CANIS
AUTOMOTIVE LABS
Security Products for the Automotive Industry

CAN Bus Security
Attacks on CAN bus and their mitigations

Dr. Ken Tindell, CTO Canis Automotive Labs
ken.tindell@canislabs.com

Document number	1901
Version	07
Issue date	2020-02-14

1 Introduction

1.1 Overview

CAN is a protocol more than 30 years old yet is still in widespread use in automotive, aerospace, marine, space and many other industries. This is because of its high reliability and low cost. It is a protocol that is almost perfectly designed for communicating sensor and actuator command data with short real-time latencies and in an atomic fashion (i.e. all receivers see the data, or none do). Achieving these properties is still a challenge for mainstream IT communications protocols such as Ethernet. But CAN was not designed with security in mind. Systems today are increasingly being connected to the Internet and at the same time malefactors are becoming increasingly sophisticated.

Connected to CAN are the devices that if deliberately attacked could result in damage or injury (for example, airbags are detonated at end-of-life using diagnostic commands sent over CAN). Guarding the CAN bus is therefore vital.

There are several aspects to CAN security:

- How is the CAN bus accessed by a malefactor?
- What can they do with access?
- What are the strategies for mitigating these threats?

The rest of this section gives an overview of how CAN bus can be accessed and a brief overview of the types of attack on CAN and the categories of mitigations

Section 2 describes attacks on CAN in detail. This covers spoofing and denial-of-service attacks and gives examples of low-level attacks on the CAN protocol itself.

Mitigation techniques are then discussed in detail. Section 3 describes what intrusion detection systems (IDS) can achieve. Section 4 examines CAN security gateways. Section 5 explains the issues surrounding the use of encryption on CAN. Section 6 describes the latest innovations in hardware to protect the CAN protocol.

Finally, Section 7 summarises the issues of CAN security with a comparison of the mitigation techniques and offers recommendations.

1.2 Access to the CAN bus

There are four broad ways to gain access to CAN bus:

- **Using the ODB-II connector**. This is standard on nearly all passenger cars and has the CAN H and CAN L twisted pair connectors available, as shown below.

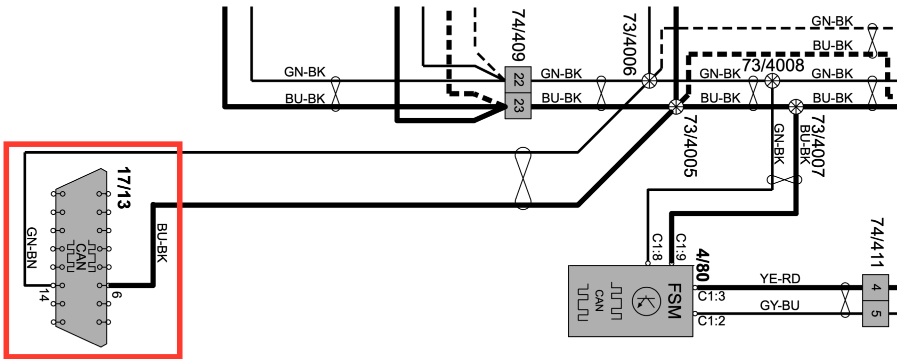


Figure 1: example of ODB-II connector in a vehicle wiring diagram

  Plugged into this connector is typically a 'dongle' that contains a CAN transceiver (to translate the CAN differential voltages into logic 0 and 1), a CAN controller (to convert between a stream of logic bits and CAN frames), and a communications interface to communicate to something else (often Bluetooth to communicate with a nearby laptop or a phone, sometimes a cellular modem).

  There may be non-malign reasons for accessing the CAN bus via a dongle. For example, insurance companies are increasingly using these to provide services to drivers. But the dongle connected to an external device creates further ways to the CAN bus: malware on a connected laptop or phone or in servers connecting to the dongle, or even vulnerabilities in the dongle firmware that can be exploited by connecting through a modem.

- **Through the wiring harness**. By knowing where the CAN bus wires are routed it is relatively straightforward to splice into the CAN H and CAN L wires. This might be done to retrofit some piece of equipment (there are kits to turn lane-keeping assistance mechatronics into an pseudo self-driving system by adding cameras and software running Machine Learning algorithms) or it might be a criminal attempting to override the anti-theft systems by making a hole in the body panels to get directly at the wiring.

- **Through the infotainment system**. Most vehicle infotainment systems are built from large software stacks over a desktop OS (e.g. Linux). The huge complexity of the software has led to a huge number of vulnerabilities that could compromise a device. These vary from local attacks (where a USB stick with a targeted virus can hijack the infotainment system by being plugged in) to nearby attacks (e.g. exploiting vulnerabilities in the WiFi, Bluetooth, RDS/TMC or DAB radio software stacks) to remote attacks over the internet (e.g. carefully manipulated fonts in a web page designed to exploit a vulnerability bug in an in-car browser).

- **Through a hijacked electronic control unit** (ECU). An individual ECU can be hijacked by programming errors in its firmware, such as vulnerabilities in the diagnostics software stack or in the software handling sensor data. The stimuli to exploit these vulnerabilities could be local to the system (such as RF messages containing corrupted tire pressure monitoring data) or remote (such as remote diagnostics commands via the internet or a compromised workshop laptop PC). The most common method is a buffer overrun attack that causes the CPU stack to be corrupted by carefully crafted data that will cause the CPU to execute it as malicious code.

Exploitable programming errors in the firmware of devices are typically found by the discovery of the use of open source libraries with known errors or by reverse engineering the firmware of an ECU and looking for bugs. This process can be partially automated with the Ghidra tool developed by the NSA that takes a binary image to produce C source code. The source code can then be put through static analysis tools such as PCLint to automatically search for bugs.

1.3 Types of CAN attack

There are three broad types of attack when access to the CAN bus has been obtained:

- Authentication attacks. These are where a receiver sees CAN frames with manipulated data as if from a legitimate source but designed to trigger an action (e.g. open the door locks).
- Protocol attacks. This is where the signal on the CAN TX pin to the transceiver does not come from a CAN controller but software that sends carefully timed signals to attack the CAN protocol itself.
- Denial of service attacks. These can vary from simple flood attacks to load the bus with otherwise legitimate traffic (causing lower priority frames to be delayed or lost) to subverting the CAN protocol.

Authentication attacks are potentially the most severe: these can cause the movement of actuators in the real world. But denial-of-service attacks also have real world consequences too: blocking legitimate traffic can prevent vital functions from being carried out, which is particularly severe if essential vehicle controls are disabled.

1.4 Attack mitigation techniques

There are several techniques for mitigating attacks.

- **Intrusion detection**. This is a technique where the traffic on the bus is inspected for abnormal behaviours. Without hardware support it cannot generally prevent an attack but even so has a use in intelligence gathering and in post-incident forensics.
- **Security gateway**. This is a hardware approach using a device with two (or more) CAN bus interfaces. The gateway copies only legitimate traffic between the trusted bus (typically a vehicle control network) and an untrusted bus that contains a device that is potentially compromised.
- **Encryption**. This is generally a software technique (sometimes with hardware assistance) where an ECU protects its CAN bus traffic using cryptographic methods. Only receivers with a key can decrypt a message and verify its legitimacy. There are several issues around practical use of encryption for protecting CAN.
- **CAN security hardware**. These approaches use a hardware device included on a PCB that monitors the CAN signals to and from the CAN bus and provides various levels of protection.

Each of these mitigation techniques are discussed later in this document.

2 Attacks on CAN bus

2.1 Bus Flood Attack

A Bus Flood Attack is very simple denial-of-service attack: transmit CAN frames as fast as possible to soak up bus bandwidth, cause legitimate frames to be delayed and for parts of the system to fail when frames don't turn up on time.

The success of the attack depends on what mitigations there might be in place. For an open bus, transmitting a frame with a CAN ID of 0 will block all other traffic because this is the highest priority frame. If there is a gateway that only allows certain IDs to pass through then only the lower priority frames can be delayed: the higher priority frames will continue to be transmitted undisturbed (this is one reason why standard ODB-II diagnostic frames have IDs 0x7df and higher, giving them very low priority).

2.2 Simple frame spoofing

Frame spoofing is a type of authentication attack: getting a receiver to accept a fake frame as if it came from a legitimate sender.

- If directly connected (e.g. via the OBD-II port) this is done by simply queueing the CAN frame through the drivers in the firmware of the connected device.
- If connected via a hijacked ECU (e.g. infotainment) this can be done by using the drivers in the device or with new drivers installed as part of the hijacking.

One problem with this simple spoofing approach is that the frames from the legitimate ECU are also received and the receivers may act on both the legitimate frame and the spoofed one.

There is a bigger problem: the CAN protocol requires that two frames with the same ID are not entered into arbitration at the same time. If this does happen then after arbitration two controllers will be transmitting and when the bits transmitted differ then one of the controllers will see a bit error (transmitting a recessive bit but receiving a dominant bit) and signal an error frame, which causes all receivers to resynchronise an arbitration to start again. If no higher priority frame has been queued in the intervening time then these two frames will win arbitration again and the process will repeat. This is denoted the Arbitration Doom Loop: each go around the loop increases the Transmit Error Counter (TEC) in each of the two controllers and they will eventually both go into the bus-off state (i.e. be logically disconnected from the bus). This takes about 4ms on a 500kbit/sec CAN bus.

It can be a problem if the legitimate ECU is driven bus-off:

- The legitimate ECU may treat the failure as a wiring fault (since a short-circuit on CAN would appear the same) and permanently move into a fail-safe state where it refuses to communicate.

- All the other frames from the ECU will cease, which will cause receivers to detect a fault and they may also decide to move into a fail-safe state.

It may be that the intent of an attack is to deliberately push a vehicle into a fail-safe state of minimum functionality (where the vehicle is driveable but with very restricted operation). This would then be a form of denial-of-service attack.

2.3 Adaptive spoofing

The problems with simple spoofing are addressed by adaptive spoofing: the attacking device listens to the bus to see when the legitimate frame is sent and then queues the spoofed frame so that it does not clash. In many designs a receiver does not act immediately on a received frame but instead stores it in a buffer associated with the ID for a control loop to look at later. If the spoofed frame is sent immediately after the legitimate frame, then it will overwrite the buffer and the receiver will most likely act on the contents of the spoofed frame. This is illustrated in the figure below: there is only a small time-window when the true data is in the receiver's buffer.

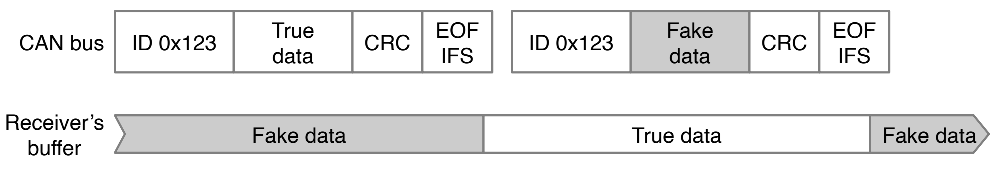


Figure 2: maximising the chances of a receiver seeing fake data

The attacker needs to respond to the frame received interrupt within a very tight timing window in order to queue a spoofed frame in time to enter arbitration: the frame receive interrupt will be raised at the second-to-last bit of the EOF field and arbitration will start within four bit times – a deadline of just 8µs at 500kbit/sec.

2.4 Error Passive Spoofing Attack

The simple spoofing approaches outlined above can be detected by monitoring the bus and looking at the timing of frames: spoofed frames will be sent more often than expected and traffic analysis can detect anomalies (of course, retrospectively detecting an attack may not be of much use when an actuator has already been commanded). Detection can be made much more difficult by a type of spoofing that subverts the CAN protocol itself: exploiting the behaviour of Error Passive mode (for more details of this attack, see Eland et al. 2017 [1]).

When a CAN controller is Error Passive (either the Transmit Error Counter or the Receive Error Counter is above 127) then the controller cannot signal errors properly: a transmitter has to basically stop sending and wait for the bus to become idle (it relies on other devices to see the stop in transmission and signal an error to resynchronise).

The attack is in two stages:

1. Driving the CAN controller of the targeted ECU into the error passive state. This is typically done by generating error frames when the targeted

ECU is sending any of its CAN frames (the TEC is increased by 8 each time it detects an error).

2. Monitoring the bus and seeing the ID of the targeted frame after it has won arbitration and then stepping in and overwriting the data and CRC fields with a spoofed payload.

The legitimate sender will detect an error with its own frame when the spoofed data field is sent (at some point it will send a recessive 1 bit but read back a dominant 0 bit). However, while error passive it cannot signal an error frame: it must send recessive bits and wait for arbitration to restart. This will leave the attacker sending the rest of the spoofed data field (and a new CRC field that matches the spoofed data). Receivers do not see this has happened: they merely see a single received frame with a spoofed payload. This also means that traffic analysis will not detect this attack.

This attack cannot be done through a CAN controller: it requires low-level access to the bus to spoof the CAN protocol itself. This means that the CAN TX and CAN RX pins on the transceiver must be under direct control of the attacker: an ODB-II dongle with an external CAN controller cannot be used:

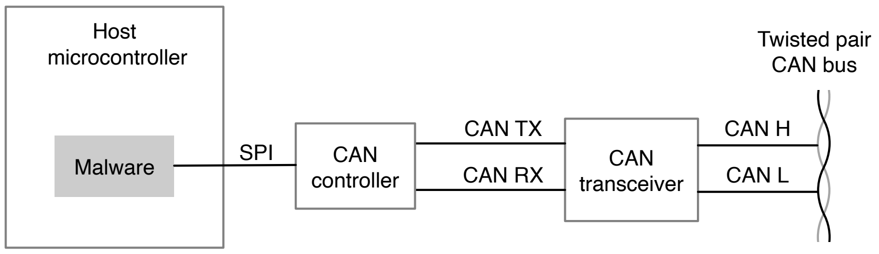


Figure 3: An external CAN controller with malware unable to directly access CAN TX and RX pins

A microcontroller with an on-chip CAN controller typically multiplexes the functions of its pins. The two pins used for the CAN controller signals to the transceiver can be controlled as general purpose I/O pins.

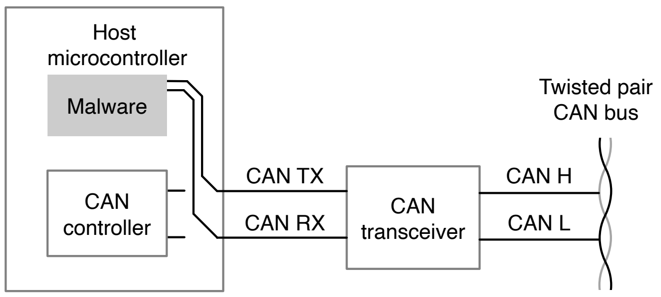


Figure 4: malware directly accessing CAN TX and RX pins by controlling the on-chip pin multiplexer

Malware can monitor the RX pin, detect the targeted frames as they start being transmitted and then drive the TX pin. In the first phase of the attack this is repeatedly driving the pin low for six CAN bit times to induce an error frame and drive the targeted ECU's CAN controller Error Passive. In the second phase the TX pin is driven with the spoofed frame data.

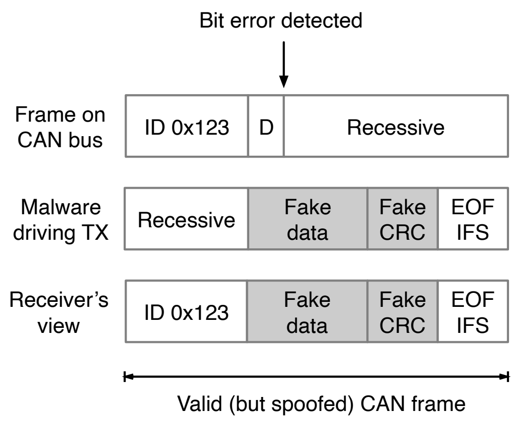


Figure 5: Hijacking a frame from a device in the error passive state

Emulating the CAN protocol in software is relatively straightforward (at 500kbit/sec each CAN bit is 2μs, plenty of time for a fast CPU): the software does not have to implement the entire CAN protocol, merely enough to achieve the spoofing.

2.5 Wire-cutting spoofing attack

If the attacker has physical access to the CAN bus and can cut wires to partition the bus then they can spoof frames to one of the partitions by emulating the other partition by gatewaying other frames and generating spoofed frames directly. This type of attack is used today by unscrupulous owners to spoof odometer readings so that although the ECU holding the odometer reading is outputting correct values, the dashboard display shows a reduced mileage and is inserted into a cut wiring harness.

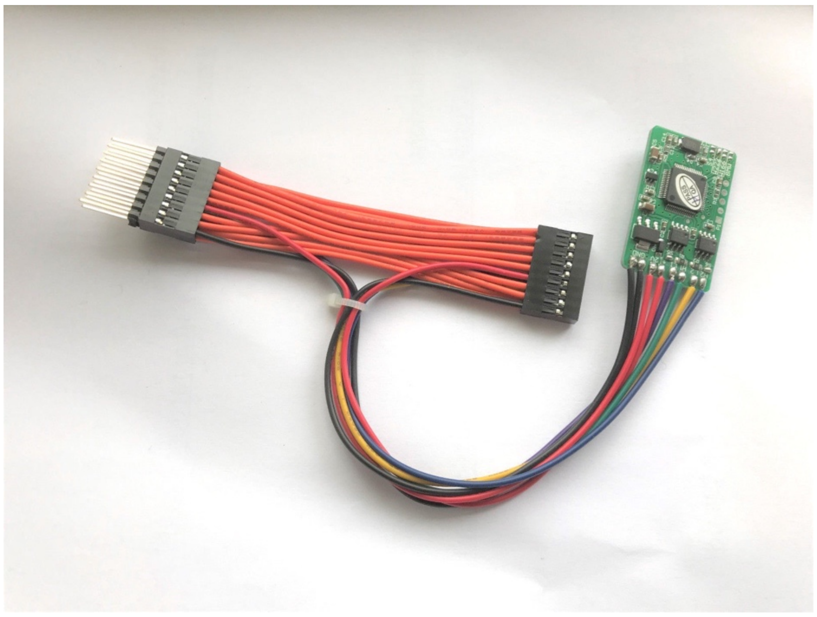


Figure 6: device for spoofing odometer CAN frames

2.6 Double Receive Attack

This attack is an exploitation of a feature of the CAN protocol that the ISO CAN specification includes a warning for [2]. The protocol defines that a receiver accepts a frame as finished at the second-to-last bit of the EOF field and that the transmitter accepts it as finished at the last bit of the EOF field. There is a very small chance of a bit error in the last bit of the EOF field: the transmitter sees a dominant bit, signals an error and re-enters the frame into arbitration. But all receivers will have already accepted the frame and passed it up to the application software. The transmitter will send the frame again and the receivers will receive the same frame again. This behaviour is a fundamental consequence of Buridan's Principle [3]. In most CAN systems the probability of failure is low: a low bit error rate multiplied by 111 (i.e. the chances of the erroneous bit falling at exactly the last bit of an 8-byte frame).

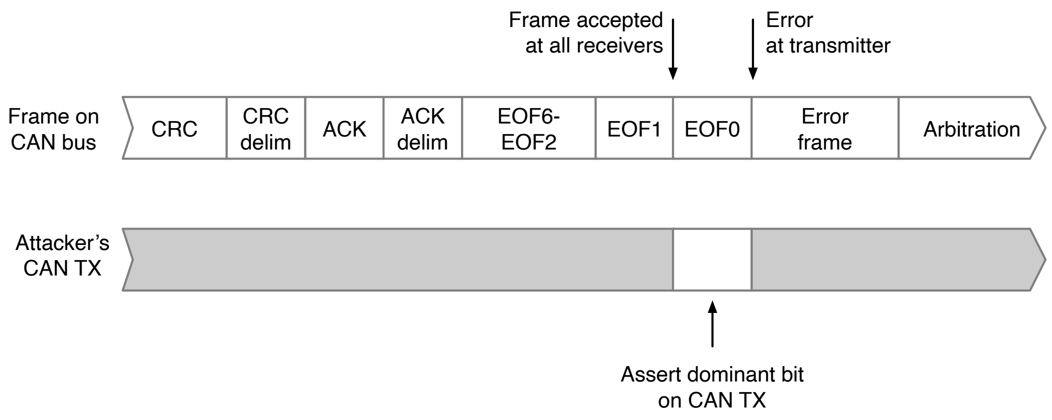


Figure 7: Asserting a dominant bit in EOF0 to trigger a double receive

The double reception of a frame can be detected by including a sequence number in frames. But most systems are not designed to do this (mostly because the probability of seeing the problem is low – error frames on CAN are anyway relatively rare).

A Double Receive Attack can be mounted if the attacker can control the CAN TX and CAN RX as general purpose I/O. A simple state machine in software implements part of the CAN protocol (there is no need for the software to implement the full CAN protocol). This is illustrated below:

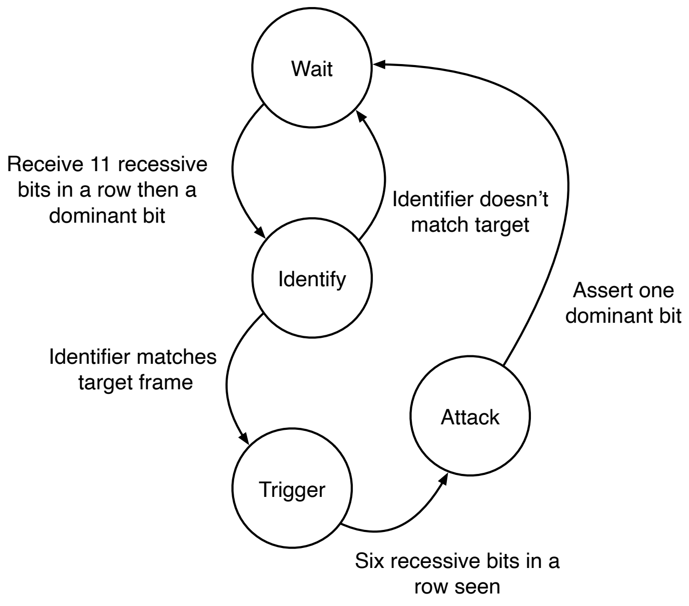

Wait
Receive 11 recessive bits in a row then a dominant bit
Identifier doesn't match target
Identify
Assert one dominant bit
Identifier matches target frame
Attack
Trigger
Six recessive bits in a row seen

Figure 80: A simple state machine for emulating CAN to attack a frame

In most microcontrollers the CAN bitstream can be decoded by polling the CAN RX pin and inspecting a free-running timer. The identifier is subject to bit stuffing but instead of running a de-stuffing algorithm it is easier to compare against a pre-computed pattern that is the ID with stuff bits. Once the frame has been identified it is a simple case to wait for six recessive bits and then assert a dominant bit on the CAN TX pin for one bit-time.

The simple approach outlined above is not particularly robust – an error occurring during the transmission of the targeted frame will not be detected, for example – but handling all the corner cases when mounting an attack is not usually an issue.

2.7 Bus-off Attack

The Bus-off Attack [4] is where a targeted ECU is driven offline: all the other ECUs continue to operate but the targeted one is removed. This might be part of a wider attack (such as a spoofing attack where the attacking device steps in and spoofs all the frames from the targeted ECU). Or it might be a simple denial-of-service attack on a fleet of vehicles: instead of trying to hijack the instrument cluster to display a Check Engine light it is probably easier to simply take the engine management ECU off the CAN bus and trigger the instrument cluster to see a failure and display a warning.

The Bus-off Attack is a low-level protocol attack driving the CAN TX pin as described earlier. But instead of targeting a specific frame, all frames from the same ECU are targeted. This forces the Transmit Error Counter above 255 and the ECU's CAN controller automatically goes bus-off.

Some ECUs will try to recover automatically, requiring the attack to be repeated. The network management and diagnostic strategy of the vehicle may eventually stop the ECU recovery process and instead set a flag in its internal non-volatile memory to stay offline. This will then typically trigger reduced driving functionality such as Limp Home mode. If the purpose of the attack is to cause trouble this would count as success.

2.8 Freeze Doom Loop Attack

The Freeze Doom Loop Attack[1] is a low-level attack that exploits a legacy feature of the CAN protocol. It effectively freezes bus traffic for an arbitrary time and could be used to delay a specific CAN frame to increase its latency or to generally remove bandwidth from the bus.

The attack could be used to trigger a specific reaction to a given frame arriving late or to delay the system responding to some condition. Or it could be used as a simple denial-of-service attack. The attack differs from others by being very difficult to detect: the error counters are not increased, and the only symptom is that frames arrive later than otherwise expected. If no timing analysis has been done to calculate the worst-case latencies of frames then the attack will resemble an inherent transient timing fault.

The CAN protocol defines a dominant bit in the first bit of the inter-frame space (IFS) as a controller signalling an overload condition. All CAN controllers go into the error recovery process but without incrementing the error counters. This is a legacy feature of CAN designed to allow slow CAN controllers to be given more time to handle a frame – no modern CAN controller generates this. The Freeze Doom Loop Attack works by asserting a dominant bit on the CAN TX pin at the first IFS bit then monitoring the error recovery and again asserting a dominant bit in the IFS field at the end of the error recovery. This can be repeated an arbitrary number of times, in effect freezing the bus for as long as desired.

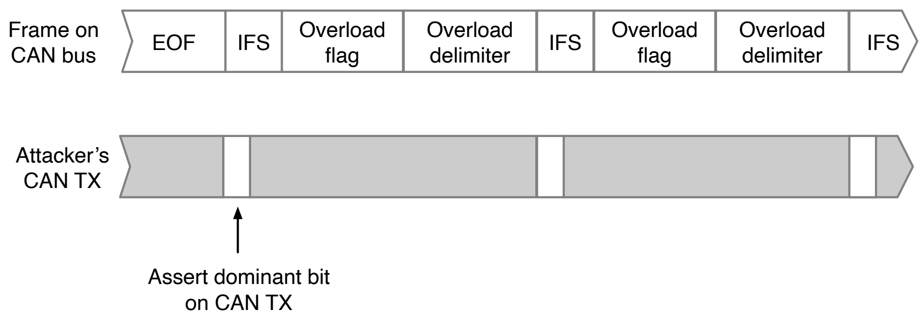


Figure 9: Freeze Doom Loop Attack

The complexity of this attack is the same as for the CAN protocol attacks described earlier: it involves a simple state machine with a timer to measure bit times, an optional comparison with a targeted ID, and a countdown of recessive bits to the IFS field.

[1] The Freeze Doom Loop Attack has not been published elsewhere. Canis Automotive Labs has not only produced a proof-of-concept but also uses the Freeze Doom Loop Attack as a mechanism for silently blocking an attacking CAN controller and as a flow control mechanism for CAN security gatewaying.

3 Intrusion detection systems

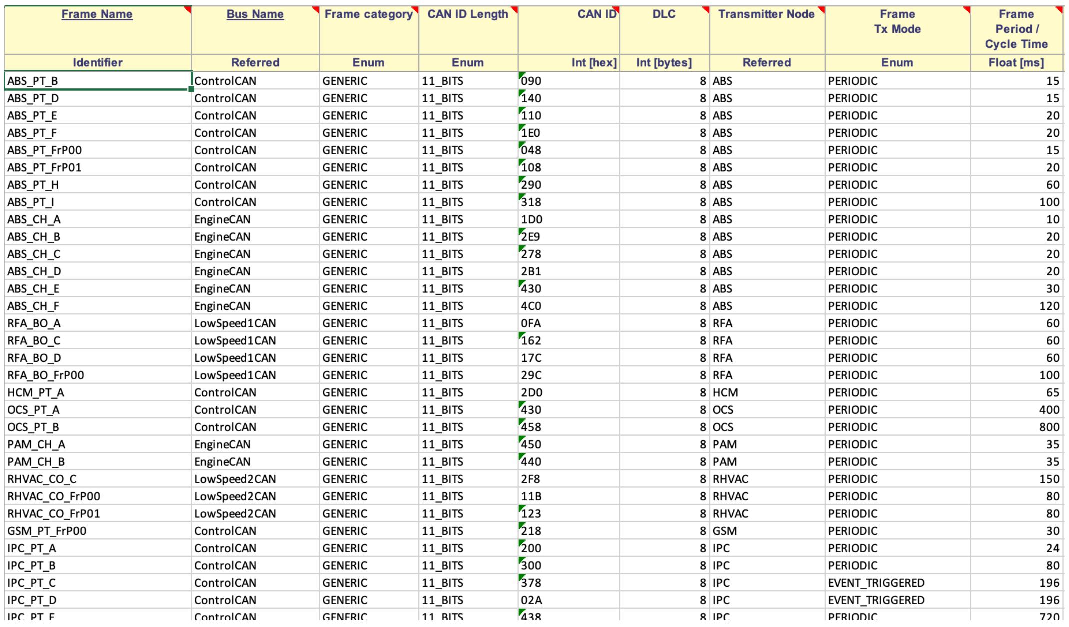
3.1 Introduction

The goal of an IDS is to detect when a likely attack is occurring and to take some action. If the IDS is purely a software application with a standard CAN controller then there is little direct action that can be taken to prevent or halt an attack: by the time the attack is detected it is generally too late to prevent a CAN frame being acted upon, and in any case an attack that includes a denial-of-service phase could prevent the IDS from communicating with targeted ECUs. However, a software-only IDS can collect data for post-incident analysis. This could be very important for preventing a repeat attack on other systems.

An IDS augmented by CAN security hardware does have the possibility of mitigating attacks before they can cause harm. This is discussed in more detail below.

3.2 Real-time traffic analysis

A CAN bus used for control is not like a mainstream IT network: it is part of an embedded system that has a very specific purpose where the communications patterns are known and fixed at design time. In most cases the CAN frames are derived from a database of signals into CAN frames transmitted periodically. Timing analysis can be used to calculate the worst-case latencies of the frames so a complete picture of the legitimate timing behaviour of the system can be known in advance.



Frame Name	Bus Name	Frame category	CAN ID Length	CAN ID	DLC	Transmitter Node	Frame Tx Mode	Frame Period / Cycle Time
Identifier	Referred	Enum	Enum	Int [hex]	Int [bytes]	Referred	Enum	Float [ms]
ABS_PT_B	ControlCAN	GENERIC	11_BITS	090	8	ABS	PERIODIC	15
ABS_PT_D	ControlCAN	GENERIC	11_BITS	140	8	ABS	PERIODIC	15
ABS_PT_E	ControlCAN	GENERIC	11_BITS	110	8	ABS	PERIODIC	20
ABS_PT_F	ControlCAN	GENERIC	11_BITS	1E0	8	ABS	PERIODIC	20
ABS_PT_FrP00	ControlCAN	GENERIC	11_BITS	048	8	ABS	PERIODIC	15
ABS_PT_FrP01	ControlCAN	GENERIC	11_BITS	108	8	ABS	PERIODIC	20
ABS_PT_H	ControlCAN	GENERIC	11_BITS	290	8	ABS	PERIODIC	60
ABS_PT_I	ControlCAN	GENERIC	11_BITS	318	8	ABS	PERIODIC	100
ABS_CH_A	EngineCAN	GENERIC	11_BITS	1D0	8	ABS	PERIODIC	10
ABS_CH_B	EngineCAN	GENERIC	11_BITS	2E9	8	ABS	PERIODIC	20
ABS_CH_C	EngineCAN	GENERIC	11_BITS	278	8	ABS	PERIODIC	20
ABS_CH_D	EngineCAN	GENERIC	11_BITS	2B1	8	ABS	PERIODIC	20
ABS_CH_E	EngineCAN	GENERIC	11_BITS	430	8	ABS	PERIODIC	30
ABS_CH_F	EngineCAN	GENERIC	11_BITS	4C0	8	ABS	PERIODIC	120
RFA_BO_A	LowSpeed1CAN	GENERIC	11_BITS	0FA	8	RFA	PERIODIC	60
RFA_BO_C	LowSpeed1CAN	GENERIC	11_BITS	162	8	RFA	PERIODIC	60
RFA_BO_D	LowSpeed1CAN	GENERIC	11_BITS	17C	8	RFA	PERIODIC	60
RFA_BO_FrP00	LowSpeed1CAN	GENERIC	11_BITS	29C	8	RFA	PERIODIC	100
HCM_PT_A	ControlCAN	GENERIC	11_BITS	2D0	8	HCM	PERIODIC	65
OCS_PT_A	ControlCAN	GENERIC	11_BITS	430	8	OCS	PERIODIC	400
OCS_PT_B	ControlCAN	GENERIC	11_BITS	458	8	OCS	PERIODIC	800
PAM_CH_A	EngineCAN	GENERIC	11_BITS	450	8	PAM	PERIODIC	35
PAM_CH_B	EngineCAN	GENERIC	11_BITS	440	8	PAM	PERIODIC	35
RHVAC_CO_C	LowSpeed2CAN	GENERIC	11_BITS	2F8	8	RHVAC	PERIODIC	150
RHVAC_CO_FrP00	LowSpeed2CAN	GENERIC	11_BITS	11B	8	RHVAC	PERIODIC	80
RHVAC_CO_FrP01	LowSpeed2CAN	GENERIC	11_BITS	123	8	RHVAC	PERIODIC	80
GSM_PT_FrP00	ControlCAN	GENERIC	11_BITS	218	8	GSM	PERIODIC	30
IPC_PT_A	ControlCAN	GENERIC	11_BITS	200	8	IPC	PERIODIC	24
IPC_PT_B	ControlCAN	GENERIC	11_BITS	300	8	IPC	PERIODIC	80
IPC_PT_C	ControlCAN	GENERIC	11_BITS	378	8	IPC	EVENT_TRIGGERED	196
IPC_PT_D	ControlCAN	GENERIC	11_BITS	02A	8	IPC	EVENT_TRIGGERED	196
IPC_PT_E	ControlCAN	GENERIC	11_BITS	438	8	IPC	PERIODIC	720

Figure 10: Excerpt of a CAN frame database with timing information

In systems that are not designed with a systematic process it is still possible to observe a system and make a guess about the expected behaviour. However, this is not ideal because the initially observed behaviour will likely not be at edges of the actual performance envelope and there is a risk of false positives (i.e. when an alarm is triggered but there is nothing wrong). Sporadic frames (i.e. those transmitted by an ECU only when an event occurs) are a particular problem because they may never be seen the observation phase.

The monitoring process involves timestamping the arrival of every CAN frame, identifying the frame from its CAN ID and continuously comparing it to the frame's timing envelope. There will typically be variance in the arrival times for a periodic frame (called 'arrival jitter') due to other traffic on the bus winning arbitration first. An idle period before the SOF bit means that the frame must have been queued at the SOF, narrowing the timing window for when the frame was generated.

The IDS must in general observe several transmissions of a frame before it can be sure there is a timing violation. For example, a frame with a period of 100ms and a worst-case latency of 90ms could be queued at time 0ms and time 100ms but be received at time 90ms and time 101ms. IDS real-time monitoring can detect likely simple spoofing attacks by seeing a frame too often (although it cannot detect which is the genuine frame and which the spoof). A Flood Attack can similarly be detected.

The Double Receive Attack will appear as if the sender is exceeding the defined real-time behaviour by transmitting a frame twice in rapid succession and so may be misidentified. This is a typical problem with any IDS: it can flag a suspicious event and log information around that event, but it can rarely be certain that the event is an attack and even less certain about the source of the attack. It cannot act without risking the consequences of a false positive.

Some CAN controllers can provide information on errors and IDS software can use this to detect an unexpectedly high rate of errors (either by seeing the Receive Error Counter increase rapidly or by timestamping individual error events) and flag a potential Bus-off Attack (see section 2.7) or an Error Passive Spoofing Attack (see 2.4) because these rely on forcing an individual ECU into an error mode.

If the CAN controller used by IDS software has the capability to report overload conditions then a Freeze Doom Loop Attack (see section 2.8) could be flagged. As with the Double Receive attack, there is a possibility that this might occur rarely from random errors, but repeated occurrences are very unlikely to be random.

3.3 Payload analysis

One useful feature of an IDS is to look inside the payload of a CAN frame to look for suspicious behaviour. This is particularly useful for frames that are part of a diagnostic session: there is rudimentary security included in the Unified Diagnostic Services (UDS) protocol – in some cases a PIN number is used – but it is often straightforward to brute force this. But an IDS could detect this brute forcing and possibly even act before a system is compromised.

A centralised IDS can also use knowledge of the context of a system when examining payloads. For example, if the connection of a diagnostic test tool is independently verified (perhaps by a signal related to a physical connector) then seeing diagnostic frames without this signal would be a strong indicator of a diagnostic session being spoofed to attack an ECU (the diagnostic software stack in an ECU is complex and a likely place for programming errors that could become vulnerabilities).

A simple IDS could also be run in each individual ECU: it is set to receive all frames and if the ID of any of them matches the CAN ID of a frame normally transmitted by the ECU then this can be flagged as a spoofing attack. There will be performance issues with this: on a 500kbit/sec bus then a frame could potentially be received every 94µs, which would amount to quite a large CPU load dedicated to monitoring the bus. Dedicated hardware for this would be more efficient.

3.4 Hardware support

A software-only IDS cannot generally prevent attacks: by the time a spoofed frame is detected it has been received by ECUs and potentially acted upon. But hardware designed to support an IDS does allow mitigations.

A CAN controller with support for generating interrupts before the frame has been received allows the IDS to decide if the frame is spoofed and to inject an error frame to prevent receivers seeing it. Canis Automotive Labs has developed a device – Mercury™ IDS – to do this. It contains a CAN state machine and can be configured to raise interrupts for the following events:

- Start-of-Frame (SOF)
- End of arbitration
- CRC check passing
- Frame received OK
- Frame transmitted OK
- Error or overload frame

Mercury IDS can be instructed to generate six dominant bits to trigger an error frame and so destroy a spoofed frame. It also reports on which errors were seen and provides a timestamp for SOF. This allows the IDS software to determine what exactly happened on the bus. For example, it can distinguish between a Flood Attack and the Double Receive Attack (the Flood Attack is the frame queued back-to-back, the Double Receive Attack requires an error at precisely the end of the first frame transmission).

IDS hardware can also be used to provide metadata to the IDS software. One example of this is transceiver fingerprinting: if the voltages of CAN H and CAN L are measured accurately at a high rate it is possible to determine the type of transceiver used and potentially even where on the cable the transceiver is located. A statistical correlation is then used to determine the probability the frame came

™ "Mercury" is a trademark of Canis Automotive Labs Ltd.

from a specific device. The false positive rate is too high use this approach to suppress an attack, but it can be useful in post-incident forensics.

Another way to obtain metadata is for security hardware to inject it at the transmitting side. Canis Automotive Labs has developed a bus guardian device that injects a pre-programmed 7-bit source address into all outgoing CAN frames (see section 6.3). Mercury IDS provides this address to the IDS software. Detecting spoofing attacks from devices with the source address is then straightforward: the software includes a table of CAN IDs and their source addresses and a spoofed frame can be immediately identified and destroyed before any device received the frame. The Canis Automotive Labs bus guardian device also accepts commands over CAN from the IDS software to disconnect its host from the bus – this allows the source of the spoofed message to be shut down (this is discussed in more detail in section 6.3).

4 CAN security gateways

4.1 Overview

A CAN security gateway copies legitimate traffic back and forth between an untrusted side and a trusted side. The trusted side is a CAN control bus (or more typically, buses where control functions run across several buses with gatewaying).

A security gateway can protect the trusted bus in the following ways:

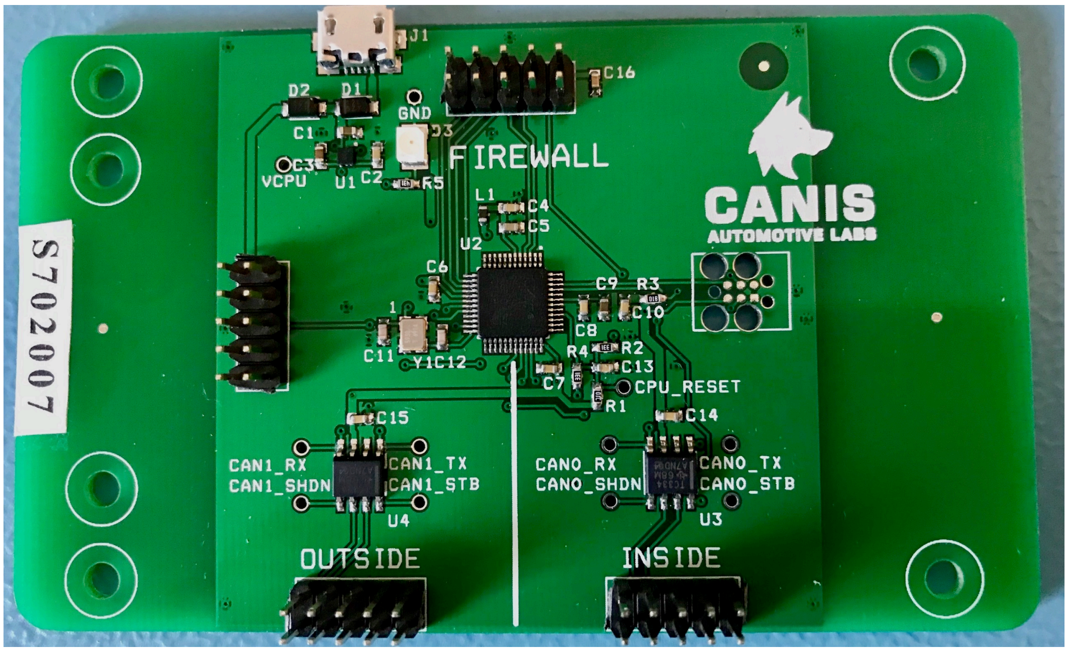
- Protects from low-level protocol attacks. Because the access to the trusted bus is only via CAN controller hardware in the gateway there is no opportunity to take direct control of the CAN TX pin and attack the CAN protocol on the trusted bus.
- Protects from denial-of-service attacks. Attempts to flood the bus will fail because the gateway can refuse to forward traffic outside of a pre-defined real-time envelope.
- Protects from spoofing attacks. Only pre-defined frames are permitted through the gateway (although if there is more than one device on the untrusted side then one of these devices can spoof frames from another).

There may be several security gateways, depending on the cable harness. Typically, both the ODB-II and the infotainment units will each be behind gateways (these are the source of most of the present-day attacks on CAN).

A security gateway may have many features, depending on the level of sophistication. Some of these are discussed below with reference to the Network Security Processor (NSP) solution developed by Canis Automotive Labs.

4.2 Firewalling functions

The Canis NSP uses an Arm-based microcontroller running custom firmware (developed entirely in-house with no third-party software – not even C libraries – so that there is complete control over the device). An example PCB with the device is shown below:



The NSP has two CAN interfaces: one on the 'outside' (the untrusted network) and on the 'inside' (the trusted network). The PCB includes a CAN transceiver for each bus, so the inputs to the hardware are the raw CAN H and CAN L lines.

The device contains flash memory and stores the CAN drop rules:

- CAN ID drop rules use mask/match values to decide to forward from one bus to another. In a simple setup this would include OBD-II diagnostic requests from the ODB-II side and responses from the control bus and nothing else. An infotainment system would have more traffic (which might include the status of 'soft' buttons on a touchscreen such as the tap of 'a deactivate airbag' button).
- Real-time drop rules prevent flood attacks. A real-time drop rule assigns an arrival pattern to a given frame (frame period and variability, frame burst size).
- Payload drop rules in the Canis NSP allow signals within a CAN frame payload to be defined, and the valid ranges expected for the signals.
- Frame rewrite rules allow signals not needed by an untrusted device to be masked out so that sensitive information is not leaked to an untrusted device.

The Canis NSP allows mode changes so that the rules can be switched in and out depending on what the system is doing. This is useful when the traffic patterns vary a lot in different modes (such as diagnostic frames only during a diagnostic mode).

Note that a security gateway cannot in general prevent the hijacking of ECUs by corrupted versions of legitimate messages. For example, imagine a wireless interface device, with WiFi and internet connectivity, used to forward diagnostic commands (from a workshop laptop, a diagnostic server, etc.). A security gateway would pass diagnostic commands through because those messages permitted.

The payload of a spoofed diagnostic messages may then trip a buffer overflow bug (for example) in a targeted ECU.

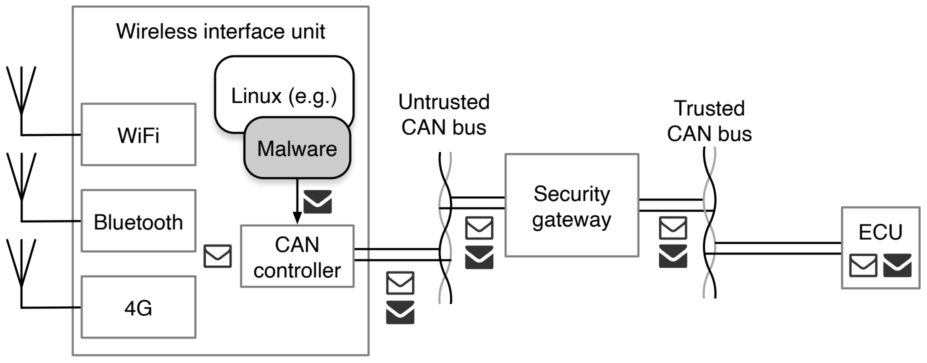


Figure 11: How a hacked device on the untrusted bus can send legitimate messages with malware payload through a security gateway and hijack an ECU on the trusted bus

The basic functionality of a security gateway is relatively straightforward but there are implementation issues that must be addressed. These are discussed below.

4.3 Secure control of gateways

The gateway must be in general controlled from elsewhere to adapt to changes. Examples include:

- To re-program the firmware with updates.
- To update drop rules to accommodate changes in the system design.
- To notify the gateway of mode changes.
- To extract diagnostic information from the gateway.

In many cases these control messages must come from the untrusted side (e.g. when new firmware is distributed) and so there must be a secure communications channel to the gateway from anywhere.

The Canis NSP addresses this by providing an encrypted end-to-end messaging system for control commands, including a bootloader that can reprogram the firmware of the gateway. There are further issues with how to do secure encrypted messaging (key distribution, cryptographic protocols, etc.) – these are discussed later.

4.4 Critical messages via the untrusted bus

The gateway may be required to transmit critical messages received from the untrusted bus across to the trusted bus. For example, frames containing soft button presses on an infotainment touch screen, or frames containing new firmware for over-the-air (OTA) download functions. A security gateway should have special support for ensuring these frames can only reach the control bus if genuine.

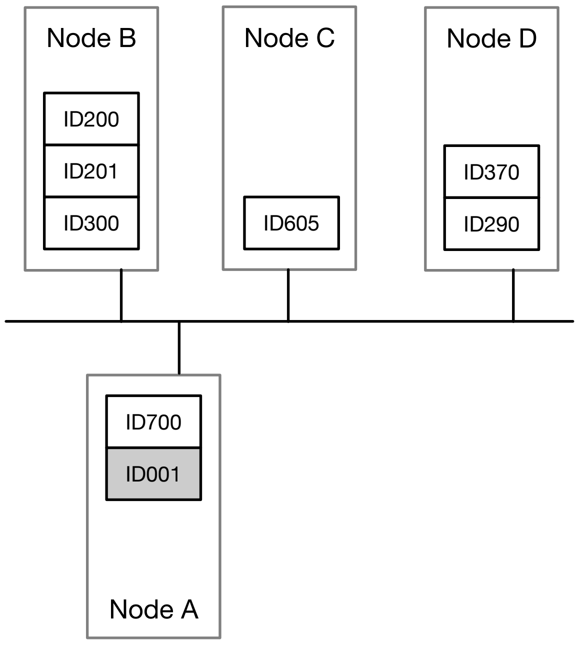
The Canis Automotive Labs NSP has support for CAN frames on the untrusted bus that are protected by end-to-end encryption. This is to support CAN frames generated by a remote server and containing OTA firmware and configuration data for ECUs. The NSP decrypts them and passes them to the trusted control bus if they are authentic.

CAN frames that represent critical messages emerging from an untrusted source can also be protected. The NSP allows some rules to be applied only with a hardware interlock input in a specific state. Critical messages can then be designated as forwarded only if that input is selected. For example, the NSP could be programmed to only forward airbag deactivation command frames from an infotainment system if a human were simultaneously pressing a physical button. Similarly, OTA firmware updates might only be forwarded if a key were turned in a lock. This would prevent purely automated attacks on a vehicle.

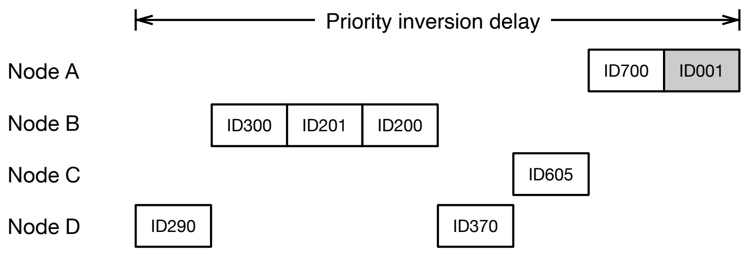
4.5 Frame queuing

The CAN protocol defines an arbitration system that selects the highest priority frame (i.e. lowest ID) frame on the bus to be transmitted. This approach should be extended into the frames within each device connected to the bus: if there are several frames ready to be sent on the bus then when arbitration starts, the highest priority of those should be the one entered into bus-wide arbitration. This is important for ensuring the real-time performance of the CAN bus. There are several reasons for this but the most important one is to avoid *priority inversion*.

Priority inversion is where a high priority frame can be held up by many lower priority frames. It typically occurs when a CAN driver implements a FIFO buffering scheme as illustrated below:



The above is an example of a system of four nodes, with FIFO queueing of CAN frames. At arbitration start, the front of each FIFO is entered into CAN arbitration. The first frame transmitted has an ID of 290. The diagram below shows the timeline of frame transmission. The highest priority frame in the system is delayed by a long sequence of lower priority frames.



The delays from priority inversion can be severe but also highly intermittent. They are also likely to cause a false positive in an NSP performing real-time traffic analysis leading. The security gateway must queue outgoing CAN frames in priority order (most CAN controller hardware is designed to support).

There is a further problem with CAN frame queuing: frame ordering. If a second instance of the same frame is received on one bus before it has been transmitted on the other bus then this second frame must not be put into the outgoing CAN controller's priority queue until after the first instance has been transmitted. To do otherwise would risk it being transmitted first: most CAN hardware will make an arbitrary choice of which frame to send when two frames have the same ID (in any case, the same logical frame may have different specific IDs: the SAE J1939 standard can use sub-fields in a 29-bit CAN identifier to indicate destination address).

Maintaining the order of frames is very important: some frames form part of a large multi-frame message (for example, SAE J1939 supports messages up to 1785 bytes) and swapping the order of frames could corrupt the contents. This means that there should be a FIFO input queue for incoming frames of a given ID. The Canis Automotive Labs NSP permits per-frame FIFO queues of configurable size to ensure the ordering of frames. The configuration also allows a logical frame FIFO to be assigned multiple CAN IDs (to allow for sub-fields within the ID to have arbitrary values and still be considered the same frame).

A general problem with CAN security gateways is buffer space: the transmitter on one side sees the frame as transmitted before it is transmitted on the other side. For a system with a single device on the untrusted side this is a problem: the sender may be sending a multi-frame message and as soon as it sees its frame as transmitted it may immediately queue the next. The untrusted bus will typically have little traffic and so there is the possibility of flooding this bus and overflowing the input buffers at the gateway. In mainstream computer networking this is addressed by flow control: signalling the source to temporarily stop transmitting. CAN has no flow control. One solution to this is to use the Freeze Doom Loop Attack as a flow control mechanism for the untrusted bus, freezing the CAN bus until the gateway buffers are no longer full.

4.6 Software correctness

The security gateway is a vital component in the securing a system so it must be free of software vulnerabilities. Great care when developing the software should be taken, using development techniques for high integrity software. These include:

- Formal requirements capture and trace.
- Ensuring high test coverage results.
- No use of untrusted third-party software.
- Use of a coding standard designed to minimize common security errors.
- Static source code analysis tools.
- Automated regression test suites.

Many of these techniques are shared with developing safety critical software.

5 Encryption techniques

5.1 Overview

Encryption provides two protections for messages: *secrecy* and *authentication*.

Encrypting CAN frame payloads for secrecy makes it harder to determine the signalling patterns (a precondition for spoofing CAN frames) but does not prevent a well-resource adversary from reverse-engineering the communications (they can usually break into an ECU microcontroller, extract the firmware and examine it to see how CAN frame payloads are handled). In any case, obfuscating CAN IDs is not feasible.

Authenticating CAN frames is much more important: spoofing attacks rely on the receiver not knowing if the message is authentic (i.e. from a legitimate source). The cryptographic authentication process is generally as follows:

- The sender computes a message authentication code (MAC) from the details of the message, using a MAC algorithm and a secret key.
- The MAC is attached to the message and both are sent together.
- The receiver performs the same computation on the message with the same secret key. If the result matches the received MAC then message must have come from a sender who knows the secret key.

Encryption is limited in what it can achieve: it can only mitigate message content and sequence attacks. While this does prevent re-wiring spoofing attacks, it cannot address denial-of-service attacks. It does not detect or prevent the Flood Attack, Freeze Doom Loop Attack and Bus-Off Attack.

There are several issues for implementing encryption on CAN:

- Bandwidth. The bus load is increased due to including a MAC with every message.
- Performance. How long it takes to generate and authenticate messages and how long it takes to start up and begin communications are critical in a real-time control network.
- Key distribution. Both ends of communication must share the same key. These keys need to be generated securely, put into devices, kept secret and potentially replaced later with new ones.
- Resisting attacks. There are specific attacks on poorly-implement encryption systems.

Although encryption is conceptually simple there are many details to get right and a single mistake can render the system useless to a skilled attacker (there are many examples of this happening before, from WiFi encryption in routers to the firmware authentication system in the XBox).

The issues above are discussed below with reference to the CryptoCAN encryption solution developed by Canis Automotive Labs.

5.2 Encrypting CAN payloads

AES is the most commonly used cipher and a lot of microcontrollers include hardware accelerators to run the algorithm. AES is a block cipher that encrypts in chunks of 16 bytes. The CryptoCAN scheme developed by Canis Automotive Labs uses AES and the 16 bytes are handled as follows:

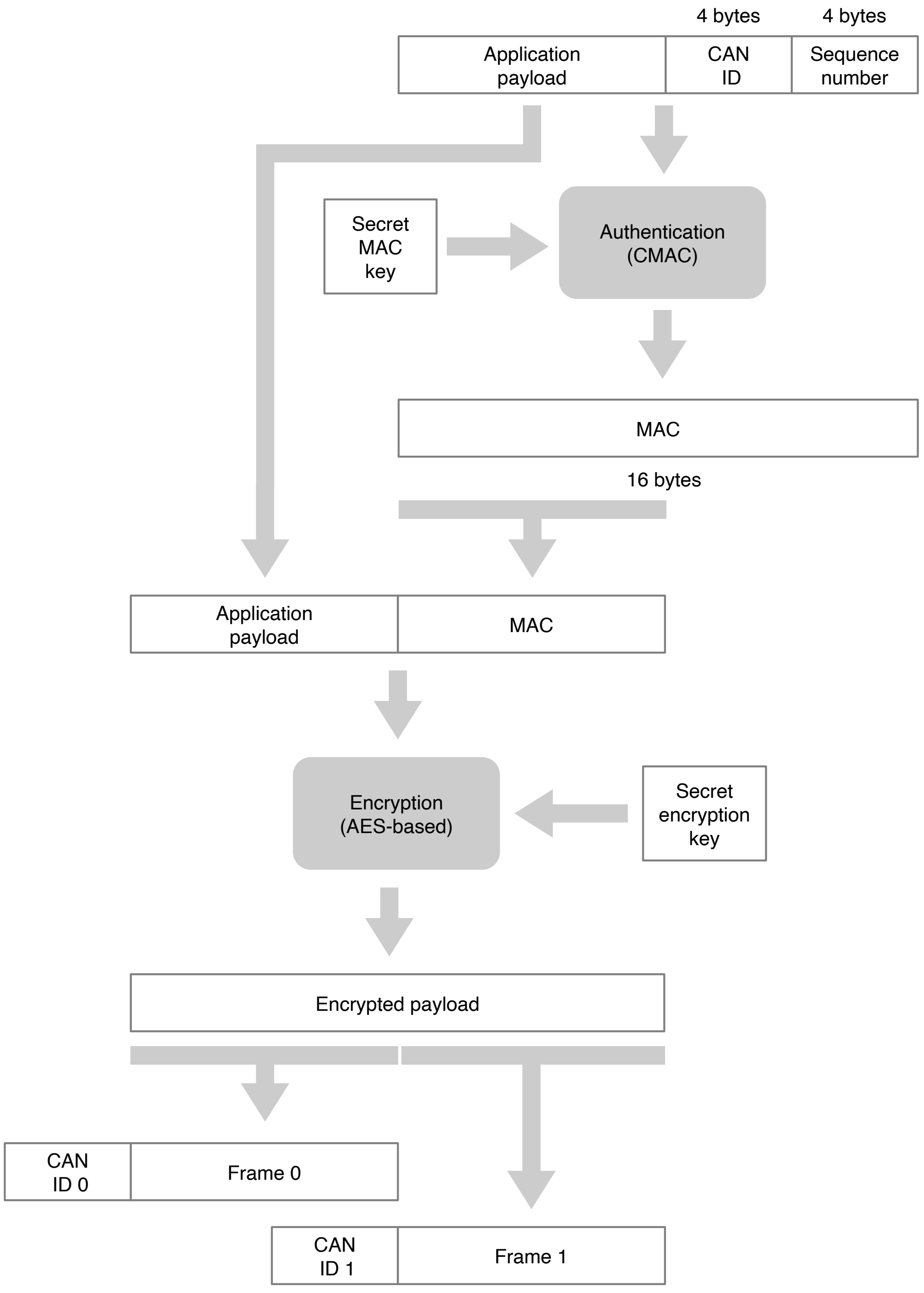


Figure 12: The basic CryptoCAN scheme

The application CAN payload, its CAN ID, a sequence number and a secret key are input to the authentication algorithm. This generates a 128-bit authentication code, but only the top 64 bits are used for the MAC (64 bits are considered the minimum to be secure – there are widely-known attacks on shorter authentication codes).

The 64 bits of the CAN payload plus the 64-bit MAC are put together to form a 128-bit block that AES encrypts. The resulting 128-bit encrypted block is split into two pieces, which are transmitted in two CAN frames that are then queued as a pair.

The CAN ID is not the same for both halves because there would be a problem with CAN drivers re-ordering the frames (as described earlier) so they differ by one bit (typically the least-significant bit): the first frame has a 0 for this bit and the second frame has a 1.

At the receiver these two frames are put together and the process reversed, as shown in the diagram below:

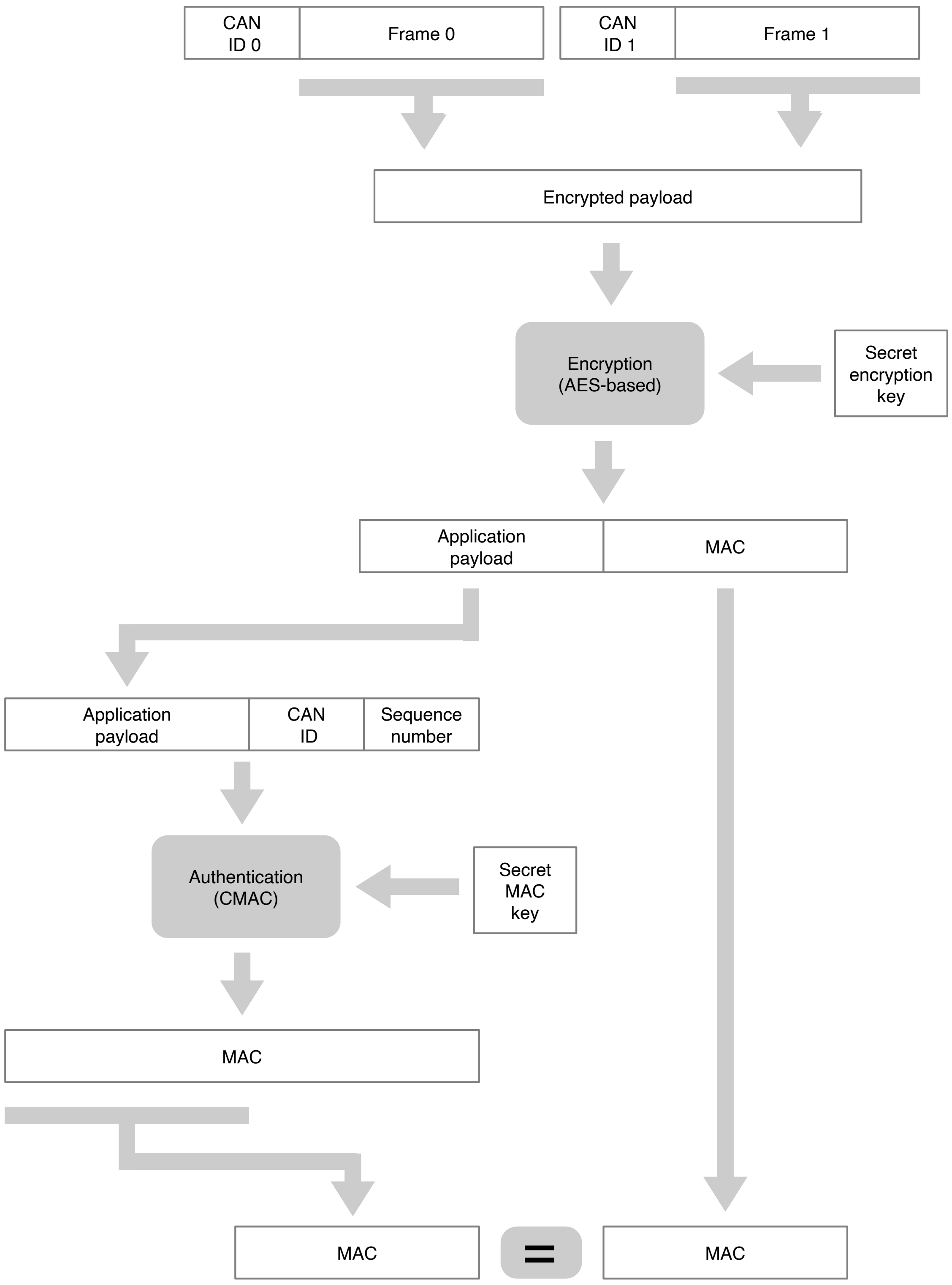


Figure 13: Authenticating a CryptoCAN message

The computed MAC and the received MAC are compared with each other and if there is a match then the application payload is assumed to be authentic and passed to the application.

The encryption steps can be skipped if there is no need for secrecy. In any case, the description here is simplified: CryptoCAN uses a cipher feedback mode with a cryptographically secure pseudo-random number generator (CSPRNG) to scramble the frames so that if two frames have the same application payload then they do not get the same encrypted values.

5.3 Replay attacks

One trivial attack on message authentication is to take the MAC and payload from a valid message seen on the bus and re-use it in another message: the payload would pass the authentication checks and the imposter message acted upon. To prevent this the MAC must be computed on the whole message, not just its payload (this is why CryptoCAN includes the CAN ID in the MAC).

A related attack is to copy an authentic message and then replay it later when the attacker wishes to get receivers to act upon it. This can be a very effective attack: the attacker cannot forge a message, but they can copy one that they know contains the desired contents. They can then send this message whenever they want to cause the same effect.

Replay attacks are defeated by including a 'freshness' value in the MAC calculation so that the calculated MAC on a stale message will not match the MAC in the message. A simple way to obtain freshness is to use sequence numbers in a message: receivers discard messages with old sequence numbers. CryptoCAN does not do this because the bandwidth costs of including sequence number in a message would be too high (it needs to be big enough to allow old messages to become valid again – a 32-bit value on a message sent every 10ms would wrap after about 500 days of continuous operation). Instead it uses a timestamp and runs its own clock to keep track of the time. The timestamp changes quickly enough that the window for replaying messages is too short to be useful.

The distribution of the timestamp is a weakness that can be attacked: the sender and receivers must agree on the timestamp so there needs to be a way reach consensus on the timestamp. This can either be via a single master issuing a timestamp or a distributed algorithm with no single master. In either case those messages need to be authenticated without the timestamp messages being subject to a replay attack: if the attacker can set the timestamp back in time then old messages from a previous session can be replayed as if new. Authenticating the timestamp distribution messages then requires an exchange of messages in a multi-step challenge-response protocol run between the timestamp distributor and each receiver.

CryptoCAN includes an API for setting the sequence number for the MAC calculation so that any of the above strategies can be used.

5.4 Side-channel attacks

A side-channel attack is where information about the keys can be leaked by small variations in an observable behaviour. One of these uses power consumption – called differential power analysis (DPA) – where the switching of transistors in a microcontroller causes tiny changes in the power supply voltage. Observing enough operations allows a statistical model to be populated and to assemble a copy of the key within a matter of a seconds. The risk of this attack is relatively low for ECUs because it requires access to the power supply pin on the MCU with specialist equipment.

Another side-channel attack is a timing attack: small variations in the time taken to encrypt or compute a MAC can leak the keys in much the same way as for DPA. For CAN frames that emerge on to the CAN bus after being encrypted there will be tiny variations in when the CAN frame is transmitted (the process of arbitration adds noise, but this merely requires more observations before the model is fully populated).

To defeat a timing side-channel attack there must be a fixed time taken for the basic cryptographic operations to take place and a frame to be queued. Hardware accelerators for AES are implemented to do this but software is often not: common implementations of AES use a lookup table that is stored in flash memory. But flash memory uses caching, and the pattern of cache misses leaks information about how the lookup table is being accessed. Any encryption implementation with variations in timing must therefore not queue encrypted CAN frames after they have been computed but instead from an independent timer countdown event.

5.5 Performance

The CryptoCAN system doubles the required bandwidth: each 8 bytes of payload is authenticated with an 8-byte MAC. This is one of the fundamental drawbacks to encryption and one of the motivations in the adoption of the CAN FD protocol.

Implementing encryption in software is difficult (see the above discussion on timing side-channel attacks) and the resulting software takes a lot of CPU time. The CryptoCAN implementation of AES on the Cortex M3 is hand-optimised assembly language and takes a fixed 1049 clock cycles – or 10.9µs at 96MHz – to encrypt a 16-byte block (the Cortex M0+ implementation takes 2064 cycles or 43µs at 48MHz). There may be several invocations of the AES operation per encrypted message (at least one for the MAC and another for the payload encryption). This adds up to a large amount of CPU time: it is a significant fraction of the time taken to transmit a CAN frame.

Hardware acceleration is one way around this problem, but it will not always be possible to use this everywhere. In any case there are sometimes architectural limitations to using it for communications. For example, the S32K microcontroller family from NXP has the AES accelerator inside the flash controller module (it stores encryption keys securely in a private area of flash memory) and the encryption accelerator cannot be used during writes to flash memory. Flash memory writes can take a long time, and this would block the sending of encrypted CAN messages.

5.6 Key management issues

Distribution of keys is a major issue with any embedded encryption system. At manufacture each ECU needs to get a set of keys that are unique for a systems: if keys are re-used between vehicles then reverse-engineering one vehicle means the encryption is broken on all vehicles.

The keys can be programmed securely in the factory, but they also need to be programmed in the field: a new replacement ECU part must get the keys for the specific system. This process of getting new keys must itself be authenticated (otherwise an attacker can just change the keys to known values).

CryptoCAN defines a protocol for a device communicating with a trusted tool. This tool can set the keys for an ECU and has extensions for other functions (it is used in the Canis Automotive Labs NSP for programming the gateway rules and for updating the firmware of the NSP as well as setting keys). A challenge-response protocol is used so that the tool and the target authenticate each other before keys can be set. The tool protocol follows the model of the HIS Secure Hardware Extension (SHE): there is a master key set unique to the ECU that is just used for encrypting and authenticating the communications with the tool.

The CryptoCAN communication between the tool and the ECU is secured end-to-end: the tool will typically be an application running on a server in a secure data centre at the OEM, with access to a secure database storing the master key for the specific ECU (the CryptoCAN protocol can assign a unique serial number to an ECU). CryptoCAN also includes a three-phase transaction commit for writing keys so that they are written atomically: if there is a power failure or other interruption while the master key is set then a key will either be the old value or the new value (otherwise a corrupted partially-written key would result in the device being unreachable – bricked, in other words).

The HIS SHE not only defines the protocol for storing keys but also specifies that the key storage is one-way: the host MCU cannot read the keys out. This is to protect against ECU hijacking: if the ECU is compromised by malware then at least that malware cannot extract the keys and pass them to another device. However, sharing authentication keys is a vulnerability of any symmetric encryption scheme.

Any device with the shared key can forge messages as if from any other device using that key. A hijacked ECU can create authenticated spoof messages.

For point-to-point communication, sharing keys at either end is not a problem. But for a group of ECUs on a CAN bus it undermines the entire encryption-based authentication. For example, if a single key is shared across all ECUs in a vehicle then then hijacking one of ECUs allows the hijacker to create a valid MAC and spoof an encrypted CAN message from any device to the rest of the group.

The shared secret problem could be mitigated by adopting asymmetric encryption (all the encryption discussed up until now has been symmetric: both sides have the same key). An asymmetric system has a key is split into two halves, a private half and a public half. The private half is held only in a single device, and the public half is put into every other device. The sender computes a digital signature of the message using the private key. The signature can be verified using the public key alone: it does not require the sharing of secrets between devices. Hijacking an ECU will not allow it to spoof message signatures because the private key is held only at the legitimate sender.

Unfortunately, the asymmetric algorithms (such as ECDSA) are very slow: instead of 1049 clock cycles on a Cortex M3 for an AES encryption operation, an ECDSA signature verification takes 12.2 million clock cycles (or 127ms at 96Mhz). In mainstream computing, asymmetric algorithms are used once to exchange temporary session keys for symmetric algorithms that then communicate more quickly. But this approach does not help with embedded CAN security:

- A long delay before starting communications on CAN is unacceptable. If an ECU is reset for some reason (e.g. a watchdog timer is triggered) it must pick up and continue as quickly as possible. When controlling a moving mechanical device like an engine there is no time to request a re-run of the key distribution phase and wait for all the other ECUs to settle on a new key.
- A session key shared across a group still allows a hijacked ECU to spoof messages from other ECUs in the group.

Until high-performance hardware for a standard digital signature implementation is ubiquitous this approach is infeasible.

6 CAN security in hardware

6.1 Introduction

The TX input into a CAN transceiver is the source of all the attacks on CAN bus. The ultimate approach to protecting CAN bus is to protect this. As discussed earlier, a simple way to achieve this is to use standalone CAN controller. But this will not protect against spoofing and flood attacks. Specific CAN security hardware can be used for more protection.

6.2 Anti-spoofing CAN transceiver

Spoofing is sending a CAN frame with an ID that another ECU normally transmits. A simple way to implement anti-spoofing is to use a secure CAN transceiver [1]. The concept is simple:

- Program into the CAN transceiver a list of the CAN IDs that are legal to send from the host.
- Any frame being received from the *bus* that matches an ID on the list is, by definition, a spoofed frame. A check for this takes place soon after arbitration finishes, and a match is deemed an error: the CAN state machine in the transceiver generates an error and all CAN controllers handle it in the normal way.
- Any frame being sent from the *host* that has an ID *not* on the list is also deemed an error (i.e. the host is originating a spoofed frame).

The approach is very simple and provides effective anti-spoofing. But there are several issues:

- **Secure and robust list re-programming**. The list will be programmed in the factory but there must also be an opportunity to re-program the list in the field: an OTA firmware update might add new CAN frames to the application in the ECU and the IDs will have to be added to the list (and, potentially, others removed). The programming mechanism must be secure, too (if software in the host can re-write the list then it can just disable the mechanism). The lists everywhere must be programmed atomically with the ECU firmware: if any failure occurs then the whole system must be in a workable state while changes are rolled back.
- **Arbitration Doom Loop**. A spoofed frame will be destroyed by the CAN error handling mechanism, all controllers will resynchronise, and a new arbitration process will start. The CAN controller sending the spoofed frame (either in host MCU or in a different ECU) will likely immediately re-enter the spoof into arbitration and the Arbitration Doom Loop (described earlier) will be entered.

The anti-spoofing transceiver does not prevent denial-of-service attacks. The Arbitration Doom Loop is a form of denial-of-service attack (typically transmitting a burst of 32 frames before the CAN controller goes bus-off).

It is possible to put into hardware a simple ‘bucket’ rate limiting algorithm to detect and block a crude Flood Attack, but this does not prevent disruption of the timing behaviour of legitimate CAN frames, nor other low-level attacks on the CAN protocol via the CAN TX pin. The protection of real-time traffic behaviour must be matched with the known behaviour of the bus (as discussed earlier with reference to intrusion detection systems). This is in general too complex to be put into hardware alone.

Th above illustrates a general point with security: **security mechanisms** should be **separate from security policies**. The mechanisms (e.g. destroying a spoofed CAN frame) are simple and suitable for hardware, but the policies (e.g. whether a CAN frame is legal in the current context) is usually specific to an application and must therefore be decided by software that forms part of the application.

6.3 Bus guardian hardware

Canis Automotive Labs has developed hardware to provide anti-spoofing and protect against denial-of-service attacks: Mercury Bus Guardian. It is placed between the CAN TX pin from the host MCU and the transceiver:

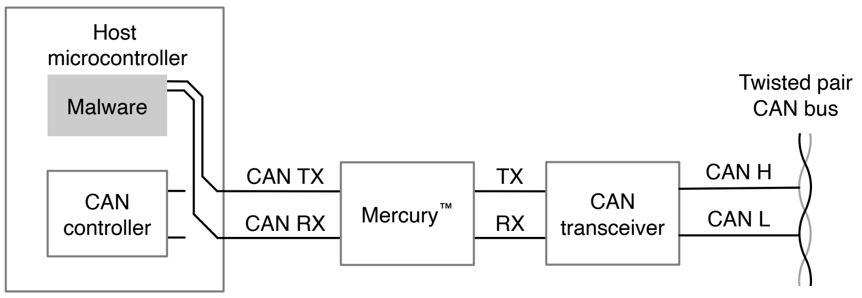


Figure 14: Mercury Bus Guardian device

Mercury Bus Guardian does the following:

- It passes through the CAN TX signal from the host but adds a header in to the CAN frame. The header contains a 7-bit source address and is encoded with *fast bits*. These are described below.
- It monitors the CAN TX signal from the host MCU to check if it is a well-formed and legitimate CAN frame. It will suspend the CAN TX signal passthrough if not.
- It accepts commands over CAN from a trusted security supervisor to suspend and resume the CAN TX signal passthrough.
- It examines other CAN frames on the bus for fast bit headers and if any is seen with matching source address it treats this as a spoofing error and triggers the normal CAN error handling mechanism.

There is no list of CAN IDs stored in a Mercury Bus Guardian – the only configuration information is the source address and the CAN bus bit rate settings. There is no need for any reconfiguration.

Fast bits are extra bits injected into the spare time *inside* a CAN bit. These are carefully placed to take advantage of the CAN protocol rules for when edges are not used for resynchronisation. The placement means that the fast bits are not seen by regular CAN controllers: fast bits are in effect out-of-band signals. This is illustrated below:

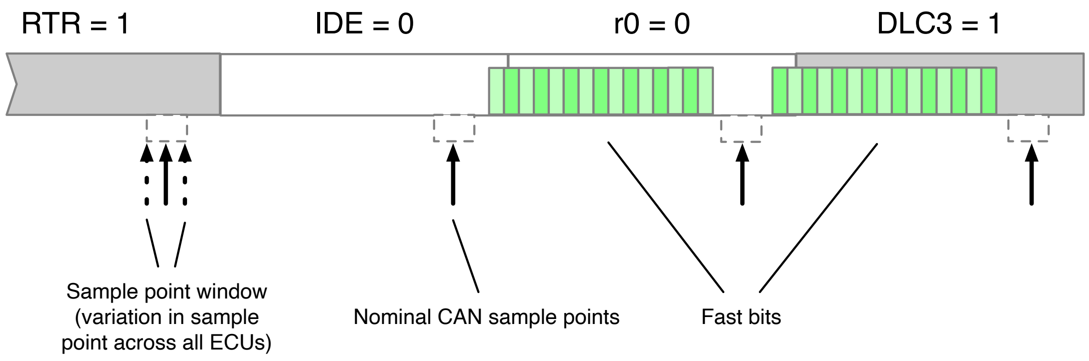

RTR = 1
IDE = 0
r0 = 0
DLC3 = 1
Sample point window (variation in sample point across all ECUs)
Nominal CAN sample points
Fast bits

Figure 15: Fast bits injected into a CAN frame

The fast bits in the diagram above are located outside the sample point window (i.e. where no CAN controllers will sample the CAN bus to obtain a value for the CAN bit). They start after the sample point where the sample is a dominant bit and finish before the sample point window that will read a recessive bit. The diagram shows a standard ID remote frame, where RTR=1 and IDE=0.[2]

Fast bits are normally transmitted at 10Mbit/sec (Canis Automotive Labs has designed a device to use these fast bits through a large 'carrier' CAN frame to hold 96 bytes of application payload while remaining backwardly compatible with CAN 2.0). The header is located in the IDE and r0 bits for standard 11-bit ID frame and in the r1 and r0 bits for an extended 29-bit ID frame.

A logic analyser trace of a CAN frame with a fast bits header is shown below:

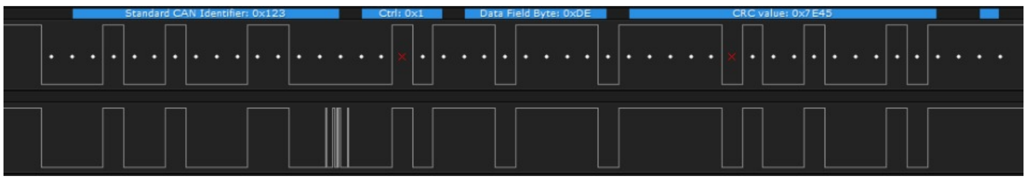


Figure 16: CAN TX from an MCU microcontroller (top line) and CAN TX into the transceiver (bottom line) with the header

The CAN frame has a standard 11-bit ID of 0x123 and a single byte payload of 0xde. What look like glitches are fast bits encoding the header.

[2] In this example the DLC field is 8 (remote frames can have a non-zero DLC; this is a corner case in the ISO 11898 CAN specification).

A hardware-assisted intrusion detection system (discussed earlier) with a Mercury IDS controller from Canis Automotive Labs obtains the header and makes it available to the ISR run after end of arbitration. The IDS software can decide if the frame is legal based on a security policy. Policy decisions might include the following:

- The CAN ID indicates it should come from one ECU but the header indicates it is coming from another ECU. It must be a spoof and should be destroyed. The IDS should send a command to suspend CAN TX passthrough at the hijacked ECU.
- Diagnostic tester frames should come from an external device without the hardware to add a header. If a header is present then the frames are probably coming from a hijacked ECU.
- A header is not present, but the CAN ID indicates the frame should come from an ECU with a Mercury Bus Guardian to add the header. The frame is probably being spoofed by a device directly wired to the bus. It should be destroyed, and the specifics of the attack noted for forensic analysis.

The policy is implemented in software and can be as sophisticated as required. Firmware updates allow the policy to be changed to adapt to new circumstances.

The ability to command a Mercury Bus Guardian to suspend CAN TX passthrough means that denial-of-service attacks can be halted. The command is contained in a CAN frame with CAN ID 0 – the highest priority CAN frame – which will always win arbitration and override a flood attack. This also mitigates the effect of the Arbitration Doom Loop: the loop is terminated after one frame.

The IDS can also protect the bus against an ECU going outside permitted real-time behaviour: the long-standing Babbling Idiot Problem of defective software, and malware transmitting frames too quickly. Mercury IDS provides timestamps to the IDS software and an arbitrarily sophisticated timing model can be used to check traffic is legal.

Mercury Bus Guardian prevents an attack on the command CAN frame from the IDS: CAN ID 0 is reserved for the IDS and CAN TX passthrough is automatically suspended if the host tries to send a CAN frame with this reserved ID.

Mercury Bus Guardian protects against low-level protocol attacks. For example, it monitors the CAN TX pin for spurious errors (i.e. the host MCU signals an error but the CAN state machine in Mercury Bus Guardian has not seen an error) and will automatically suspend for a short time the CAN TX passthrough – long enough for the IDS to transmit a CAN frame with a command to permanently suspend it, blocking a Bus-off Attack. Other low-level protocol attacks are also detected and automatically stopped.

7 Summary

7.1 Comparison of mitigation techniques

The table below summarises how different types of attack can be mitigated by different techniques. The group "Untrusted source" is for attacks originating from sources that should not be trusted (e.g. the OBD-II connector or the infotainment device). The group "Hijacked trusted ECU" is for attacks originating from ECUs on the trusted bus that have been hijacked by at attacker exploiting a hole in the perimeter defences. The group "Direct wiring" is for attacks originating from an external device directly wired on to the CAN bus.

Attack access	Untrusted source						Hijacked trusted ECU						Direct wiring						
Attack type	Bus-off	Flood	Adaptive spoof	Error Passive spoof	Double Receive	Freeze Doom Loop	Bus-off	Flood	Adaptive spoof	Error Passive spoof	Double Receive	Freeze Doom Loop	Bus-off	Flood	Adaptive spoof	Wire-cutting spoof	Error Passive spoof	Double Receive	Freeze Doom Loop
Software IDS	○	○5	○			○	○	○5	○			○	○	○5		○			○
Security gateway	●	●1	●	●	●1	●													
Encryption			◑2	◑2	●				◑2	◑2	●				●	●	●	●	
Whitelisting transceiver		◑3	●	●				◑3	●	●				◑3	●		●		
Bus guardian with hardware IDS	●	●	●	●	◑4	●	●	●	●	●	◑4	●	●	●	●		●	◑4	●

○	Can detect attack
●	Can prevent attack
◑	Partial prevention
1	Only for if real-time drop rules configured
2	Does not apply for frames where hijacked ECU has the authentication key
3	Protection not compatible with timing analysis
4	Prevented except for first incident
5	Cannot distinguish between double-receive and a real-time violation

It is clear from the above that the most effective way to protect a CAN bus from attacks is to adopt a hardware security device with a hardware-assisted IDS. The specific advantages of the hardware solution are:

- Provides protection for denial-of service attacks from hijacked ECUs (neither security gateway nor encryption can do this).
- More efficient than encryption for anti-spoofing: the number of logic gates required for the Mercury Bus Guardian is considerably smaller than those for a hardware AES accelerator or hardware security module (HSM).
- Minimal configuration: a source address (that can be set once in the factory) is the only necessary configuration of a device. There is no need to distribute and re-distribute keys.
- No software is required on any ECU (except in the IDS). This means the hardware can be easily added to the PCB design of an ECU.
- Uniquely protects against low-level CAN protocol attacks such as the Bus-off Attack.
- Security policy is separated from security mechanisms.

To protect against partitioning and re-wiring of the CAN bus, extra measures are required. For example, encryption could be used (in particular, the authentication of messages) to protect against odometer spoofing hardware that isolates a dashboard ECU behind a spoofing gateway device (as described earlier).

CAN bus security is an important issue that should be addressed specifically. But it should always be considered as part of a wider security strategy, with policies and mechanism in areas ranging from an OEM's infrastructure to secure firmware updates booting in a microcontroller.

Often it is not possible to immediately adopt the ideal security solution. But in these cases, it is better to introduce something to mitigate attacks than do nothing.

7.2 CAN security recommendations

The following is a list of recommendations for improving CAN security.

1. **Prevent double receive errors**. The software should include a sequence number in the CAN frame so that an automatic retransmission by the hardware can be detected and spurious frames discarded. A single bit that is toggled by the CAN drivers is sufficient.
2. **Perform timing analysis**. Key to detecting and preventing bus flood attacks is knowing the worst-case real-time behaviour of legitimate bus traffic.
3. **Do not allow ODB-II or infotainment direct access to the control CAN bus in a vehicle**. These are the two biggest threats to CAN bus security. They should be behind a security gateway.
4. **Use hardware interlocks for critical operations**. Do not allow OTA downloads or disable airbag commands in a vehicle to be activated without a hardware interlock that a physically present human must activate.
5. **Use hardware to guard the CAN TX pin.** The low-level protocol attacks depend on direct access to the CAN TX pin. Use an external CAN controller connected via SPI, a microcontroller with a pin multiplexer that can be permanently locked to the internal CAN controller or the Mercury Bus Guardian.
6. **Use an IDS to log suspicious traffic**. At a minimum there should be evidence collected for post-incident analysis.
7. **Use a hardware-assisted IDS to prevent attacks**. An IDS that can destroy CAN frames before they are received can guard ECUs from spoofing attacks. Attacks originating from ECUs with Mercury Bus Guardian can be shut down.
8. **Use encryption to protect against re-wiring attacks**. For situations where an attacker is motivated to re-wire a CAN bus then encryption with authentication should be used to protect critical messages (such as odometer readings).

8 Change history

Version 7

Added description of wire-cutting spoofing attack and photograph of a device for spoofing odometer readings (section 2.5).

Updated description of encryption section to note that it protects against re-wiring attacks (section 5.1).

Updated mitigations comparison table to include the wire-cutting spoofing attack (section 7.1)

Updated recommendations to include use of cryptographic authentication to protect messages that could be attacked by re-wiring (section 7.2).

9 References

1 “Cyber security enhancing CAN transceivers”, Elend, B., Adamson, T., International CAN Conference 2017, pp. 08-1 to 08-4.

2 “Road Vehicles – Controller Area Network (CAN) – Part 1: Data link layer and physical signalling”, ISO 11898-1, second edition 2015-12-15, note at end of section 10.7 “Frame validation”.

3 “Buridan’s Principle”, Lamport, L., Foundations of Physics 42(8), August 2012.

4 “Error Handling of In-Vehicle Networks Makes Them Vulnerable”, Cho, K-T., Shin, K., 23rd ACM Conference on Computer and Communications Security, Vienna, 2016.