Automotive ECUs by Adding Monitoring Capabilities," Embedded Security in Cars Conference, 15th escar Europe, Berlin, isits AG, 2016.
[11] R. Kurachi, and T. Fujikura, "Shift Left: Fuzzing Earlier in the Automotive Software Development Lifecycle using HIL Systems," 16th escar Europe, Brussels, isits AG, 2018.
[12] P. Lapczynski, H. Heinemann, T. Schöneberger, and E. Metzker, "Automatically Generating Fuzz Tests from Automotive Communication Databases," 5th escar USA, Detroit, isits AG, 2017.
[13] P. S. Oruganti, M. Appel, and Q. Ahmed, "Hardware-in-loop Based Automotive Embedded Systems Cybersecurity Evaluation Testbed," Proceedings of the ACM Workshop on Automotive Cybersecurity, AutoSec '19, New York, NY, USA, pp. 41–44, ACM, 2019.
[14] D. S. Fowler, M. Cheah, S. A. Shaikh, and J. Bryans, "Towards a Testbed for Automotive Cybersecurity," Software Testing, Verification, and Validation, ICST, International Conference on, Tokyo, IEEE Computer Society, pp. 540–541, 2017.
[15] J. Liu, S. Zhang, W. Sun and Y. Shi, "In-Vehicle Network Attacks and Countermeasures: Challenges and Future Directions," in IEEE Network, vol. 31, no. 5, pp. 50-58, 2017.