CAN-based bootloaders: Advantages and Disadvantages of CANopen bootloaders, J1939 DM bootloaders, J1939 CAM11/CAM21 bootloaders, and UDS bootloaders

Torsten Gedenk (emotas embedded communication)

NMEA 2000 is a plug-and-play communications CAN-based standard used for connecting marine sensors and display units within ships and boats. It sits amongst other NMEA marine communications protocols from NMEA 0183 at the lower-end through to the Ethernet-based NMEA ONENET standard. NMEA 2000 itself uses many of the features that are in common with SAEJ1939 and ISO11783. The standard has enabled the easy integration of electronic devices into a vessel.
However, as with all CAN-based protocols, several vulnerabilities to cyberattacks have been identified. Many are at the CAN level, whilst others are in common with those protocols from the SAEJ1939 family of protocols.
This paper will discuss the known vulnerabilities that have been identified with the NMEA 2000 protocol. These include weaknesses with the address claim and transport protocols, and covert communication channels using methods based on steganography. Activities with the aim of making NMEA 2000 robust to cyberattacks are described.

I. Tasks of an embedded bootloader

A bootloader usually used in embedded devices is a piece of firmware which is usually located at the start of the program area and thus which is started at a reset of the of the ECU. The primary tasks of an embedded bootloader are the following:

- start of application
- verification of application
- reception of new application in case of an update
- authentication of update tool (if possible)

In order to achieve these tasks the bootloader must be able to erase and write to the flash memory of the ECU and there must be a way to communicate with an update tool. In a CAN network it requires the support of a CAN-based higher layer protocol

II. Requirements for embedded bootloaders

Fast transmission: In order to reduce the required time to update the firmware of an ECU, the fast transmission of the data is one of the most important requirement and it is the one which is affected most by CAN and the higher-layer protocols. Because of this the data transfer for all different protocols will be examined in a separate section of the this paper.

Fast start of the application: The time the bootloader needs to check if the application is valid should be as short as possible in order to achieve a fast boot-up of the ECU. For small application it is common that the bootloader calculates a checksum for the application at each start, but this approach takes a longer time with larger application. Thus it is also common that the checksum of the stored application is only calculated once after the download and the specific check of the application memory at each start is skipped.

Checks of the application and download tool
The bootloader needs to perform a couple of checks on the application and on the download tool. It has to verify that the downloaded firmware is suitable for the ECU, that the data of the firmware has not been modified during the transfer and that the download tool is allowed to perform the action.

The plausibility check to verify that a firmware is suitable for a target is usually proprietary and does not depend on the communication protocol. A suitable method is to prefix the .bin or .hex file of the application with a header containing metadata that is checked in the bootloader.

For data transfer, one could rely on the data link layer and its capabilities, but most download files include a checksum in the header that is compared with a calculated one after or during the download.

Authentication of the download tool is an important issue, but the capabilities of the various higher layer protocols vary widely.

Another requirement is access to different memory areas, e.g. one for the application and additional areas for configuration or calibration data. Some protocols only allow access to fixed areas, while others allow arbitrary access to memory addresses.

Another topic that is more and more important is the encryption of the firmware. Thus a bootloader should be able to decrypt an encrypted firmware. Although some protocols define a way to signal if an application is encrypted or compressed, the encryption or compression is proprietary in all cases.

Small code size: As with all requirements most of the discussed features increase both the complexity of the bootloader and code size as well. So occasionally one might have to accept that not all requirements can be met if a specific code size shall not be exceeded.

The table below shows an example memory layout with 16 KiB flash memory used by the bootloader.

Table 1: Flash layout example

start address	end address	usage of flash memory
0800’0000h	0800’3FFFh	bootloader
0800’4000h	0800’40FFh	configuration data
0800’4100h	081F’EFFFh	application
081F’F000h	081F’FFFFh	additional data

III. A brief introduction into CAN based protocols

The paper does not aim to explain all mentioned CAN based higher-protocol and in order to avoid unnecessary complexity the explanation will focus on classical CAN and the transport protocols provided by these CAN based protocols. Additionally, only the classical CAN variants of the protocols are discussed, but all principles apply to CAN FD as well.

In CANopen a new firmware will be written to a CANopen object (e.g. 0x1F51:1) and the data will be written to the object via a so called SDO transfer. The CANopen specifications do not define any memory addresses where the object will be stored and the protocol does not allow to define any memory addresses, so the assignment of a memory address is implemented implicitly in the bootloader.

There are three different variants of an SDO transfer but only two of them allow the transfer of data exceeding the length of a CAN message. The CANopen segmented SDO transfer starts with an initialization message and after that the payload is transferred in CAN frames, which contain one protocol byte and seven bytes of payload.

CAN-ID	Type	Len	0	1	2	3	4	5	6	7	
0x67f	EXT RTR	8	0x21	0x50	0x1f	0x01	0x33	0x00	0x00	0x00	SDO segmented write request
0x5ff	EXT RTR	8	0x60	0x50	0x1f	0x01	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x00	0x65	0x6d	0x6f	0x74	0x61	0x73	0x20	1st data
0x5ff	EXT RTR	8	0x20	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x10	0x6c	0x69	0x65	0x66	0x65	0x72	0x74	2nd data
0x5ff	EXT RTR	8	0x30	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x00	0x20	0x53	0x74	0x61	0x63	0x6b	0x73	3rd data
0x5ff	EXT RTR	8	0x20	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x10	0x20	0x66	0xc3	0xbc	0x72	0x20	0x43	4th data
0x5ff	EXT RTR	8	0x30	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x00	0x41	0x4e	0x6f	0x70	0x65	0x6e	0x2c	5th data
0x5ff	EXT RTR	8	0x20	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x10	0x20	0x4a	0x31	0x39	0x33	0x39	0x20	6th data
0x5ff	EXT RTR	8	0x30	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x00	0x75	0x6e	0x64	0x20	0x55	0x44	0x53	7th data
0x5ff	EXT RTR	8	0x20	0x00	0x00	0x00	0x00	0x00	0x00	0x00	confirmation
0x67f	EXT RTR	8	0x1b	0x2e	0x0a	0x00	0x00	0x00	0x00	0x00	8th dat
0x5ff	EXT RTR	8	0x30	0x00	0x00	0x00	0x00	0x00	0x00	0x00	final confirmation

Figure 1: CANopen Segmented SDO Transfer

Each CAN frame of a segmented SDO transfer is confirmed by another CAN frame from the ECU. This simplifies the implementation and reduces the problem of receiving multiple CAN with the same CAN-ID messages back-to-back, but it delays the firmware download significantly. In order to

improve the SDO transfer speed, another SDO transfer - the SDO block transfer had been standardized. The SDO block transfer also starts with an initialization messages, but after that the payload is transferred in blocks of CAN messages up to 127 CAN frames that are confirmed after one block.

CAN-ID	Type	Len	0	1	2	3	4	5	6	7	
0x67f	EXT RTR	8	0xc2	0x50	0x1f	0x01	0x33	0x00	0x00	0x00	SDO block transfer write req
0x5ff	EXT RTR	8	0xa0	0x50	0x1f	0x01	0x06	0x00	0x00	0x00	response
0x67f	EXT RTR	8	0x01	0x65	0x6d	0x6f	0x74	0x61	0x73	0x20	sub-block data 1
0x67f	EXT RTR	8	0x02	0x6c	0x69	0x65	0x66	0x65	0x72	0x74	sub-block data 2
0x67f	EXT RTR	8	0x03	0x20	0x53	0x74	0x61	0x63	0x6b	0x73	sub-block data 3
0x67f	EXT RTR	8	0x04	0x20	0x66	0xc3	0xbc	0x72	0x20	0x43	sub-block data 4
0x67f	EXT RTR	8	0x05	0x41	0x4e	0x6f	0x70	0x65	0x6e	0x2c	sub-block data 5
0x67f	EXT RTR	8	0x06	0x20	0x4a	0x31	0x39	0x33	0x39	0x20	sub-block data 6
0x5ff	EXT RTR	8	0xa2	0x06	0x06	0x00	0x00	0x00	0x00	0x00	sub-block confirmation
0x67f	EXT RTR	8	0x01	0x75	0x6e	0x64	0x20	0x55	0x44	0x53	sub-block data 1
0x67f	EXT RTR	8	0x82	0x2e	0x0a	0x00	0x00	0x00	0x00	0x00	final data
0x5ff	EXT RTR	8	0xa2	0x02	0x06	0x00	0x00	0x00	0x00	0x00	sub-block confirmation
0x67f	EXT RTR	8	0xd5	0x00	0x00	0x00	0x00	0x00	0x00	0x00	write end request
0x5ff	EXT RTR	8	0xa1	0x00	0x00	0x00	0x00	0x00	0x00	0x00	write end response

Figure 2: CANopen SDO Block Transfer

J1939 CAM11/CAM21 is a way to use the CANopen SDO transfers in J1939, but the J1939 Connection Mode Data Transfer (CMDT) is much more common. J1939 Connection Mode Data Transfer (CMDT) implements a hand-shake between to nodes. The transfer starts with a Request-to-Send (RTS) message where the data and the size are announced. The ECU replies with a Clear-to-Send (CTS) message indicating the maximum number of CAN message in one block. The download sends a block of up to 255 messages which are confirmed by a CTS message again. The transfer is closed with an End-of-Message-Acknowledge at the end of the transfer.

CAN-ID	Type	Len	0	1	2	3	4	5	6	7	
0x1cec54f9	EXT RTR	8	0x10	0x31	0x00	0x07	0x0d	0x00	0xef	0x00	RTS
0x1cecf954	EXT RTR	8	0x11	0x06	0x01	0xff	0xff	0x00	0xef	0x00	CTS block size = 6
0x1ceb54f9	EXT RTR	8	0x01	0x65	0x6d	0x6f	0x74	0x61	0x73	0x20	data 1
0x1ceb54f9	EXT RTR	8	0x02	0x6c	0x69	0x65	0x66	0x65	0x72	0x74	data 2
0x1ceb54f9	EXT RTR	8	0x03	0x20	0x53	0x74	0x61	0x63	0x6b	0x73	data 3
0x1ceb54f9	EXT RTR	8	0x04	0x20	0x66	0x75	0x65	0x72	0x20	0x43	data 4
0x1ceb54f9	EXT RTR	8	0x05	0x41	0x4e	0x6f	0x70	0x65	0x6e	0x2c	data 5
0x1ceb54f9	EXT RTR	8	0x06	0x20	0x4a	0x31	0x39	0x33	0x39	0x20	data 6
0x1cecf954	EXT RTR	8	0x11	0x01	0x07	0xff	0xff	0x00	0xef	0x00	CTS
0x1ceb54f9	EXT RTR	8	0x07	0x75	0x6e	0x64	0x20	0x55	0x44	0x53	data 7
0x1cecf954	EXT RTR	8	0x13	0x31	0x00	0x07	0xff	0x00	0xef	0x00	EACK

Figure 3: J1939 CMDT Transport Protocol (block size 6 only to illustrate the protocol)

UDS uses ISO-TP as transport protocol. ISO-TP is standardized in ISO 15745-2 and is merely a transport protocol without any application layer. A ISO-TP transfer also starts with an initialization message (First Frame) followed by a Flow Control message and multiple Consecutive Frames, which contain one byte of protocol data and seven bytes of payload as the other protocols. The Flow Control message contains a block size indicating the number of CAN messages per block and a separation time to force the transmitter to introduce gaps between the CAN messages.

CAN-ID	Type	Len	0	1	2	3	4	5	6	7	
0x401	EXT RTR	8	0x10	0x33	0x65	0x6d	0x6f	0x74	0x61	0x73	First Frame
0x400	EXT RTR	8	0x30	0x06	0x28	0xcc	0xcc	0xcc	0xcc	0xcc	Flow Control.CTS
0x401	EXT RTR	8	0x21	0x20	0x6c	0x69	0x65	0x66	0x65	0x72	Consecutive Frame
0x401	EXT RTR	8	0x22	0x74	0x20	0x53	0x74	0x61	0x63	0x6b	Consecutive Frame
0x401	EXT RTR	8	0x23	0x73	0x20	0x66	0xc3	0xbc	0x72	0x20	Consecutive Frame
0x401	EXT RTR	8	0x24	0x43	0x41	0x4e	0x6f	0x70	0x65	0x6e	Consecutive Frame
0x401	EXT RTR	8	0x25	0x2c	0x20	0x4a	0x31	0x39	0x33	0x39	Consecutive Frame
0x401	EXT RTR	8	0x26	0x20	0x75	0x6e	0x64	0x20	0x55	0x44	Consecutive Frame
0x400	EXT RTR	8	0x30	0x06	0x28	0xcc	0xcc	0xcc	0xcc	0xcc	Flow Control.CTS
0x401	EXT RTR	8	0x27	0x53	0x2e	0x0a	0x00	0x67	0x00	0x73	Consecutive Frame

Figure 4: ISO-TP

So from a wider perspective the CANopen SDO block transfer, the CMDT transport protocol from J1939 and ISO-TP are very similar. Without gaps between the CAN messages all three transport protocols offer the same performance, which is significantly higher than the CANopen segmented SDO transfer.

IV. Bootloader features of the different protocols

CANopen according to the specification CiA 302 defines the following sequence of actions

- erase of a fixed memory section
- firmware transfer via segmented SDO transfer or SDO block transfer to a fixed memory section
- read of checksum to verify the correct download

Multiple memory sections can be defined but there is no defined way to write to a specific flash memory address. Furthermore CANopen according to CiA 302 does not provide standardized mechanisms for authentication, identification of the download tool and security handling. Certain CANopen application profiles, which define domain-specific functionality for specific application such as e.g. elevators or light-electric vehicles, have defined password objects to prevent unauthorized access to the bootloader. However these password objects an easily be defeated by reply attacks.

The J1939 PGNs CAM11/CAM21 provide a way to transmit 'CANopen Application Messages' in a J1939 network. The J1939 Digital Annex reserves the 2 PGNs for CANopen and the CANopen specification CiA 510 defines how to use the SDO protocols within these PGNs. So a J1939 CAM11/CAM21 bootloader is a variant of a CANopen bootloader in a J1939 environment. Reasons to use such a bootloader may be existing CANopen knowledge that shall be reused and to avoid the complexity of a J1939 DM bootloader.

J1939 itself defines a set of so called diagnostic messages (DM) in the specification J1939-73. A subset of these diagnostic message is suitable for a firmware download:

- DM14 – Memory Access Request
- DM15 – Memory Access Response
- DM 16 – Binary Data Transfer
- DM 17 – Boot Load Data
- DM18 – Data Security

In general these Diagnostic Messages allow a specification of a specific address in the device's memory and a seed-key based authorization of the download tool before the transmission of the data. A fingerprint to store an identification of the download tool or repair shop is not standardized with J1939.

UDS defines various so called sessions and the bootloader is usually active after a transition into the programming session. Write access to the device's flash can be granted via a Seed-Key-Mechanism or an Authentication mechanisms and the UDS service RequestDownload allows to transmit the specific address and the size of the data. It is possible to write a fingerprint into an ECU to identify the download tool or repair shop. After that the firmware transfer itself is realized by multiple TransferData services which use ISO-TP as transport protocol. Additionally, there are services to check the transferred firmware by checking a CRC or other means.

V. CiA TF Generic CAN Bootloaders

The CiA Task Force generic CAN Bootloaders started its work in 2021 and finished the CiA specification CiA 710 in 20241. Despite the name of the working group the work was focused on improving the CANopen bootloader capabilities. The transfer speed for CANopen using the SDO block transfer was already at par with other protocols, but CANopen bootloaders according to CiA 302 lack the flexibility of other protocols and no security mechanisms had been standardized so far for CANopen.
The result of work was the CiA 710 specification which adds additional objects to the CANopen object dictionary, and these objects provide additional features for the bootloader:

- object 0x1F59 –Autostart behavior of the applications

Definition which application shall be started, if applicable.

- object 0x1F5A – Application security access

Read access to this object provides a seed for a security algorithms and writing a key to this object unlocks write access to the memory. The algorithm itself is manufacturer-specific.

- object 0x1F5B – Mode switch delay

Definition of possible delays between the transition from application to bootloader or vice versa.

- object 0x1F5C – Bootloader mode switch

Writing a specific value to this object triggers the transition to the bootloader (if allowed).

- object 0x1F5D – application version string

String to identify the version of each program. The format of this string is manufacturer-specific.

- object 0x1F5E – application or bootloader identity

String to identify the application or bootloader. The format of this string is manufacture-specific.

- object 0x1F6F – Timeout flash operations

Definition of a maximum timeout for flash operations (erase, write) when the device may be non-responsive.

So most of the weak points of the previous CANopen bootloaders had been overcome and also additional features such as autostart configurations and transition delays had been added.

VI. Comparison of various bootloader approaches

The following table compares and summarizes the various bootloader approaches with the different higher-layer CAN protocols.

Table 2: Comparision of various bootloader (+ good/fast/small size, o average, - no feature/slow)

	CANopen segmented SDO	CANopen SDO block Transfer	J1939 DM CMDT	J1939 CAM11/CAM21 SDO Block Transfer	UDS ISO-TP
flash size	+	o	o	o	o
transfer speed	--	+	+	+	+
flexible memory addresses	-	-	+	-	+
security access	- (o with CiA 710)	- (o with CiA 710)	+	- (o with CiA 710)	+
identification (fingerprint)	- (+ with CiA 710)	- (+ with CiA 710)	-	- (+ with CiA 710)	+
wide spread use	o	o	-	--	+

VII. Conclusion and outlook

The comparison chart shows that all variants offer the same speed, but a UDS based CAN bootloader offers best results regarding flexibility and security. With its efficient use of CAN-IDs - only two or three are required – it can also be used in parallel with other protocols in the same network.

CAN FD has not been discussed in detail in this paper but with a frame length of 64 bytes all statements in this protocol also apply to CAN FD.

Finally, possible CAN XL based implementations of the discussed higher-layer protocols have not been standardized yet. Anyway, one can assume that any firmware will not fit into a single CAN XL frame and so transport protocols will also be required in the future. Due the higher speed and the longer payload of CAN XL the performance criteria will become less crucial and the additional features of the different variants such as encryption, security and authentication will become more important.

References

[1] CiA 301, Version 4.2.0, CANopen application layer and communication profile
[2] CiA 302, Framework for CANopen managers
[3] SAE J1939-21, Data Link Layer
[4] SAE J1939-73, Application Layer-Diagnotics
[5] ISO 15765-2, Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services
[5] ISO 14229-1, Road vehicles – Unified diagnostic services (UDS), Part 1: Application layer

Torsten Gedenk
emotas embedded communication
Fritz-Haber-Str. 9
DE-06217 Merseburg
www.emotas.de