2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops

Fuzz Testing for Automotive Cyber-security

Daniel S. Fowler, Jeremy Bryans and Siraj Ahmed Shaikh
Systems Security Group
Coventry University
Coventry, CV1 5FB, UK
Email: {fowlerd3,jeremy.bryans,siraj.shaikh}@coventry.ac.uk

Paul Wooderson
HORIBA MIRA Limited
Watling Street
Nuneaton, CV10 0TU, UK
Email: paul.wooderson@horiba-mira.com

***Abstract*—There is increasing computational complexity within the connected car, and with the advent of autonomous vehicles, how do manufacturers test for cyber-security assurance? The fuzz test is a successful black box testing method that hackers have used to find security weaknesses in various domains. Therefore, should the fuzz test, mentioned (without any details) in SAE J3061, be applied more widely into the vehicle systems development process to help reduce vulnerabilities? To investigate this question a custom fuzzer was developed to allow for experimentation against a target vehicle's CAN bus (used as the data interconnect for the vehicle's ECUs). The results demonstrate that the fuzz test has a part to play as one of the many security tests that a vehicle's systems need to undergo before being made ready for series production. However, previous problems raised when cyber testing a vehicle were confirmed. Thus, in adding the fuzz test to the automotive engineering tool box some issues are raised that need addressing in future research.**

I. Introduction

All mass manufacture vehicles require internal networked computers in order to function. The computers in vehicles are called Electronic Control Units (ECUs). The ECUs run everything from the engine to the interior lights and are commonly interconnected with the low-cost Controller Area Network (CAN) data bus. Other networks found in vehicles include FlexRay, Media Oriented Systems Transport (MOST), Local Interconnect Network (LIN) and two-wire Ethernet (100BASE-T1). These vehicular systems maybe connected directly or indirectly to the Internet. This is achieved via built in cellular communications, or via driver or passenger gadgets, for example smartphones. This makes the cars we drive, or the transport we use, cyber-physical systems (CPS) and Internet of Things (IoT) devices. These connected cars have been shown to suffer from similar cyber-security vulnerabilities [1] as other computer based networked systems, whether home, office or industrial. Thus, the connected and autonomous vehicle (CAV) needs to be resilient against cyber attacks. Therefore, cyber-security testing has been added to the list of tasks that are part of, or should be part of, a manufacturer's vehicle engineering process [2], as governments are now stipulating [3].

Applying established cyber-security testing methods to automotive engineering is challenging due to the domain specific technology and CPS environment. Is the fuzz test, a proven dynamic test method for software, a worthwhile and useful addition to the automotive domain? The contribution here is a method to examine that question, via a custom fuzz test program, and to begin to address the challenges in bringing the fuzz test into automotive systems testing.

After briefly introducing the fuzz test the existing work in applying it to vehicle systems is examined. The motivation and challenges in testing CPS systems are covered. The vehicle technology subjected to the fuzz test, the CAN bus, is then introduced. Running the custom software and the resultant output is presented, before discussing the observations and further work, and conclusion.

A. What is a fuzz test?

The fuzz test is a dynamic analysis test method, i.e. it is performed against a running system (as opposed to static analysis of source code). A fuzzer is a program that performs fuzz tests. 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 occurs the conditions that caused it are recorded and the system is reset.
- 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 number of tests that are executed.

Being able to cause software to fail is one of the methods attackers use to penetrate systems. As such fuzzing is now a mature and well established method to find vulnerabilities in applications [4] and operating systems [5], and is used to help reverse engineer automotive systems [6]. Yet its use in general testing of automotive systems is low (Figure 1) [7].

II. Related Work

Research into vehicle hacking is often concerned with proving vulnerabilities in connected vehicles [8]. Some researchers do offer solutions alongside the attack [9]. Despite early practical demonstrations [10] in compromising the Controller Area Network (CAN), it was not until practical connected car cyber attacks were widely publicised in the mass media in 2010 [11], that interest in cyber attacks against vehicles increased. The original 2010 [12] research that achieved the widespread media coverage noted that:

2325-6664/18/$31.00 ©2018 IEEE
DOI 10.1109/DSN-W.2018.00070

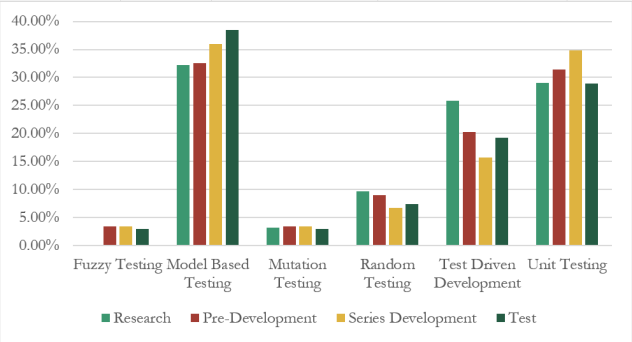

Fig. 1. Testing methods in the automotive industry, derived from data from [7]

> In fact, because the range of valid CAN packets is rather small, significant damage can be done by simple fuzzing of packets (i.e., iterative testing of random or partially random packets). Indeed, for attackers seeking indiscriminate disruption, fuzzing is an effective attack by itself.

Although their main use of fuzzing was to help reverse engineer the target vehicle's systems. However, in web services and general information systems fuzzing is used to break software. Yet the value of fuzzing for car hacking, so far, has been in helping to find how vehicle systems function [6]. This is because the operational details of a vehicles internal systems are commercial secrets. Often the only way to determine what a particular CAN message does is to capture the network packets while operating a vehicle feature. There has been little work done on the usefulness of the fuzz test 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 [13]. 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.

The test oracle problem (how to determine, or not, the correct responses of a system) [14] is a challenge for CPSs, particularly with automating the entire security testing process. In [15] a link from a hardware-in-the-loop (HIL) and software-in-the-loop (SIL) test and development system, into the Python based open source fuzzer, called booFuzz, is presented. They propose several ways to address the test oracle issue:

- Network communication monitoring.
- Monitoring through a component debug interface. Normally used for hardware debugging and not normally available once manufacturing begins.
- Direct and indirect monitoring of system signals available internally to the simulator.
- Use of the automotive Universal Measurement and Calibration Protocol (XCP) that allows remote access to the internals of an ECU.

TABLE I
AUTOMOTIVE CAN FUZZING TOOLS

Tool	*License*	*Approach*
beStorm	Commercial	Protocol based
Defensics	Commercial	Protocol based
CANoe/booFuzz	Mixed	Design based
Peach	Mixed	Protocol based
Custom software	As required	As required

- Monitoring of the physical responses of the system with external sensors.

However, it was not considered that any extra monitoring capabilities may be used by the attackers, who look for any source of information to help break systems. Thus, supporting XCP may help with detailed ECU diagnostics, but it provides another channel that may be exploited. One interesting point noted is that automotive ECUs have different 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. It is important for system testers to cover all the states of an ECU, as these different states have been previously exploited [1].

How a commercial fuzzer is configured to interface to an automotive network is provided in [16]. No practical application is given, only publishing results on data packet throughput rates. In [17] another commercial fuzzing tool is used to test a single ECU. The test environment defines a comparison module that acts as the test oracle, verifying or not the correct operation of the ECU when its CAN messages are being fuzzed.

Table I lists the fuzzers used in the published referenced work, plus the Peach fuzzer, which is advertised as supporting automotive testing[1]. Most are general purpose commercial products, booFuzz is open source and Peach has an open source version. They all require configuring to work with automotive systems. The two main approaches are 1) protocol, using the format of the CAN data packets (Table III), or 2) input from the system design, i.e. having pre-existing knowledge of the data packet contents (and could be informed from the source code running in an ECU). In the following section the need for the automotive fuzz test is examined.

III. MOTIVATION

Safety is a key design objective for a vehicle. Cars are tested for operational correctness, certification, and homologation against international and national standards. The International Organization for Standardization (ISO) publishes ISO 26262 [18] for functional safety of electrical and electronic systems. However, for the connected car the cyber attack is a threat beyond the normal functional operation. How can that threat be addressed? The Society of Automotive Engineers (SAE) J3061 publication [2] provides some best practice guidelines for CPSs as a starting point for introducing security aware processes and designs, plus ISO/SAE AWI 21434 (Road

[1] https://www.peach.tech/

Vehicles, Cybersecurity Engineering) is in development. However, for systems engineers applicable methods are needed.

A. From Physical Locks to Cyber-locks

Fundamentally security is an economic problem [19], enough security must be added to a system to dissuade the adversaries. Security measures were once only physical, e.g. stronger locks. Now authentication, data and communications encryption, and other cryptography measures are required for the cyber-locks to protect security properties, in the form of the CIA triad:

- Confidentiality - preventing the viewing of sensitive data.
- Integrity - preventing changes in data values.
- Availability - protecting system operations.

Still, cyber-locks do not guarantee system security. The general problem with computer code is that it is rarely error free. There will be a given error rate in the hundreds of thousands of lines of code (LOC) in a computational system, despite best efforts to eliminate bugs [20]. The goal of the attacker is to find the code errors that can be exploited for gain. The greater the effort (i.e. cost) involved in finding a weakness, the more likely the attacker will look elsewhere. However, the manufacturer faces a dilemma, at a certain point reducing the number of errors in a system becomes exponentially harder [21], and any project has finite resources. Methods that can alleviate this tension are therefore beneficial.

B. The CPS Fuzz Test Challenges

If hackers use fuzzing to break systems why not deploy the same technique during pre-production testing to improve cyber-security resilience. Indeed fuzz testing is one part of the well regarded Microsoft Secure Development Lifecycle (SDL) [22], and is listed in J3061. However, in J3061 there is no coverage of any resources to aid manufacturers, and there has been little in-depth research into applying the fuzz test to CPS domains. What are the challenges in getting the fuzz test more widely used in the automotive industry?

1) The CPS nature of the vehicle: For the CPSs that are now being deployed around us, the computational command and control results in physical non-computational actions. A vehicle driver pressing a switch can cause a digital command to be sent over a data network, and the action results in a real world output, e.g. a light being turned on. Thus, monitoring of the system under test (SUT) or device under test (DUT) has added CPS complexity. However, for the pre-production vehicle design phase this CPS monitoring complexity can be mitigated with the aid of hardware-in-the-loop and software-in-the-loop equipment to simulate the physical world.

2) The large volumes of data generated by a vehicle: The data volumes continuously generated inside a vehicle will only increase as more computational systems are incorporated into cars. The data volume issue is a problem not unfamiliar in traditional IT security. Furthermore, different communications protocols are deployed in the automotive industry, plus there is a range of data types and formats to handle. Vehicle data handling is another big data problem.

3) Incomplete knowledge: Functional tests that target the known specification and interfaces of a component are easily determined. However, additional features developed for other uses, e.g. to support other customers or for component testing, may be present. An undocumented application programming interface (API), as well as an untested code path, could be exploitable. Automotive engineers need to factor in the unknowns.

4) Defining measureable metrics: The complexity of a CPS means no two similar systems, or similar components are comparable. This means that measuring the effectiveness of a fuzz test is difficult. In other domains the fuzz test is orientated towards the final count of the number of bugs found [5]. However, this can only be relative to other runs on the same system, plus, due to the random nature of the fuzz test, the comparisons can only be approximations. For the attacker the total number of flaws found is irrelevant. They are after the one flaw that gives them the ability to violate the CIA triad. Whereas the system manufacturer needs to find as many of these flaws as possible prior to system delivery. However, if no flaws are found it does not mean none exist, it just means that testing has not triggered anything.

Given the above motivation and challenges, the hypothesis is that the fuzz test is beneficial for improving the security of the networked ECUs in a vehicle. If true, then adding the fuzz test to vehicle testing methods, as listed in J3061, is a valid assertion. Here a custom fuzzer is used to apply the fuzz test to a target vehicle. In doing so what can be learnt to address the challenges? How does the fuzzer need to be adapted? What is needed for future applications of the fuzz test in the automotive domain? Whilst empirical results do exist for applying the fuzz test to vehicle systems, there have been few published experiments with usable lessons.

Here the fuzzer is interfacing to the vehicle via the CAN bus, thus a brief introduction to CAN is provided.

IV. The Automotive CAN bus

The CAN bus was introduced in the 1980's, for those interested in the bit level protocol there are plenty of resources available [23]. Each node (ECU) on the bus can initiate data transmission, with only one node at a time transmitting. For a standard CAN data frame the 11-bit arbitration identifier (a.k.a. packet id) allows for the highest priority messages to continue transmission in the event of two to more nodes transmitting simultaneously. A standard CAN packet has up to eight bytes (64 bits) for data. The CAN transceiver chips in a node handles the protocol automatically, providing the id, data length and data bytes to the higher level application. Table II shows some CAN packets captured from the target vehicle. By today's standards the transmission speed of standard CAN is modest, designed to support up to 1Mb/s. A common transmission speed used in cars is 500kb/s.

CAN, designed without security in mind, is easily accessible. Equipment to interface to CAN is low cost. A CAN bus is usually exposed in a vehicle via the open, in-cabin On-board Diagnostics (OBD) port, plus a wire tap is possible wherever

TABLE II
EXAMPLES OF CAN PACKETS CAPTURED 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
5330.007	04B0	8	00 00 00 00 00 00 00 00
5331.029	04F2	8	00 53 6C 00 00 00 00 00
5338.165	0215	7	00 1C 01 00 00 01 40

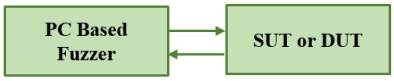

Fig. 2. PC based fuzzer for automotive testing

the vehicle wiring is accessible. These factors allow for straightforward research into the manipulation of CAN data. A compromised ECU or man-in-the-middle (MITM) attack (for example an aftermarket device attached to a vehicle's OBD connector) can spoof the messages transmitted, affecting normal operation and threatening vehicle and passenger safety. This CAN manipulation is one element of successful cyber attacks against vehicles [1]. Strengthening the security of CAN is a useful goal, however, despite several schemes available to add encryption to CAN, no scheme meets all the criteria for deployment in series production [24].

V. METHODOLOGY

The available commercial and mixed licensed fuzzers listed in Table I have been specifically customised for use with automotive systems. The custom fuzzer introduced here, Figure 2, developed for HORIBA MIRA Ltd., is specifically programmed to deal with the CAN format. The fuzzer's design is simple, with a smaller number of sub-components than, for example, [15] which jumps between Python and .NET technologies when executing.

1) PC based fuzzer: The fuzz test software was developed on a computer linked to a Vector vehicle simulator (Figure 8). The Vector equipment is widely used in industry for design, validation, HIL, and SIL testing of networked control systems, including vehicular CAN based systems. The use of a vehicle simulator 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 messages, a communications API handling module, and a CAN bus traffic monitor.

2) Link to the SUT or DUT via the vehicle data bus or ECU interface: A Universal Serial Bus (USB) to CAN hardware adaptor is used to connect the fuzzer software to the CAN bus, here a PCAN-USB product manufactured by PEAK-System is used. The PCAN-USB device has an Application Programming Interface (API) that allows the device to be accessed from C#.

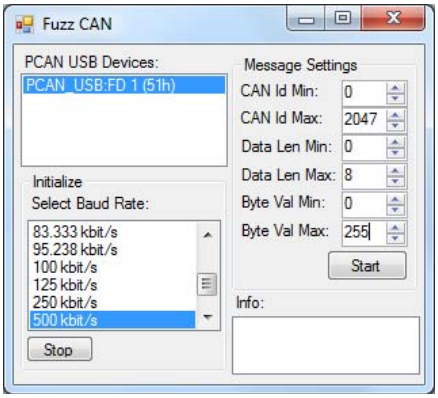

Fig. 3. UI to configure the CAN fuzzer

The USB CAN adaptor requires 9-way D-type sockets wired to conform to the CANopen specifications (CiA303-1)[2]. 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 fuzzer is configured through its UI, providing control over the data injected into the SUT or DUT (via the CAN bus), Figure 3. Through the UI the fuzzer can be programmed to generate a variation on a single bit in a single message, to every bit in every message. This feature is important due to the combinatorial explosion problem with the CAN data stream. A standard CAN packet with a 11-bit id and a one byte payload has half a million packet combinations (2^{19}). At a 1ms transmission frequency (the current minimum for this fuzzer) it is 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 needs to be targeted (for example by fuzzing around known message ids monitored on the CAN bus, or being informed by the design).

Once configured the fuzzer executes against the SUT or DUT, sending out the random CAN data. It monitors the target system to record and act upon responses to the injected messages.

VI. RESULTS AND ANALYSIS

The target vehicle uses 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 III. From the parameters defined in the fuzzer the random CAN data packets are generated, Table IV.

The fuzzer analyses CAN data to allow for data integrity checks. Figure 4 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

[2]https://www.can-cia.org/standardization/specifications/

TABLE III
FUZZING ELEMENTS OF A CAN DATA PACKET FOR THE TARGET VEHICLE

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

TABLE IV
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

of eight bit values. In comparison Figure 5 shows the same calculation on 66144 CAN packets generated by the the fuzzer. The linear distribution, with a overall mean value of 127 for all bytes in all messages, providing evidence that the fuzzer is correctly generating an even spread of byte values.

The effect of the random CAN packets from the fuzzer can be measured in the Vector simulator. The normal vehicle signals are illustrated in Figure 6. Figure 7, captured over a shorter period than Figure 6, illustrates the effect of the randomised data packets on the 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 8 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.

Once the fuzz test software was operational it could be used against physical targets, i.e. vehicles and vehicle components. Previous car hacking research has shown that permanent damage to vehicles is possible, for example rendering a ECU non-functional (bricking) [25]. Therefore, before testing against the target vehicle, an available instrument cluster, used on the target vehicle, was fuzzed. Running the fuzzer against the in-

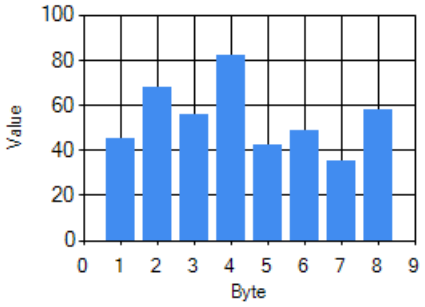

Fig. 4. Mean values for each data byte position from 100000 captured vehicle CAN messages

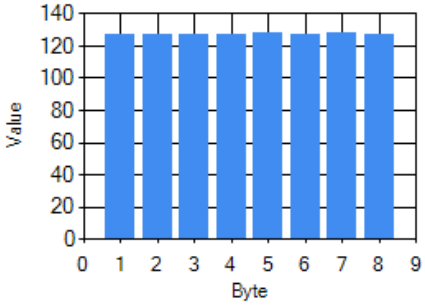

Fig. 5. Mean values for each data byte position from 66144 randomly generated CAN messages

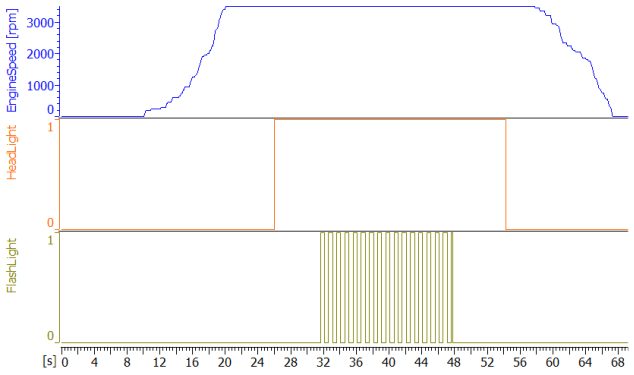

Fig. 6. Simulated vehicle signals

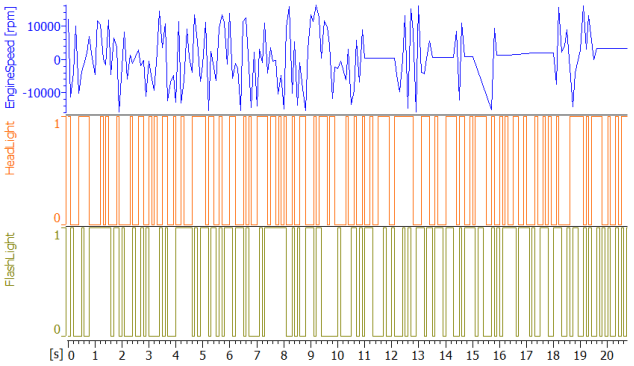

Fig. 7. Effect of fuzzing on signals

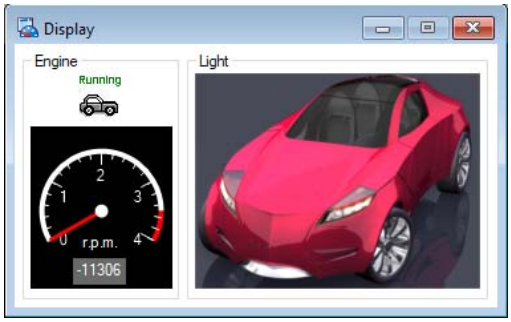

Fig. 8. Inappropriate value on a vehicle simulator display via fuzzing

Fig. 9. Crashing a vehicle component as a result of fuzzing

strument cluster immediately resulted in Malfunction Indicator Lights (MIL) illumination, warning sounds and erratic gauge needles. Furthermore, a digital display began to display the word **crash** at a regular rate. Cycling the power to the cluster removes any MILs that became illuminated. Unfortunately the crash message would not clear.

This damage to the component required a reassessment of using the fuzzer against the target vehicle, a shared resource that would incur repair costs if damaged. Therefore, the experiment was limited to just checking that the fuzzer had an effect on the target vehicle. Instead of running the fuzzer across the entire CAN message space, only a small range of messages would be fuzzed. Messages IDs that had been previously observed on the vehicles CAN buses in normal operation, for example the message known to affect the instrument cluster gauge needles. With the fuzzer connected to the vehicle using an OBD cable (via the USB to CAN adaptor), fuzzed messages were sent into the idling target vehicle. The target vehicle exposes two CAN buses, the fuzzer was tested on both buses. The vehicle exhibited similar behaviour to the cluster testing, namely the illumination of various MIL lights, warning sounds from the instrument area, fluctuating gauge readings, error messages on a central vehicle console, and erratic engine idling RPM. Once it was observed that fuzzing had a significant affect it was halted to prevent possible damage, as in Figure 9.

In order to prevent the possibility of damage to the target vehicle's components, further testing of the fuzzer was performed against a bench-top hardware configuration. The bench based configuration was implemented to represent an increasing common feature of connected cars, namely the control of vehicle functionality via an app, Figure 10.

A CAN bus target was constructed from Arduino single board computers (SBCs) fitted with CAN interfaces. Each SBC acting as an ECU on the network. The messages on the CAN bus were a small subset of those transmitted on the target vehicle's CAN bus. One of the ECUs acts a Body Control Module (BCM), with a Light Emitting Diode (LED) representing the lock status of the vehicle (off for locked, on for unlocked), Figure 11.

A diagram of the replicated functionality is shown in Figure 12. The external phone app sends an unlock command to a vehicles infotainment ECU (a.k.a. head unit). This is

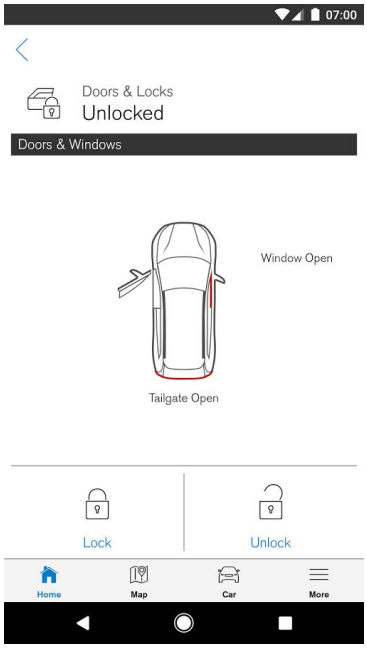

Fig. 10. Vehicle control via a manufacturer's smartphone app available from the app stores

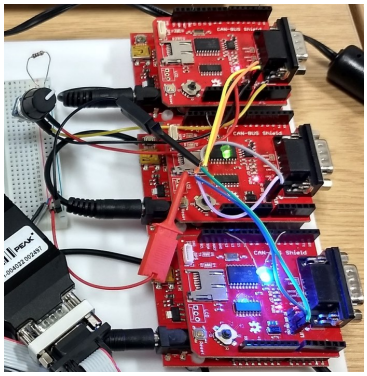

Fig. 11. CAN bus connecting three single board computers acting as ECUs

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 (via the OBD port or a compromised ECU). When the fuzzer runs it has no knowledge of the CAN message to activate the locks.

For this experiment a PC app acts as the smartphone app, Figure 13, sending the lock and unlock command as a proxy for the infotainment ECU. This causes the LED to turn on and off, indicating the normal system operation in locking and unlocking the door. With the fuzzer, Figure 3, 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 message. For the real vehicle

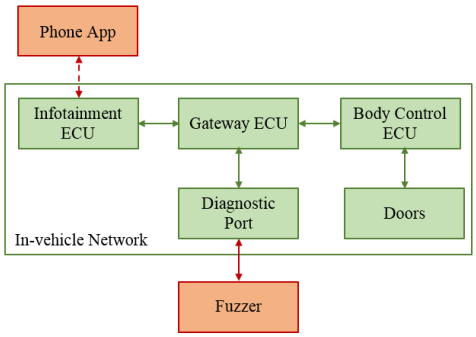

Fig. 12. Remote vehicle unlock functionality

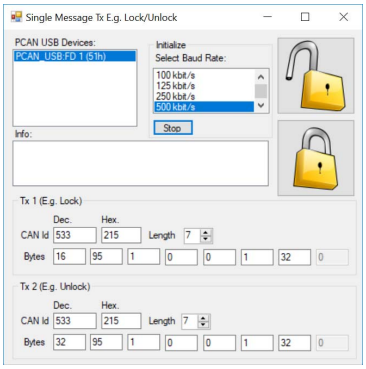

Fig. 13. PC vehicle lock/unlock app

another detection mechanism would have been required, for example a sensor on the door lock.

The fuzzer currently has a maximum message transmission rate of one message 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 V. The unlock code was testing for a specific byte value in byte position one in a message 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 time taken for the fuzzer to find the correct unlock message. If the change had been to check for a two byte value the time increase would have even greater.

VII. Discussion

These experiments have proved useful in pulling together several strands of information from the few works available on applying the fuzz test to the vehicular CAN bus. The results confirm previously made statements on fuzz testing vehicles, namely:

- The fuzz test can be used to reverse engineer vehicle messages.
- 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 components can lead to vehicle component damage.

Since fuzz testing has a detrimental effect on a running vehicle, and the simulator, it suggests that vehicle systems need additional logic to ignore nonsensical CAN message values, and sequences of such values. Thus, it is apparent that the CAN bus needs additional engineering considerations when operating in a connected car. Therefore, protection of the CAN bus and vehicle components from external cyber attacks is now a functional requirement for the design of connected vehicle systems. For such systems the aim of these experiments was to show the method that can be used to start incorporating the fuzz test into the testing regime.

It has also been shown that a fuzz test can be used to verify a systems conformance to the CIA triad. The fuzz 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 a gateway ECU in newer vehicles indicates that manufacturers are responding to the issue. Furthermore, simple modifications to a design improve security. Here, by changing the CAN message to activate the unlock and therefore, increasing the time to find that feature by random fuzz testing.

The developed software will be used and extended for further study of fuzz testing vehicle systems. It is a useful research area as there is little quantitative data available on running such tests on automotive systems. Other ways that the work can be extended include:

- Use the fuzz test to determine the effectiveness of protection measures, for example vehicle firewalls and gateways, or additions to ECU software to mitigate cyber attacks.
- Investigate manipulation of data packets at the bit level to fuzz CAN protocol control bits (the data link layer).
- Apply the techniques to the Flexible Data-rate (FD) version of CAN.
- Fuzz the APIs for vehicle engineering tools (e.g. CAN interface devices) to ensure their resilience. For example fuzz the API for the PEAK USB CAN adaptor used in study.
- Use video processing software, for example OpenCV, to monitor the cyber-physical actions of vehicle and devices being fuzz tested.

TABLE V
Fuzzer run times to activate unlock

Message	*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

VIII. Conclusion

Published work, section II, has been more concerned with the configuration of commercial fuzz test software for automotive testing, rather than results that directly examine the use of the fuzz test to tackle CIA weaknesses in automotive systems. If the consumer is to maintain trust in future CAV vehicles then test methods to exercise CIA resilience, such as the fuzz test, are required [26]. As such a body of knowledge is needed on how to apply the fuzz test to the automotive domain that goes beyond simply stating that the fuzz test must be used on vehicles [2]. The contribution here begins to address the need for more research in this area, providing some foundation for making the fuzz test a worthwhile automotive test method.

A simple to use CAN fuzzer was developed to perform a fuzz test on vehicle systems. In applying it to a test vehicle it was evident that the systems were not resilient enough. Another scenario was developed which demonstrated that the fuzz test 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 domain is likely to be in fuzz testing in a specific message space, close to known messages, whether determined from design or data traffic capture. Further research is needed to explore the benefits of running such targeted fuzz tests, particularly with regards to finding unconsidered code paths in ECUs.

It was also apparent that vehicle and component manufacturers need to consider supporting such research, due to the cost implications of working with their vehicles and components. It is not practical for security researchers to have access to a single component or vehicle, ideally access to several are required. This has been previously acknowledged as a problem and cost issue, and can be mitigated to some extent using simulation [27].

Finally, the developed fuzz tester will be the foundation of a useful tool that will find application in the security testing of vehicle systems. However, it is only one small piece of the connected car security engineering and testing process.

Acknowledgment

This research is partially supported by HORIBA MIRA Ltd. as part of their collaboration with Coventry University's Institute for Future Transport and Cities.

References

[1] C. Valasek and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, vol. 2015, pp. 1–91, 2015.

[2] SAE International, "J3061 - Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," Warrendale, 2016.

[3] HM Government, "The Key Principles of Cyber Security for Connected and Automated Vehicles," HM Government, Tech. Rep., 2017.

[4] N. Rathaus and G. Evron, *Open Source Fuzzing Tools*. Burlington: Syngress, 2007.

[5] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox Fuzzing for Security Testing," *Queue*, vol. 10, no. 1, pp. 20:20—-20:27, jan 2012. [Online]. Available: http://doi.acm.org/10.1145/2090147.2094081

[6] C. Smith, *The Car Hacker's Handbook : A Guide for the Penetration Tester*. No Starch Press, 2016.

[7] 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.

[8] 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.

[9] 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.

[10] T. Hoppe and J. Dittman, "Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy," in *Proceedings of the 2nd workshop on embedded systems security (WESS)*, 2007, pp. 1–6.

[11] BBC, "Hack attacks mounted on car control systems," p. 1, 2010. [Online]. Available: http://www.bbc.co.uk/news/10119492

[12] 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.

[13] 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.

[14] 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, 5 2015.

[15] 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., jun 2017.

[16] 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)*, jul 2016, pp. 1–6.

[17] 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.

[18] ISO, "ISO 26262-1:2011 Road vehicles - Functional safety - Part 1: Vocabulary," Geneva, p. 23, 2011. [Online]. Available: https://www.iso.org/standard/43464.html

[19] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed. Indianapolis: Wiley Publishing Inc., 2008.

[20] B. G. Kolkhorst and A. J. Macina, "Developing error-free software," *IEEE Aerospace and Electronic Systems Magazine*, vol. 3, no. 11, pp. 25–31, nov 1988.

[21] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, dec 2010. [Online]. Available: https://doi.org/10.1007/s10515-010-0069-5

[22] M. Meng and W. Khoo, "An Analysis of Secure Software Development Lifecycle from an Automotive Development Perspective," SAE, Warrendale, Tech. Rep., 2016. [Online]. Available: https://doi.org/10.4271/2016-01-0040

[23] Bosch, "CAN Specification Version 2.0," Robert Bosch GmbH, Tech. Rep., 1991.

[24] 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)*, Sept 2017, pp. 1–7.

[25] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *20th USENIX Security Symposium*. San Francisco, 2011.

[26] 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, March 2018.

[27] 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, 2017.