1 Acronyms and abbreviations

ACK	Acknowledgement
API	Application Programming Interface
CAN	Controller Area Network
CAR	California Air Resources Board
CLC	Cyclic Redundancy Check
DLC	Data Link Connector
DLC	Date Length Code
DoS	Denial of Service
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
EOF	End Of Frame
ID	Identifier
IDE	Identifier Extension Bit
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
RTR	Remote Transmission Request
TPMS	Tire Pressure Monitoring System
V2P	Vehicle-to-pedestrian
V2N	Vehicle-to-network
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle

Contents

2 Introduction

Motivation A lot of things have happened to the automotive industry, ever since the introduction of the Ford Model T in 1917. Although the main purpose of these machines remains the same (e.g. getting someone from point A to B swiftly), the relative comfort, speed and efficiency at which they do so has improved dramatically. The most important contributor to this evolution is without a doubt the introduction of electronic computers into the vehicle's architecture. The modern vehicle has been appropriately called a "Computer on wheels" [2], since each one contains up to 100 millions lines of code, spread out over as much as 70 individual ECUs (Electronic Control Units) [1]. Each ECU is a small computer controlled device that is designed to perform a specific function (e.g. braking, opening the door, speed control, etc). In addition to having this wide variety of embedded devices, a modern vehicle will also employ a data bus that allows all ECUs inside a vehicle to communicate with each other. There are multiple standards that are employed even within a single vehicle, but the CAN (Controller Area Network) protocol is probably the most common and will be the focus of this paper. Alongside internal communication networks, a lot of newer models will also introduce some way of performing external communications. This can range from vehicle-to-infrastructure (V2I) communications (e.g. wireless gas payment at a gas station, wireless diagnostics at a repair shop or even virtual traffic lights), vehicle-to-vehicle communications (V2V) (e.g. automatically following another vehicle), vehicle-to-network communications (V2N) (connecting your vehicle to an already existing network, like the cellular communications network for example) and vehicle-to-pedestrian communications (V2P) [3][4][5]. All of the extended functionality introduced greatly improves the vehicle's flexibility and comfort, but also makes them increasingly vulnerable. As with any interconnected computer system it is susceptible to attack from malicious agents if the right level of security[28] is not attained. This is made even worse by the introduction external communications (e.g. V2I, V2V, V2N and V2P) since this allows for remote attacks, eliminating the need for physical access to the internal network (which of course should be hard to obtain). A lot of the ECUs that facilitate external communications can now be seen as potential attack vectors. One of these (and also one of the most vulnerable) is the OBD-II (On board Diagnostics) interface. OBD-II systems are widely deployed in auto-mobiles as a way of getting diagnostics information from the vehicle. OBD-II introduces a physical interface into the vehicle (usually under the steering wheel) called the Data Link Connector (DLC). This physical interface allows full access to the internal network. It has been repeatedly shown [6][7][24][25] that a set of messages or signals could be injected on a car's CAN bus to control key components (e.g. lights, locks, brakes, and engine) as well as injecting code into key ECUs. The focus of this thesis will be to try and mitigate this kind of illegal access by introducing access control to the OBD-II interface.

Challenges The main challenge of this research topic is to introduce a solution that ports well to the kind of hardware that is found in modern vehicles. Introducing new components into the internal vehicle network would surely simplify things (if this were the case the solution could consist of introducing a small component that acts as a firewall for the OBD-II interface). However this would imply that any potential real-world implementations by a car manufacturer would require the installation of these components in millions of existing vehicles, which (being a very costly endeavour) would deter any manufacturers from doing so. It is the opinion of the researchers of this paper to focus on designing a solution that is easy to introduce into an existing vehicle network by way of a simple software update. Because of this decision it becomes important to take existing vehicle networks and solutions (a.k.a the context) into account when designing a solution. A leading example of constraints introduced by the context are the memory limitations of ECU micro controllers. Indeed, any solution that isn't portable to the network (because of memory limitations, limited processing power, incompatible architectures, etc) is rendered useless. Another example is the impact of the context on the threat model. It is impossible to deny the owner of the vehicle direct access to it's internal network (In theory the owner could bypass the OBD-II interface by gaining direct access to the wires that compromise the network, or exploiting a vulnerability of a different entry point [6]), It is easy to see that in a proposed role-based access control solution this threat is out of scope. The main challenges of this thesis paper are summarized as follows:

- **portability:** The solution should port well to existing vehicle networks.
- **Security:** The solution should be sufficiently secure according to current computer safety standards.
- **Speed:** The solution should not impede the operation of other processes running on the same network.

Context As mentioned before the goal of this paper is to secure the OBD-II interface in modern cars. Before we move on it is interesting to take a look at some other issues regarding internal vehicle networks, as well as some proposed solutions to these problems. Aside from the OBD-II interface there are numerous points of entry to the internal vehicle network, both physical (Breaking into the vehicle and directly connecting to the network) or wireless (Bluetooth, TPMS or Tire Pressure Monitoring System, Radio system, etc) [6]. Take Bluetooth for example: A lot of cars include Bluetooth functionality to allow users to connect their phones and play music. The Bluetooth protocol has a large protocol-stack and has been known to have problems in the past [6]. By exploiting the vulnerabilities of a car's Bluetooth interface, a malicious agent is able to interfere with the internal network remotely (using his/her mobile phone). Another problem is that it is easy for a phone to get compromised (visiting a malicious website) [7]. This problem would be solved by using a more secure version that does not contain the aforementioned vulnerabilities.

Another approach is to secure the protocol that is used for communication within the network. As mentioned before, the CAN protocol is probably the most popular one since almost every new passenger car manufactured in Europe is equipped with at least one CAN network [19]. CAN in itself is a simple bus protocol that allows nodes on a network to send and receive messages. However CAN is a low-level protocol and does not natively support any security features. A number of secure CAN variations have been proposed: Leia[46], VatiCAN[47], VulCAN[45] and CANopen[41]. In the case of CANopen their contributions were summarized as follows:

> 'a generic design for efficient vehicle message authentication, plus software component attestation and isolation using lightweight trusted computing technology'.

Contributions The contribution of this thesis can be summarized as follows:

- To mitigate the problem of unauthorised access to the vehicle CAN network via the OBD-II Data Link Connector. Since OBD-II was designed solely for diagnostic purposes (what's in a name) the goal is to limit access accordingly. 2 Basic protocol designs are proposed.
- The proposed designs have been implemented on hardware similar to the ones found in modern cars, and a small demo is used to illustrate their operation.
- These implementations are then tested to determine if they are in accordance with the challenges proposed in section 2.

3 Literature Review

3.1 Vehicle Network Infrastructure

As mentioned before Todays automobiles contain a series of different electronic components networked together to be responsible for monitoring and controlling the state of the vehicle. Each component can communicate with all other components on the same network. The safety of the vehicle (and consequently the driver) relies on near real time communication between these various ECUs. While communicating with each other, ECUs are responsible for predicting crashes, detecting skids, performing anti-lock braking (ABS), etc [7]. There are only a couple of operations that are performed without using computer control (with the parking brake and steering being the last holdouts)[26] Let's take a closer look at ABS to get a sense of how the internal vehicle network operates. ABS was designed to keep the wheels from locking up during braking. It consists of 3 main components: wheel speed sensors, a pump and a controller. Here's how it works:

- The controller monitors the wheel speed sensors constantly (So each speed sensor periodically sends a message to the Controller).
- The controller will recognize a wheel locking whenever it detects a rapid deceleration.

- Whenever it does detect a wheel locking up, it will use the pump (again by sending a message over the network) to regulate the pressure on the brake of that particular wheel, thereby keeping it from locking up.

Figure 3.1 shows the infrastructure of a typical vehicle network. As mentioned before There are multiple communication standards that are employed even within a single vehicle. In the theoretical example from figure 3.1 we have: Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST) and FlexRay. Each of these protocols specifies how messages are exchanged within the appropriate sub-networks (e.g safety, infotainment, chassis, etc). A critical component in these types of networks (the presence of sub-networks with different communication protocols) is the Gateway ECU (called control gateway in figure 3.1). This component performs a frame or signal mapping function between two communication systems, thereby allowing ECU's on different sub-networks using distinct communication protocols to exchange messages nonetheless.

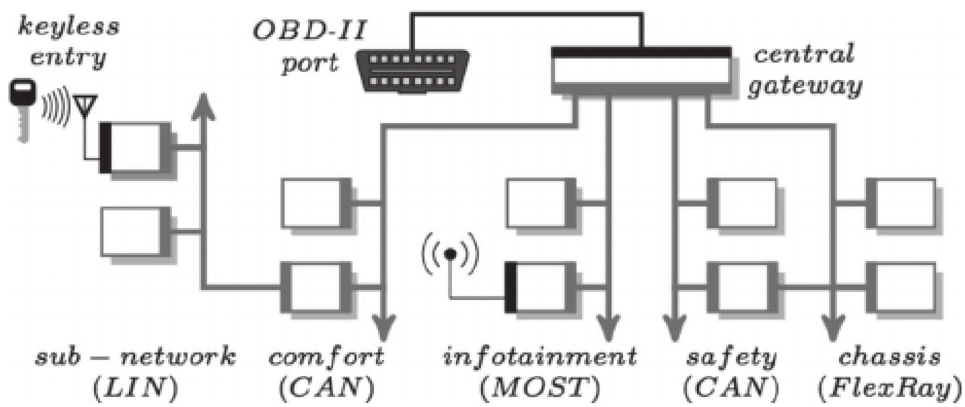

Figure 1: Typical Vehicle Network Infrastructure [8]

On top of acting as an intermediate between the different sub-networks of the vehicle, the Gateway also acts as an entry point for OBD-II messages. Any message sent via the OBD-II DLC will be translated and forwarded by the Gateway to the appropriate sub-network. It comes as no surprise that this component will have to play a crucial role when introducing access control to the OBD-II interface.

3.2 OBD-II

Goal OBD-II (On Board Diagnostics) is a specification that was introduced to allow for self diagnostic and reporting functionality for ECU's inside a vehicle, and has been mandatory in every car produced in the united states since 1996. [11]. It allows users (testers, developers, repairmen, etc) to query ECU's about diagnostics information in order to perform a detailed analysis of the vehicles internal systems. Specifically the goals of OBD-II upon introduction were:

- Standardisation: information is communicated in a standardized format to allow for 1 tool to be used on many vehicles.
- Certification: Every vehicle manufacturer required to submit certification application for review and approval, which includes a detailed description of how the OBD-II protocol was implemented.
- Help lowering emissions by identifying emission controls in need of repair.

The system can also be very useful in a number of other situations: A repairman looking for a specific component that is to be repaired, an employee at the factory testing all components before the vehicle is ready to be sold, a policeman analysing a vehicle after a crash to determine what caused the accident, a software developer testing the operation of a newly developed ECU, etc.

Brief History There were a lot of different proprietary diagnostics systems introduced over the years, before a standard arrived with the introduction of OBD-II. This brief history cited from [12] does a decent job of concisely explaining how OBD-II came to be:

> The origins of OBDII actually date back to 1982 in California, when the California Air Resources Board (ARB) began developing regulations that would require all vehicles sold in that state starting in 1988 to have an onboard diagnostic system to detect emission failures. The original onboard diagnostic system (which has since become known as OBDI) was relatively simple and only monitored the oxygen sensor, EGR system, fuel delivery system and engine control module.

OBDI was a step in the right direction, but lacked any requirement for standardization between different makes and models of vehicles. You still had to have different adapters to work on different vehicles, and some systems could only be accessed with costly "dealer" scan tools. So when ARB set about to develop standards for the current OBDII system, standardization was a priority: a standardized 16-pin data link connector (DLC) with specific pins assigned specific functions, standardized electronic protocols, standardized diagnostic trouble codes (DTCs), and standardized terminology.

Another limitation of OBDI was that it couldn't detect certain kinds of problems such as a dead catalytic converter or one that had been removed. Nor could it detect ignition misfires or evaporative emission problems. Furthermore, OBDI systems would only illuminate the MIL light after a failure had occurred. It had no way of monitoring progressive deterioration of emissions-related components. So it became apparent that a more sophisticated system would be required. The California Air Resources Board eventually developed standards for the next generation OBD system, which were proposed in 1989 and became known as OBDII. The new standards required a phase-in starting in 1994. The auto makers were given until the 1996 model year to complete the phase-in for their California vehicles.

Similar standards were incorporated into the federal Clean Air Act in 1990 which also required all 49-state vehicles to be OBDII equipped by 1996 – with one loophole. The OBDII systems would not have to be fully compliant until 1999. So some 1996 OBDII systems may lack one of the features normally required to meet the OBDII specs, such as the evaporative emissions purge test.

DLC In order to allow for a user to communicate with the vehicle's internal network, OBD-II introduces the data link connector (DLC). The DLC is a 16-pin hardware interface (although only 9 pins are specified by the standard) that is generally found close to the steering wheel (by law it is required to be installed within 0.61 m of the steering wheel) [11]. There are 2 basic types of connectors: Type A as seen in figure 3.2 (using a 12V power supply) and Type B as seen in figure 3.2 (using a 24V power supply). the design of the two connector types prevents the insertion of a type A male plug into a type B female socket.

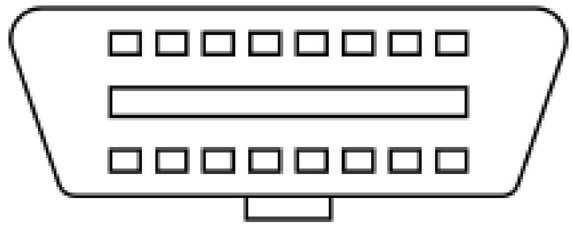

Figure 2: Type A female connector [11]

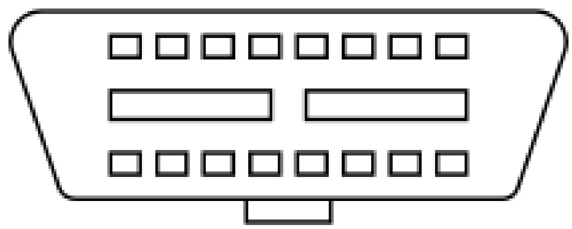

Figure 3: Type B female connector [11]

PID's OBD-II introduces parameter PID's, which are codes used to identify and query specific data. The protocol is designed to work with multiple signalling protocols (the messaging protocol that is used to request and receive data from the network) but the CAN protocol is mostly implemented (Since 2008 all new vehicles sold in the us implement this signalling protocol[13]).

There are multiple ways for a user to interact with this interface:

- Standard Diagnostic scanning tool is a dedicated device that consists of a small hand-held module (equipped with a small screen and some buttons) connected to a male DLC-connector (The DLC inside the vehicle is always female).
- An advanced Diagnostic scanning tool that includes a DLC-connector with wifi/Bluetooth compatibility, allowing for remote diagnostics using a smartphone.

- A DLC-connector with a usb adapter allowing access via dedicated software on a pc. Since 2014 all new cars in the US support the SAE J2534 PassThru standard, which is a Windows API (Application Programming Interface) that provides a standard way to communicate with a cars internal buses [10].[1]
- A data logger, which is designed to capture real-time data while the vehicle is in operation.

Typically the ODB-II is used like this (CAN as signalling protocol): First the user enters the PID of the data he/she wants to query into a diagnostic tool. Secondly this data is packaged in a CAN frame and sent on the CAN-bus. Thirdly the ECU that is responsible for the data identified by the PID in the message recognizes it as it's own, and transmits a CAN frame containing the requested data. Fourthly the diagnostic tool recognizes the response and displays the data to the user. [17] Aside from this the OBD-II port can then be to upgrade the ECU's firmware or to perform a myriad of diagnostic tasks[10].

3.3 CAN

> "Today, CAN has established itself worldwide as the backbone for the networking of embedded systems and this not only in automotive technology"
>
> Dr. Siegfried Dais, Prof. Dr. Uwe Kiencke, Martin Litschel

The CAN protocol has become a ubiquitous part of the automotive industry. In the context of internal vehicle networks, CAN messages have multiple purposes: First there's informative messages that are designed to transmit data from and to ECUs (e.g. the Anti-Lock System (ABS) broadcasting the speed of each wheel). Secondly there's action messages that are designed to request another ECU to perform an action (e.g. daptive Cruise Control (ACC) module requesting the brakes to be applied). Lastly there are the diagnostic messages defined by the OBD-II protocol. [24] Naturally the last type of message is the focus of this paper. The following paragraphs are dedicated to the CAN protocol.

[1]For more information on SAE J2534, see the full API reference at: https://tunertools.com/prodimages/DrewTech/Manuals/PassThru_API-1.pdf

Brief History The history of CAN starts in 1983 when a couple of engineers at Bosch (soon aided by engineers from Mercedes-Benz and Intel) start developing a new serial bus system for use in the auto mobile industry. It wasn't long before CAN was officially introduced at the SAE congress in Detroit as: 'Automotive Serial Controller Area Network'. The main characteristics of this protocol were:

- An arbitration method that allows bus access to the message with the highest priority without delays.
- No master CAN node that is in charge of the bus.
- Transmitted messages are identified by their content, not by their destination or origin.
- This identification also determines the priority of the message within the network.

It didn't take long before the first CAN controller chips were developed in 1987 (by Intel and Philips respectively) and The first official CAN specifications were standardised in the 90's, effectively paving the way for the CAN protocol to become the industry staple it is today. To this day Bosh has been making sure that all CAN chips comply with their proposed standards in order to avoid incompatible implementations. [2].

Architecture A typical CAN network consists of a series of nodes (with a minimal of 2 in order for the network to be functional) connected by a two-wire bus. It is important to note that there are 2 physical CAN specifications: high speed CAN (see [34]) and low speed (or fault tolerant) CAN (see [35]). Every CAN node consists of:

- CPU: effectively the 'brain' of the node, deciding what messages are sent and taking the appropriate course of action whenever a message is received.
- Controller: in charge of reading and writing bits to and from the CAN bus.
- Transceiver: acts as in an intermediate between the bus and the controller, thereby translating between different signal levels.

[2] For a comprehensive history of the CAN protocol confer [19]

This architecture specifies the minimum requirements of a CAN node. More often than not these nodes will include other components (e.g. sensors, actuators) that are connected to the CPU. It should be clear from this specification that this architecture applies to any common vehicle network (e.g. ECU's act as CAN nodes).

CAN Frames Since CAN is a message based protocol, it facilitates communication by transmitting short bursts of data called CAN frames. There are four different types of CAN frames:

- Data frame: used to transmit data with a specific identifier.
- Remote frame: used to request the transmission of data with a specific identifier.
- Error frame: transmitted whenever a node detects an error on the bus.
- Overload frame: transmitted by a node to include a delay between data or remote frame.

There are 2 frame formats: base frame format and the extended frame format. The only difference being that the extended frame format uses a 29 identifier bits and the base frame format only uses 11. Table 1 lists all the fields of a base format data frame. The extended frame format is the same except for an additional identifier field (18 bits) right after the identifier extension bit (IDE) field.

Data Transmission The operation of the CAN protocol is pretty straightforward: a node transmits a message with a specific ID on the bus. Any node that is connected to the same bus is able to receive the message (broadcast), but only the nodes that are listening for this specific ID will take action. It is worth noting that the ID is used to identify the content, not the sender or receiver. As a matter of fact CAN does not provide any way of authenticating the sender or receiver, which results in various security related difficulties (see 3.4) . Aside from identifying the content, this ID is also used to solve the issue of message arbitration. CAN is a carrier sense multiple access protocol, whereby each nodes observes the bus before transmitting data on it, if it detects that the bus is in use it waits for some time before trying again.

[3] A cyclic redundancy check is a way of detecting accidental changes to transmitted data (e.g. due to noise, interference, etc). For more information on cyclic redundancy checking see [22].

name	size (bits)	purpose
start-of-frame	1	Denote start of transmission
identifier (ID)	11	Unique identifier + determines priority
remote transmission request (RTR)	1	must be 1 for remote frames
identifier extension bit (IDE)	1	must be 1 for extended frames
reserved bit	1	reserved for future use
data length code (DLC)	4	length of data field
data field	64	data to be transmitted
cyclic redundancy check (CLC)	15	check for errors [3]
CRC delimiter	1	must be 1
acknowledgement slot (ACK)	1	used for message acknowledgement
ACK delimiter	1	must be 1
end-of-frame (EOF)	7	must be 1

Table 1: base frame format [18]

This does not prevent nodes from starting a data transfer at the same time, this is where bit wise message arbitration provides a solution. [21].

bit wise message arbitration Whenever 2 (or more) nodes initiate a transmission on the bus at the same time, bit wise message arbitration is performed. Every bit of the message ID can be either 1 or 0. The CAN specifications use the term dominant (logical 0) and recessive (logical 1). These terms originate from the fact that whenever more than one bit is simultaneously written to the bus, and one of these is dominant, the dominant bit 'wins', meaning a logical 0 will be seen on the bus. Whenever a node transmits a logical 1 but sees a logical 0, it realizes that there is a contention and re-queues its message for later transmission. Since the identifier is transmitted at the start of the CAN frame, the node with the numerically lowest identifier transmits more zeros at the start of the frame, and that is the node that wins the arbitration. Concisely put we can say that messages with lower ID's have priority over messages with higher ID's. The decision to identify messages by their content (instead of their sender or receiver) is motivated by the fact that certain very important types of messages (e.g. errors) can be given a very low id, thereby ensuring they are less prone to be delayed. This approach does introduce some issues when it comes to security (see 3.4).

Layering In line with most networking protocols, it is common practice to decompose them into different abstract layers. This is done to simplify the design and make modularisation easier [16]. In the case of CAN the layers are:

- **Application layer:** OBD-II, CANOpen, VulCAN etc.
- **Object layer:** message filtering and status handling.
- **Transfer layer:** error detection, message arbitration, bit timing, etc.
- **Physical layer:** signal voltages, pin-out configuration, etc.

For more information on the CAN protocol see [34] and [35].

3.4 OBD-II Security Issues

> "CAN, by design, offers no protection from manipulation"
>
> Dan Klinedinst, Christopher King

It is a well-known fact that the automotive industry has always considered safety a critical engineering concern (especially since the public awareness around lethal accidents has only increased over the years). Unfortunately it is unclear whether developers (especially concerning the internal network) have anticipated an adversary in their design. We known for a fact that the CAN protocol for example did not because of 3 reasons. firstly there is no inherent support for addressing, encryption, authentication, or longer data lengths[24]. The second reason is that most of the networks and ECUs were designed when access to the bus required physical access to the vehicle, therefore security was not a primary concern. The third is that speed and timing are deemed more important to the safety and performance of the vehicle than data security [2]. This vulnerability is worsened by the fact that the attack surface for modern auto mobiles is growing swiftly as more sophisticated services and communications features are incorporated into vehicles [9][10]. The OBD-II specification is one of these since the interface it introduces provides direct access to the internal vehicle network. This allows malicious agents to easily construct and insert CAN messages to alter the vehicles behaviour, as has been frequently demonstrated by Charlie Miller and Chris Valaseks exploits [6][24][25]. Before analysing the attack vectors that OBD-II introduces, and the possible impact thereof on the safety and security of the vehicle, let's first take a closer look at CAN's shortcomings when it comes to safety.

CAN security challenges The CAN protocol has a number of inherent vulnerabilities that are common to any implementation. The most obvious and important ones are:

- **Broadcast nature:** CAN frames are both physically and logically (no destination address) broadcasted on the network. This means that a malicious node on the bus can snoop on all communications or even worse: send packets to any other node on the network [10][9].

- **No authentication:** CAN frames do not have source identifier fields, so there is no way for any node to be aware of the source of any messages it receives. This means that any compromised component (or any other form of unsanctioned access to the CAN bus for that matter) can inject arbitrary messages. Whereas the system has no way of knowing these messages were not sent by the appropriate component[10][9][20].

- **No encryption:** We've mentioned that speed and timing are deemed more important to the safety and performance of the vehicle than data security. A clear result of this is the decision to omit any encryption capabilities. This is because the limited computational power of ECUs makes it difficult to implement robust cryptographic algorithms. [20].

- **Susceptibility to Denial of Service (DoS):** This problem arises mainly from the protocol's message arbitration method. Any malicious node can effectively spam the bus with high priority messages (only zeroes as ID) causing all other nodes to back off (no protection against "babbling idiots"[1])[10][9].

- **Not Byzantine fault tolerant:** In most distributed systems, malicious attacks and software errors can cause a node to exhibit Byzantine (i.e. arbitrary) behaviour[15]. Because of the distributed nature of any CAN system, there is imperfect information on whether a component has failed (or has been compromised by attack) or not. This could result in situations where entire system services fail since a common consensus cannot be reached[14]. [4].

[4] For more information on Byzantine faults, and how it is tolerated in a system see [15] and [14].

OBD-II From the discussion on CAN's security vulnerabilities it is clear that any CAN implementation, and by extension the application layer protocols that run on top of it, are not secure by default. OBD-II is certainly no exception. The OBD-II port can access all CAN buses that make up the vehicle network. This is vital since service technicians should be able to diagnose and update almost any ECU in a vehicle[10][9]. The high level of access technicians have over the vehicle they are diagnosing/repairing is not privileged however. Anyone with physical access to the vehicle, and in possession of the right tools, is granted the same level of access. The problem here is that the system has no way of knowing which OBD-II connections are to be trusted and which aren't, which is exactly the problem this paper attempts to address. Before looking at some example OBD-II exploits, it is a good idea to define a sufficient attacker model first.

Attacker model Here, an attempt is made to define the types of attackers that this paper aims to defend against. a similar classification as [31] and [8] is followed:

- **Insider or outsider:** The insider is considered an interactive member of the network, meaning he/she can communicate with other members freely. The outsider however is limited in the diversity of attacks he/she can mount.

  Classification: **outsider**, since the attacker is not part of the CAN bus. It is worth noting however that when the attacker uses the OBD port to mount an attack, he/she can communicate with the other nodes on the bus. This however is treated by this paper as part of a successful attack, not as an a priori capability of the attacker.

- **Malicious or rational:** A malicious attacker exploits the system for reasons other than personal gain, making them more unpredictable since their motives and resources can vary. A rational attacker however is motivated solely by personal profit, be it money or fame, making them more predictable.

  Classification: **both**, since an attacker using the OBD port as attack vector could be both malicious (e.g. Endangering the life of a rival) and rational (e.g. lowering the internal odometer value before selling the vehicle).

- **Active or passive:** An active attacker is able to generate and transmit messages, whereas a passive attacker is constrained to eavesdropping.

Classification: **active**, since we know the attacker can have access to tools (e.g. PassThru) that grant him/her active capabilities.

- **Local or extended:** This criterion is based on the scope of the attacker. A local attacker has only limited attack vectors, whereas an extended attacker has access to lots.

  Classification: **local**, since a comprehensive analysis of multiple attack vectors is out of scope for this paper (c.f. [1][3][4][6][8][9][10][26][27]).

Example

4 Solution

Quote: The OBD-II port was created to provide consumers with choice and control over their purchase. At the same time, this freedom must be balanced with thoughtful conversations on how to limit adversaries access to vehicle internals, Dan Klinedinst Christopher King

References

[1] Lee Pike, Jamey Sharp, Mark Tullsen, Patrick C. Hickey and James Bielman, *Securing the automobile: A comprehensive approach*, 2015.

[2] Dan Klinedinst, Christopher King, *On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle*, 2016.

[3] Pierre Kleberger, *On Securing the Connected Car*, 2015.

[4] Brian Russell, Aaron Guzman, Paul Lanois, Drew Van Duren, *Observations and Recommendations on Connected Vehicle Security*, Cloud Security Alliance Internet of Things Working group, 2017.

[5] Siam Ahmed, https://www.geotab.com/blog/connected-vehicle-technology/, *Get to Know Connected Vehicle Technology: V2V, V2X, V2I*, 2018.

[6] Charlie Miller, Chris Valasek, *A Survey of Remote Automotive Attack Surfaces*, 2015.

[7] Aastha Yadav, Gaurav Bose, Radhika Bhange, Karan Kapoor, N.Ch.S.N Iyengar, Ronnie D. Caytiles, *Security, Vulnerability and Protection of Vehicular On-board Diagnostics*, 2016.

[8] Jonathan Petit, Steven E Schladover, *Potential Cyberattacks on Automated Vehicles*, IEEE Transactions on Intelligent Transportation Systems September 2014.

[9] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, and Tadayoshi Kohno, *Experimental Security Analysis of a Modern Automobile*, Department of Computer Science and Engineering University of Washington, 2010.

[10] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage, *Experimental Security Analysis of a Modern Automobile*, Department of Computer Science and Engineering University of California San Diego, 2010.

[11] https://en.wikipedia.org/wiki/On-board_diagnostics.

[12] Autotap, *OBDII: Past, Present & Future*, http://www.autotap.com/techlibrary/obdii_past_present_future.asp.

[13] Autotap, http://www.obdii.com/connector.html

[14] https://en.wikipedia.org/wiki/Byzantine_fault_tolerance

[15] Miguel Castro, Barabara Liskov, *Practical Byzantine Fault Tolerance*, in: the Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans,USA,February1999.

[16] https://en.wikipedia.org/wiki/Protocol_stack

[17] https://en.wikipedia.org/wiki/OBD-II_PIDs.

[18] https://en.wikipedia.org/wiki/CAN_bus.

[19] https://www.can-cia.org/can-knowledge/can/can-history/.

[20] Robert Buttigieg, Mario Farrugia and Clyde Meli, Security Issues in Controller Area Networks in Automobiles, University of Malta, 2017.

[21] Abdul Rehwan Anwar, *Bus arbitration in CAN (Controller Area Network)*, https://www.linkedin.com/pulse/bus-arbitration-can-controller-area-network-abdul-rehman-anwer,2017.

[22] https://en.wikipedia.org/wiki/Cyclic_redundancy_check

[23] https://en.wikipedia.org/wiki/Anti-lock_braking _system.

[24] Charlie Miller, Chris Valasek, *CAN Message Injection*, OG Dynamite Edition , 2016.

[25] Charlie Miller, Chris Valasek, *Adventures in Automotive Networks and Control Units.*

[26] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage University of California, San Diego Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno, University of Washington *Comprehensive Experimental Analyses of Automotive Attack Surfaces.*

[27] Stephanie Bayer, Thomas Enderle, Dennis Kengo Oka , Marko Wolf. *Security Crash Test Practical Security Evaluations of Automotive Onboard IT Components*, 2015.

[28] European Union Agency for Network and Information Security (ENISA), *Cyber Security and Resilience of smart cars*, 2016.

[29] Allen Lyons, California Air Resources Board , *On-Board Diagnostics (OBD) Program Overview*, 2015 .

[30] Marien Saarinen, http://www.autoexpress.co.uk/car-news/consumer-news/92304/car-hacking-study-shows-over-100-models-at-risk, *Car hacking: study shows over 100 models at risk*, 2015.

[31] Maxim Raya and Jean-Pierre Hubaux, *Securing vehicular ad hoc networks*, Journal of Computer Security 15, 2007.

[32] *ISO 14229-1:2013: Road vehicles Unified diagnostic services (UDS) Part 1: Specification and requirements.* ISO, 2013.

[33] *ISO 13400-1:2011: Road vehicles Diagnostic communication over Internet Protocol (DoIP) Part 1: General information and use case definition.* ISO, 2011.

[34] *ISO11898-2:2016: Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit.* ISO, 2016.

[35] *ISO11898-3:2006: Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface.* ISO, 2006.

[36] Alexei Sintsov, *(pen)testing vehicles with CANToolz*, 2016.

[37] Philip Koopman. *A case study of toyota unintended acceleration and software safety.* Public seminar, September 2014.

[38] *Guidelines for the Use of the C Language in Critical Systems.* MISRA, 2004.

[39] Koen Claessen and John Hughes. *QuickCheck: A lightweight tool for random testing of haskell programs.* In Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP 00, pages 268279. ACM, 2000.

[40] Bart Jacobs, Frank Piessens, *The VeriFast Program Verifier*, Department of Computer Science, Katholieke Universiteit Leuven, Belgium.

[41] Olaf Pfeiffer, Christian Keydel, *Scalable CAN security for CAN, CANopen and other protocols*, 2017.

[42] Andreea-Ina Radu and Flavio D. Garcia School of Computer Science, University of Birmingham, UK, *LeiA: A Lightweight Authentication Protocol for CAN*,

[43] Stefan N urnberger, Christian Rossow CISPA, Saarland University, Germany *VatiCAN Vetted authenticated CAN bus*

[44] Job Noorman, Pieter Agten ,Wilfried Daniels ,Raoul Strackx Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Frank Piessens ,Ingrid Verbauwhede, *Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base*, iMinds-DistriNet and iMinds-COSIC, KU Leuven.

[45] Jan Tobias Muhlberg, Frank Piessens, Jo Van Bulck, *VulCAN: E icient Component Authentication and So ware Isolation for Automotive Control Networks*, Imec-Distrinet KuLeuven, 2017.

[46] Andreea-Ina Radu, Flavio D. Garcia, *LeiA: A Lightweight Authentication Protocol forCAN*, School of Computer Science,University of Birmingham, UK.

[47] Stefan N urnberger, Christian Rossow, *vatiCAN, Vetted, Authenticated CAN Bus*, CISPA, Saarland University, Germany.

[48] ISO-26262-1, Road vehicles Functional Safety, ISO 26262-1, International Or- ganization for Standardization, Geneva, Switzerland, 2011.

[49] Cesar Bernardini, Muhammad Rizwan Asghar, Bruno Crispo, *Security and privacy in vehicular communications: Challenges and opportunities*, 2017.