Development of a Man-in-the-Middle Attack Device for the CAN Bus

András Gazdag[a], Csongor Ferenczi[b], Levente Buttyán[c]

[a]CrySyS Lab, BME-HIT
agazdag@crysys.hu

[b]CrySyS Lab, BME-HIT
csferenczi@crysys.hu

[c]CrySyS Lab, BME-HIT
buttyan@crysys.hu

Conference on Information Technology and Data Science
November 6–8, 2020
Debrecen, Hungary

Abstract

Modern vehicles are full of embedded controllers called ECUs (Electronic Control Units). They are responsible for different functionalities involving processing information from sensors and controlling actuators. To perform their functions, ECUs also need to communicate with each other. Most vehicles use a Controller Area Network (CAN) for ECU communication. The original design of the CAN bus was focusing on safety and reliability properties. Security was not an issue because these networks were considered to be isolated systems. These assumptions were correct for a long time, but they no longer hold. Modern vehicles have many interfaces towards the outside world, which renders the internal network accessible to an attacker. Bluetooth, Wifi, wireless Tire Pressure Monitoring System (TPMS), or the On-Board Diagnostics (OBD) port are all options for attackers to either directly access the CAN network or compromise a component attached to it. It is possible to inject fake messages, or potentially, to modify messages on the CAN, and hence, forcing some ECUs to act upon these fake messages, which may influence the overall behavior of the vehicle.

Modification attacks are complex both to carry out and to detect. The main difficulty of modification attacks is that the sender checks whether the transmitted bits correctly appear on the bus or not for safety reasons. The only network level way to circumvent this protection is to physically separate the sender and the attacked ECU on the CAN bus. This can be achieved with a physical layer Man-in-the-Middle attack. We built a proof-of-concept hardware device capable of modifying the CAN traffic in real-time to demonstrate that this attack is possible. It has two CAN interfaces to read messages from the original CAN bus and either just forward or modify-and-forward traffic to the attacked CAN bus. We showed with measurements that we can perform a message modification attack while keeping the introduced delay within what is allowed by the CAN specification.

Keywords: Vehicle Security, CAN, ISO 11898, Man-in-the-Middle attack

1. Introduction

Modern cars are not just simple mechanical devices with an engine and four wheels like they used to be. They are full of Electronic Control Units (ECU) that control the car, monitor the drivers' every movement, and try to keep them safe on the road, while providing convenience features to them. Thus, more and more components have to communicate with each other in a car, which means that we need reliable communication channels between these components. One of the most common communication solutions is the Controller Area Network (CAN) bus.

When it comes to manufacturers, they put a great emphasis on the safety of the cars. There are ABS, ESP, crash avoidance systems, or even better and better crumple zones in every car. The inter-component communication channels are protected against errors caused by the high noise environment, but security seems to be less important for manufacturers. The internal components are rarely designed to be secure from a potential attacker, although it is not unthinkable that a malicious device could be installed into a car and do some harm. For instance, it only takes one mechanic to plug such a device into an On-Board Diagnostics (OBD) port of a targeted car, and we have access to the vehicle's CAN bus. Moreover, with the spread of Bluetooth OBD debugging probes, which can connect to the owner's smartphone, users themselves connect more and more, potentially compromised, devices onto the CAN bus.

Unfortunately, the CAN bus has no security, only safety measures. It was designed to be a robust communication bus that can withstand a high amount of noise while providing relatively high transfer speeds. At the end of every CAN message, a Cyclic Redundancy Check (CRC) code ensures that if the content of the message changes during transmission, the receiver will detect it. However, the CAN standard does not provide support for message authentication. Thus, just by receiving a message with a given ID does not guarantee that the source of the message was any particular device. This leads to the issue that messages can be injected onto the bus without the communication partners ever noticing that a message was not sent by the correct device and could contain false data.

Besides injecting messages, an attacker could modify the content of the message, but achieving this is much harder. With the current techniques, it is only feasible by compromising an ECU, and sending messages with modified values. This attack produces no extra messages, nor highly deviating values; thus, it is much harder to detect than simple message injection attacks, but until now, it required a more in-depth knowledge[1] to carry out than just plugging in a device to the OBD port. In this paper, we present our solution to modify CAN messages in real-time, without compromising any ECU.

We have created a device, which is capable of modifying ISO 11898 high-speed CAN messages in real-time. It can handle up to 100% bus load with a bus speed of 500 kbps or less, and about 60-100% busload at 1 Mbps, which is the maximum speed specified in the standard. It introduces a delay of 260 μs, which is even at 1 Mbps is well within the delay of re-sending a message due to high traffic or a transient error on the bus. The device itself provides a wireless interface that can be used to remotely configure the attack parameters. All this, while consisting of cheap, commonly available parts.

In Chapter 2, we are going to take a deeper look at the current attacks against the CAN bus and their mitigations, and how our solution differs from them. In Chapter 3, we summarize the important parts of the CAN standard. In Chapter 4, we discuss the design principles of the introduced device, the hardware and software architecture, and the operation of the device. In Chapter 5, we take a look at the measurements of the device, and finally, in Chapter 6, we conclude our work.

2. Related work

There is a large amount of published research on the (in)security of the CAN bus. Many of them propose a solution to protect the CAN bus [12][11], but there are several papers discussing different kinds of attacks against it [9][3]. We can generally categorize the attacks into two groups: message injection and message modification attacks.

2.1. Attacks against the CAN bus

2.1.1. Message injection attacks

Message injection attacks [8] are possible due to various properties of the CAN bus. First of all, since the CAN bus is a broadcast channel, during operation, every ECU connected to the CAN bus receives all of the messages. This cost-saving measure saves a lot of money for the manufacturers, since they only have to wire a few twisted pair cables throughout the whole car in order to connect the ECUs to each other. When receiving a message, the ECUs decide whether they are interested in the given message based on its ID and process it or discard it. On the other hand, this principle also makes it easy for an attacker to eavesdrop messages on the bus,

[1] `http://illmatics.com/remote\%20attack\%20surfaces.pdf`

monitor the values of the sensors in real-time, or reverse engineer the messages and their purposes for a given vehicle type. The second issue with the standard is that the CAN bus does not support any kind of cryptographic message authentication measures. Any ECU can create a message with an arbitrary ID and send it to the other ECUs via the CAN bus, and the ECUs will not be able to differentiate the messages coming from two different ECUs, if they have the same ID. This makes an attacker able to craft arbitrary messages and send them to the ECUs via the CAN bus. An attacker can have multiple goals to exploit these properties, for instance, overwriting values or forcing the network into a Denial of Service attack state.

Message injection attacks have several drawbacks. First of all, both the rapidly changing value behaviour and the at least doubled message periodicity is easily detectable [4]. Upon detection, the targeted ECU might switch into a fallback mode, where it ignores both the original and the injected values. Secondly, the ECU might have safety margins built-in. For instance, most, if not all of the lane assist systems have a maximum steering angle it is allowed to perform in order to keep the car in the lane. If the attacker tries to induce a higher steering angle than this maximum value, the lane assist system might just deactivate.

2.1.2. Message modification attacks

Message modification attacks are based on the idea that instead of injecting messages, one could modify the message that is being sent. There are several safety measures in the CAN standard that makes it hard if not impossible to modify messages transmitted by another ECU on-the-fly (e.g., CRC, bit stuffing, sender ECU monitoring the bus during transmission, etc.). Thus, until now, the easiest way to realize this attack was to compromise an ECU [10] and modify the messages before they are even sent.

2.1.3. Denial of Service (DoS)

In a DoS attack, the attacker's goal is to render a given CAN bus unusable, which can be achieved in two ways: by adhering to the CAN standard or by breaking it. The first option is to send as many messages to the CAN bus with the lowest possible ID as physically possible. When the bus is idle, if two or more ECUs want to transmit at the same time, the one with the lowest ID will have priority. Thus, since the zero ID has priority over every other message ID, none of the regular messages will win the arbitration against the injected message, which will lead to the starvation of the regular ECUs. The second method is to force the CAN_H and CAN_L wires into dominant state and hold it there as long as the attacker wants to. While the second method could be easier to implement, it triggers the error detection in the ECUs; thus, the connected subsystem will detect that there is an error with the CAN bus. On the other hand, the first solution does adhere to the rules of the CAN, and the ECUs might only think that everything is okay with the CAN bus, except it is busy at the moment. Nonetheless, using both solutions, an attacker can render a given CAN bus unusable.

2.2. Attack mitigations

There are several proposed solutions to the issues mentioned in the previous section. However, while the following measures could increase the security of the CAN bus, they are more theoretical than practically applicable solution.

2.2.1. CANAuth

As mentioned before, one of the biggest deficiencies of the CAN bus is that it lacks message authentication. CANAuth [5] solves this issue by using a symmetric key based HMAC. The main idea behind the CANAuth is the following. As described in section 3.1, every bit transmitted on the CAN bus get sampled at the 75% percentile point of the bit time to ensure a reliable sampling. However, technology has developed a lot since the introduction of the CAN bus, and microelectronics are much faster nowadays. Thus we could use a higher sampling rate in order to hide authentication data in the propagation segment of every CAN bit.

2.2.2. CAN firewall

Another mitigation method is to use firewalls [1]. By using a firewall, we can physically split a CAN bus into multiple separate segments and control the traffic going between them. For instance, we can apply whitelist or blacklist-based message filtering on each of the different segments, introduce rate limiting, etc. Thus, we can limit the possibility of message injection and DoS attacks. Adding a CAN firewall to an existing car should not require redesigning the car, since the firewall itself is just a simple device with two CAN interfaces, which can be inserted between the separated CAN segments.

While this solution could appear as the ultimate solution to the shortcomings of the CAN standard, it has several issues. For instance, we have to separate every important ECU or at least every important segment with a firewall, in order to be effective, which creates an excess cost for the manufacturer. An even bigger issue is the management of the firewall. Who gets to write the filtering rules? How are the rules updated, if an update is required? Who is responsible if an important packet gets dropped unintentionally? What if an airbag does not open during an accident due to a malformed firewall rule? While these edge cases could seem unimportant, not being able to address them satisfactorily may make manufacturers deciding not to use this technology.

2.2.3. Secure CAN transceivers

A third mitigation technique is the secure CAN transceivers [2] proposed by NXP. Their idea is to introduce a new firewall-like security defense layer at the CAN transceiver level. Using this new layer, they can prevent message spoofing in both the transmitting and the receiving side; i.e., detect malicious ECUs and evade DoS attacks.

2.2.4. Mitigations in practise

Based on the mitigation techniques introduced in the previous subsections, we can say that while these mitigations are great theoretical results, and could more or less enhance the security of the CAN bus, we have to emphasize that they are still not widely used in practice, and thus, do not provide protection against current attacks. NXP's Secure CAN transceivers could be the next easy to implement security measure, but our solution introduced in this paper could still circumvent it after a minor modification.

3. The ISO 11898 High-speed CAN standard

The CAN bus[6, 7] is a broadcast, serial communication protocol used mainly in vehicles. It was designed to be robust, withstand high external RF noise, while providing a high-speed communication link between the ECUs. The CAN bus is a cost-effective solution, because it enables the manufacturers to connect the ECUs by placing only one twisted pair cable between them.

The CAN bus uses two wires called CAN high (CAN_H) and CAN low (CAN_L), in order to implement differential signalling. The cables are twisted pair cables. On each end of the cables, the wires are connected to each other using a terminating resistor in order to achieve a nominal 120 Ohm impedance. The CAN bus has two states, driven, and not driven. When it is not driven, the CAN_H and CAN_L wires get pulled to about the same 2.5V nominal voltage using the passive pull resistors placed in the CAN transceivers. This state is also called a "recessive" bit, which represents a binary 1. When the CAN bus is driven, at least one CAN node pulls the CAN_H wire to 3.5V and CAN_L wire to 1.5V nominal voltage. This state is also called a "dominant" bit, which represents a binary 0.

3.1. Bit Timing

Every CAN bus has a nominal bitrate, which gets preconfigured in the ECUs by the car manufacturer. The maximum bitrate specified by the standard is 1 Mbps, but 500 kbps and 250 kbps is also frequently used bitrates.

Every bit time can be divided into the following four segments (Figure 1):

- *synchronization segment (Sync_Seg):* This segment is used for synchronization. At the SOF, every receiving ECU synchronizes itself to the edge of the first bit, which is called a "hard sync". There is also another kind of synchronization called bit resynchronization, which is performed at the synchronization segment of each bit, by the ECU fine-tuning its inner clock based on the deviance between the expected time of a potential edge, and the actual time of its detection.

- *propagation time segment (Prop_Seg):* This segment is used to compensate for physical delay times within the network (e.g., signal propagation time, internal delay of the ECUs, etc.)

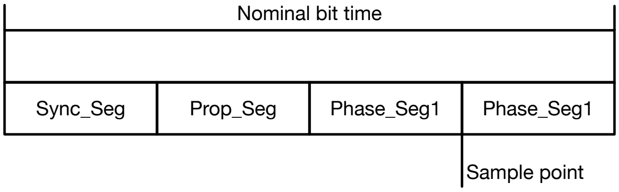


Figure 1: The four segments of the bit time.

- *phase buffer segment 1 (Phase_ Seg1) and Phase buffer segment 2 (Phase_ Seg2):* These segments are used for edge phase error compensation. The sampling of the bus occurs after the phase buffer segment 1. The length of these segments can be fine-tuned by resynchronization, and thus, the sampling point can be moved backward or forward.

3.2. Frame timing

Messages can be transmitted based on events (e.g., receiving a remote frame, a sensor value triggering a transmission) or by a trigger coming from an internal timer. The internal timer method is called Time-Triggered Communication (TTC). During communication, after every frame's EOF segment, there is a 3-bit long intermission period. After these 3 recessive bits, the bus is considered idle, and any following dominant is considered as the SOF of the next frame. When the ECUs detects that the bus is idle, any of them may start to transmit. If two or more ECUs happen to transmit at nearly the same time, the conflict between them is resolved using contention-based arbitration.

3.3. Contention-based arbitration

When a node starts to transmit a frame, it monitors the bus during the arbitration segment of the MAC frame in order to check whether the data on the bus is the same as it is transmitting. In case it detects that despite transmitting a recessive bit, the bus is still in dominant state, it knows that another ECU tries to transmit data, and it terminates the arbitration process and turns into a receiver. This allows the other ECU to transmit its message as nothing happened, and the other ECU that lost the arbitration, can retry sending its message at a later time. Using this method requires the bus has to adhere to three key elements:

- The ID of the message types has to be unique.
- A data frame with a given ID and a non-zero Data Length Code (DLC) value may only be sent by one ECU.

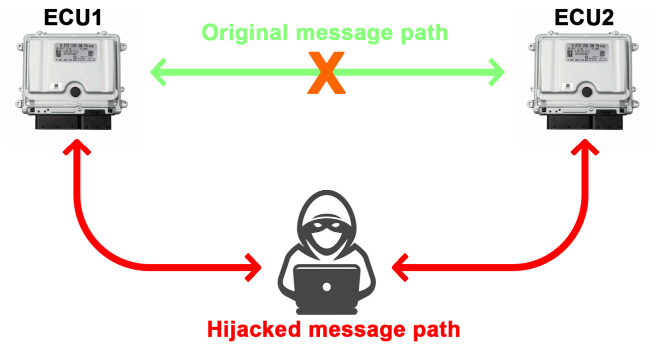


Figure 2: The concept of a MITM attack in the automotive industry.

- The remote frame's DLC value has to be the same as the data frame it requests.

Using this contention-based arbitration ensures that the ECU with the higher priority frame will always win the arbitration, because its ID contains more dominant bits, than the other potential transmitter ECU frame's ID.

4. Malicious CAN gateway

Since the CAN standard does not support message authentication, if we manage to insert a special device in the communication line between the two ECUs, then we can add, modify, or even delete messages with an arbitrary ID. The visualization of the attack can be seen in Figure 2.

4.1. Design

In order to create a CAN gateway, we used NodeMCU ESP32s microcontrollers. ESP32 is a two-core, 240MHz versatile microcontroller, with a built-in CAN controller and Wi-Fi module. It comes with the Espressif IoT Development Framework (ESP-IDF), which is based on the popular Free Real-Time OS (FreeRTOS) platform. Using the ESP-IDF allows us to create a hard-real-time device using its built-in scheduler, while providing convenient features, for instance, a built-in web server, which is useful for device configuration.

There were several requirements for the proof-of-concept malicious CAN gateway:

- It shall be able to handle ISO 11898 High-Speed CAN messages, with a bus speed of up to 1Mbps.
- It shall have as low introduced delay as possible, preferably low enough that it is not significantly greater than a lost arbitration or a transient error.

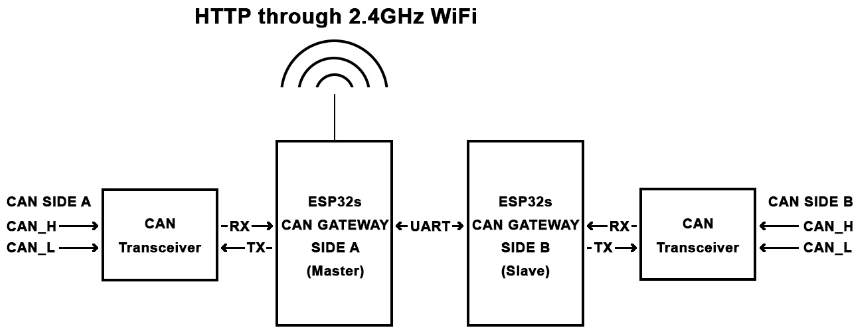


Figure 3: High-level architecture of the CAN gateway.

- Despite being a proof-of-concept device:
  - It shall be robust enough to be used later at least for research.
  - It shall be easy to configure during testing, and configuration shall not require reprogramming of the device.
  - It shall be as serviceable as possible, without requiring specific hardware knowledge. The components shall be replaceable in case it is required.

The first step of the design process was to create a high-level architecture of the CAN gateway, which can be seen in Figure 3. As one can see on the figure, the CAN gateway consists of two CAN transceivers, and two ESP32s microcontrollers. One of the microcontrollers is the master, and the other one is the slave. Each microcontroller handles one side of the CAN bus via its built-in CAN controller, and the corresponding CAN transceiver. The communication between the microcontrollers is realized via a UART line. The master provides a web interface for configuration through its 2.4GHz Wi-Fi module, working as an access point.

4.2. Implementation

After designing the schematics and verifying it on a breadboard, we have built a soldered version of the circuit. We used a protoboard as the base of the device, which allowed us to apply minor changes to the hardware design without having to rebuild the complete circuit again. The components have been soldered onto the protoboard using sockets in order to make a potential component replacement easier to achieve. A picture of the finished proof-of-concept CAN gateway board can be seen on Figure 4.

4.3. Operation

After powering up the device, it provides a web interface for configuration through its Wi-Fi access point, as we mentioned before, which is a key element in making the

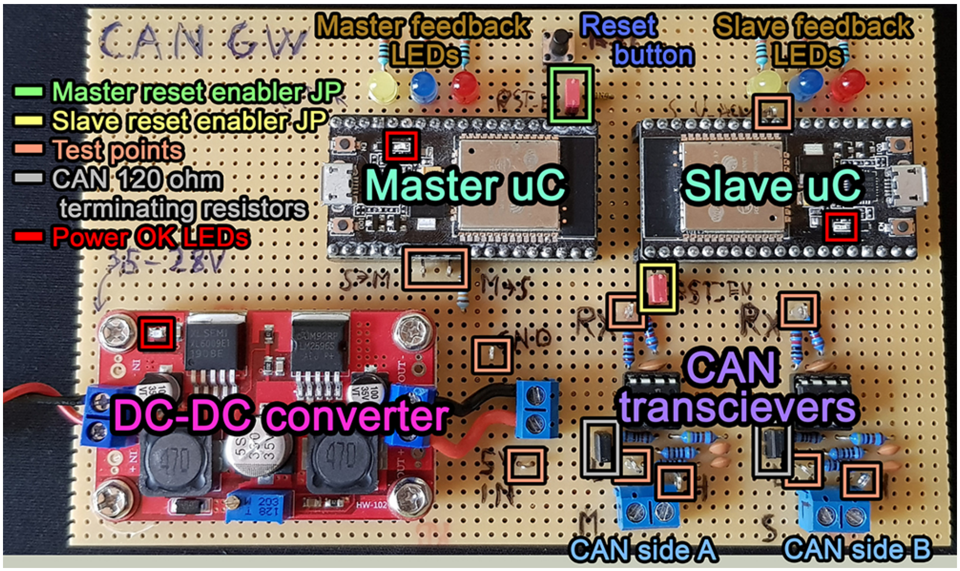
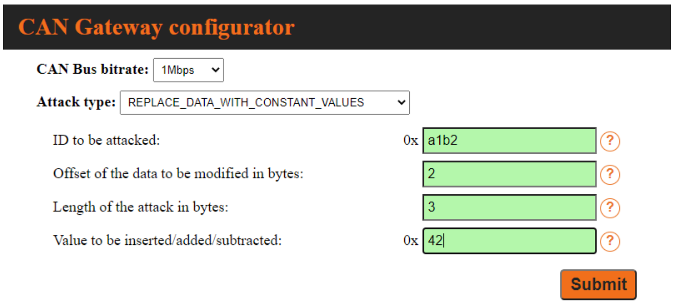


Figure 4: The top view of the proof-of-concept CAN gateway board.


CAN Gateway configurator

CAN Bus bitrate: 1Mbps

Attack type: REPLACE_DATA_WITH_CONSTANT_VALUES

ID to be attacked:	0x a1b2
Offset of the data to be modified in bytes:	2
Length of the attack in bytes:	3
Value to be inserted/added/subtracted:	0x 42

Figure 5: The graphical web configuration interface of the CAN gateway.

device usable for research purposes. This web interface allows the user to configure it using a simple web browser, or via direct POST request (e.g., using curl). The graphical web configuration interface of the device can be seen on Figure 5. The configurable parameters are the following:

- *Bitrate:* This parameter can set the bitrate of the CAN bus.
- *Id:* The id of the CAN message to be attacked.
- *Offset and AttackLength:* The offset controls the position of the first byte to be attacked, and the attackLength determines how many bytes will be attacked.
- *ByteValue:* Some of the attack types require an additional parameter, which will replace the original or will be added or subtracted from the selected byte values.
- *AttackType:* There are several different attacks the device can perform:
  - `Passthrough`: In this mode, the device relays the traffic without modifying any of the messages.
  - `Replace-data-with-constant-values`: In this mode, the device replaces the selected bytes in the message with the given ByteValue parameter.
  - `Replace-data-with-random-values`: In this mode, the device generates random bytes for each of the selected bytes in the message, and replaces them.
  - `Add-delta-value-to-data`: In this mode, we add the given ByteValue parameter to each of the selected bytes in the message. In case the resulting values would overflow, it gets capped at the maximum 255 value.
  - `Subtract-delta-value-from-data`: Similar to the previous attack type, but the byte ByteValue is subtracted instead of added. In case the resulting value would underflow, it gets bounded at the minimum 0 value.
  - `Increase-data-until-max-value`: In this mode, we take the lowest value from the selected bytes, increase it by one and replace all of the selected bytes if the increased byte is higher than the original. This is repeated until the max value (0xff) is reached.
  - `Decrease-data-until-min-value`: This attack type is similar to the previous one, but at the start, we take the highest value from the selected bytes and decrease it every message, until we reach 0x00.
  - `Replace-data-with-increasing-counter`: We start a counter from 0 and increase it by one at every occurrence of the message. The selected bytes get replaced with the counter. The counter can overflow.

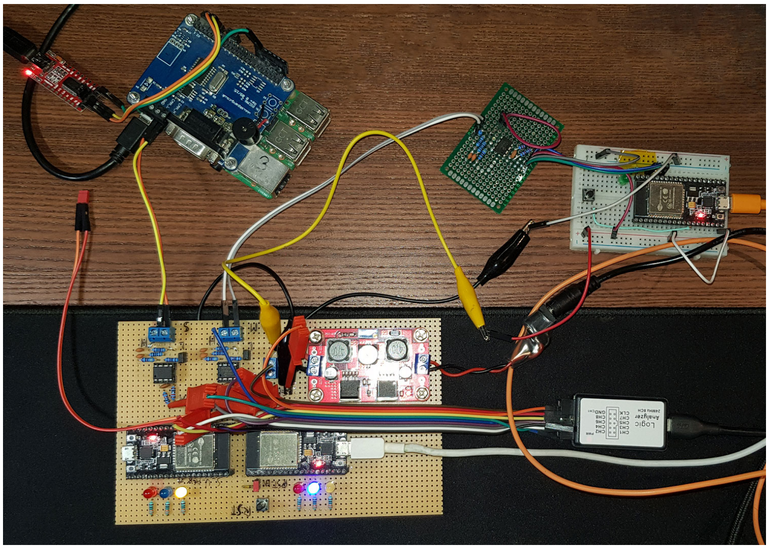


Figure 6: The CAN gateway testbed using a Raspberry Pi and a NodeMCU ESP32s.

– `Replace-data-with-decreasing-counter`: Similar to the previous attack type, but the selected bytes get replaced by a decreasing counter starting from 255, which can underflow.

After the user sends the configuration via the graphical web interface or via a direct POST request, the device validates the configuration on the server-side, and if it is correct, it starts the attack phase.

5. Evaluation

During the evaluation, we performed two tests. First, we created a testbed using a Raspberry Pi with a PiCAN shield; and a NodeMCU ESP32s with an additional CAN Transceiver as our CAN nodes. Later, we used a vehicle testbed built from actual vehicle components to verify the functionalities of our device in a close-to-real-world setting.

5.1. Raspberry Pi testbed

The nodes were configured to send messages to each other via the CAN bus; however, they could only send these messages through the CAN gateway. The testbed can be seen on Figure 6.

During the measurements, we tested all attack types, with different IDs, offsets, and attack lengths, while logging both the UART lines, as well as the messages on both sides of the gateway. As we found, the CAN gateway managed to modify the messages with an introduced delay of 260us on the 1Mbps CAN bus. This delay is only approximately 2.3 times longer than a lost arbitration, which could be caused by a busy bus or a short transient fault.

One example of the tested attacks has the following parameters:

- *Bitrate:* 1 Mbps
- *Id:* 0x090
- *Offset:* 2
- *AttackLength:* 3
- *ByteValue:* 0x08
- *AttackType:* `Replace-data-with-constant-values`

A screenshot of a measurement can be seen on Figure 7. There are two messages traveling on the bus at the same time:

```
A→B: ID: 0x090, Data:  0x00 0x80 0x80 0x80 0x41 0x41 0x00
B→A: ID: 0x045, Data:  0x01 0xf2 0x03 0xf4 0x05 0xf6 0x07 0xf8
```

However, after both messages go through the CAN gateway, the targeted message with the 0x090 ID arrives with changed values. On the arriving side, the following messages are present:

```
A→B: ID: 0x090, Data:  0x00 0x80 0x08 0x08 0x08 0x41 0x00
B→A: ID: 0x045, Data:  0x01 0xf2 0x03 0xf4 0x05 0xf6 0x07 0xf8
```

Thus, we can say that the CAN gateway has successfully modified the preconfigured part of the message.

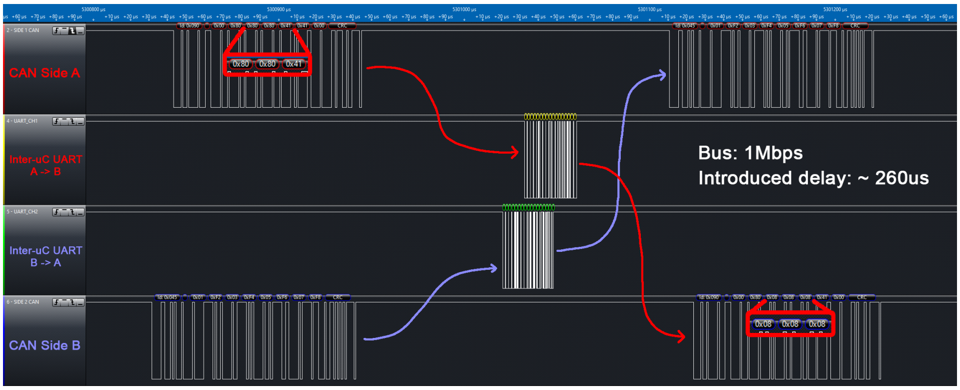


Figure 7: Simultaneous message transmission in both direction.

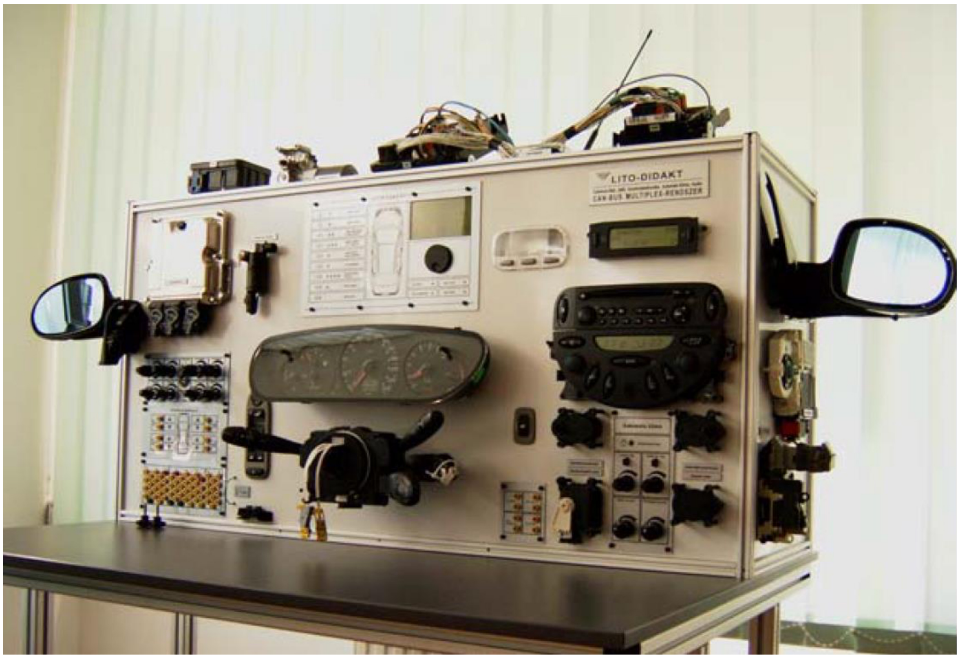


Figure 8: The Citroen C5 test bench.

5.2. Vehicle testbed

After verifying that our attack worked in our local testbed, we tested the malicious CAN gateway on a Citroen C5 test bench, which contains the electronics of a real-life Citroen C5 (Figure 8).

We found a CAN bus connection point between the dashboard and the ECU, we connected the CAN gateway to this point. The attack we performed was to overwrite the tachometer value with a constant, and thus force the dash to show a modified engine rpm instead of the real one. The attack parameters were the following:

- *Bitrate:* 250 kbps
- *Id:* 0x208
- *Offset:* 0
- *AttackLength:* 1
- *ByteValue:* 0x30
- *AttackType:* `Replace-data-with-constant-values`

As one can see in Figure 9, despite the engine idling at 810 rpm, the dashboard shows the modified values of around 1500 rpm.

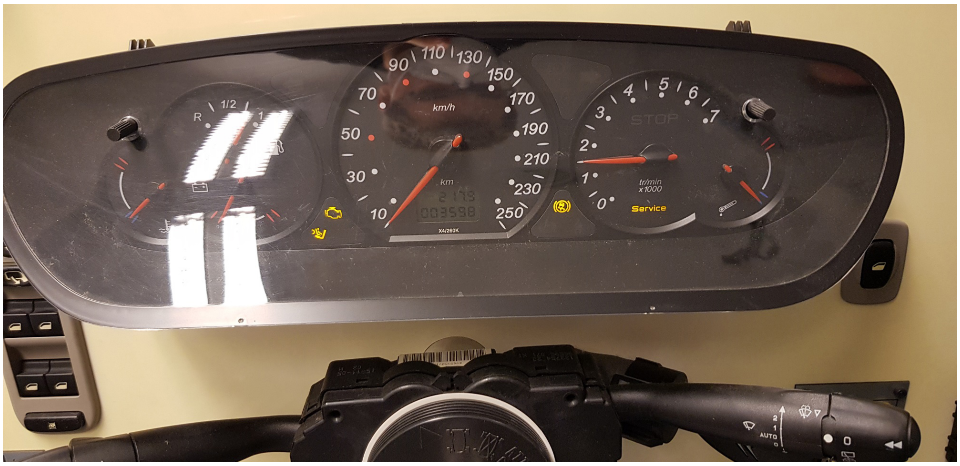


Figure 9: The attack of the tachometer displays different rpm than the real one.

6. Conclusion

In summary, the CAN bus, which is one of the most common communication solutions for ECUs, has several security flaws, since security was not in focus during its development. There are no message authentication measures; the ECUs decide to act upon a message or not based only on the CAN ID field of the message. Hence, it is possible to inject fake messages, or modify existing messages on the CAN, and by doing that, to force some ECUs to act upon these fake messages, which may influence the vehicle's overall behavior.

While message injection attacks are easy to implement, they provide several side effects, making them almost trivial to detect. Message modification attacks are hard to realize and require a more profound knowledge of the field, but they are much less detectable. In this work, our goal was to design and implement a device, capable of modifying ISO 11898 high-speed CAN messages in real-time.

Our CAN gateway device was designed to perform a Man-in-the-Middle attack by separating the targeted ECU and the rest of the CAN bus. Being in a man-in-the-middle position allows our device to modify the content of any message passing through the CAN gateway without any excess message or increase in the busload. It is capable of modifying CAN messages in real-time with a minuscule introduced delay of $260\,\mu$s, without being detectable by current measures. It can handle up to 100% bus load with a bus speed of 500 kbps or less, and about 60-100% busload at 1 Mbps. The device is built from low-cost, commonly available parts, and it provides a wireless interface that can be used to remotely configure the attack parameters.

Acknowledgements

This work has been partially funded by the European Commission via the H2020-ECSEL-2017 project SECREDAS (Grant Agreement no. 783119) and partially supported from the grant GINOP- 2.1.1-15 (the project has been supported by the European Union, co-financed by the European Social Fund. EFOP-3.6.2-16-2017-00002). The research presented in this paper have also been supported by the NRDI Office, Ministry of Innovation and Technology, Hungary, within the framework of the Autonomous Systems National Laboratory Programme, and the NRDI Fund based on the charter of bolster issued by the NRDI Office.

References

[1] Arilou: *Feasible car cyber defense*, ESCAR, 2010.

[2] Bernd Elend, Thierry Walrant, Georg Olma: *Securing CAN Communication Efficiently With Minimal System Impact*, NXP, 2020,
url: `https://www.nxp.com/docs/en/white-paper/SECURECARTRANA4FS.pdf`.

[3] E. Evenchick: *Hopping On the CAN Bus*, Black Hat Asia, 2015,
url: `https://www.blackhat.com/docs/asia-15/materials/asia-15-Evenchick-Hopping-On-The-Can-Bus.pdf`.

[4] András Gazdag, Dóra Neubrandt, Levente Buttyán, Zsolt Szalay: *Detection of Injection Attacks in Compressed CAN Traffic Logs*, in: International Workshop on Cyber Security for Intelligent Transportation Systems, Held in Conjunction with ESORICS 2018, Springer, 2018.

[5] Anthony Herrewege, Dave Singelée, Ingrid Verbauwhede: *CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus*, in: Jan. 2011, p. 7.

[6] *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*, Standard, Geneva, CH: International Organization for Standardization, Dec. 2015.

[7] *Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit*, Standard, Geneva, CH: International Organization for Standardization, Dec. 2016.

[8] K. Koscher, A. Czeskis, F. Roesner, et al.: *Experimental Security Analysis of a Modern Automobile*, in: 2010 IEEE Symposium on Security and Privacy, 2010, pp. 447–462,
doi: `10.1109/SP.2010.34`.

[9] C. Miller, C. Valasek: *Adventures in Automotive Networks and Control Units*, IOActive Labs Research, 2013,
url: `https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf`.

[10] C. Miller, C. Valasek: *Remote Exploitation of an Unaltered Passenger Vehicle*, IOActive Labs Research, 2015,
url: `https://ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf`.

[11] H. M. Song, H. R. Kim, H. K. Kim: *Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network*, in: 2016 International Conference on Information Networking (ICOIN), 2016, pp. 63–68,
doi: `10.1109/ICOIN.2016.7427089`.

[12] A. Taylor, N. Japkowicz, S. Leblanc: *Frequency-based anomaly detection for the automotive CAN bus*, in: 2015 World Congress on Industrial Control Systems Security (WCICSS), 2015, pp. 45–49,
doi: `10.1109/WCICSS.2015.7420322`.