2022 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
CYBERSECURITY OF GROUND SYSTEMS TECHNICAL SESSION
AUGUST 16-18, 2022 - NOVI, MICHIGAN

DEFENDING CONTROLLER AREA NETWORK (CAN) BUSES

Dr. Kenneth Tindell[1]

[1]CTO, Canis Automotive Labs, UK

ABSTRACT

Modern ground vehicles rely on Controller Area Network (CAN) bus for communication between Electronic Control Units (ECUs) as a vital component to connect sensors and actuators together in a mission-critical distributed real-time vehicle control system. CAN is well-suited to this task and over the more than three decades since its inception it has become a proven and ubiquitous technology. But its age means that it was not designed for modern security threats of local and remote attacks and special techniques must be deployed to protect CAN. This paper provides a simple taxonomy of attacks on CAN, including how an attack accesses a CAN bus, and discusses four techniques used to defend against these attacks.

Citation: K Tindell, "Defending In-vehicle CAN Buses From Attacks," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 16-18, 2022.

1. INTRODUCTION

CAN bus was created in the mid-1980s to provide a robust atomic broadcast system to connect ECUs in passenger cars to replace individual signaling wires. Since its inception it has become a proven technology in vehicles as diverse as yachts and spacecraft and is ubiquitous in system ground vehicles because of the wide availability of off-the-shelf components (from standard microcontroller silicon to ECUs). CAN was never designed with security in mind – in the mid-1980s there was no notion of any embedded systems being connected to the internet let alone vehicles. The next section will give a simple taxonomy of the attacks on CAN. Then different ways to defend CAN will be presented, including a novel hardware-based approach.

2. TAXONOMY OF ATTACKS

The 'CIA' triad of security is a useful model for security: *C*onfidentiality, *I*ntegrity, *A*vailability. In a vehicle control system, confidentiality is the least important because it is mostly sensor data and actuator commands (although there can be some information that is sensitive, such as firmware being downloaded to re-program ECUs). Integrity, on the other hand, is a vital property: when an ECU connected to an actuator receives a command to move then it must be that the command is genuine. In CAN a broadcast message (called a *frame*) contains an *identifier* that indicates the contents and priority of the frame. ECUs use the identifier to determine how to (or whether to) act on the frame's contents. There is no inherent protection preventing a device connected to CAN bus from deliberately sending a CAN frame an identifier normally used by another ECU. Such a frame is usually called a *spoof* and protection against spoofing is the primary requirement when defending CAN.

Availability is the second key requirement for a secure system: the system must be able to defend against attacks that prevent legitimate communication. For a mission-critical distributed real-time control system, disrupting communications will cause it to fail. CAN includes some protections to prevent a failing device from disrupting the bus (for example, error confinement rules leading to a ‘bus off’ state where the failing device is disconnected from the bus). But these protections are designed for component failure and can be trivially avoided by a denial-of-service attack.

There are two ways to mount an attack on a CAN bus. The simplest is a *frame attack*: to use existing CAN hardware – the CAN *controller* – to send frames on the bus. For example, to send a spoof frame. Or to send frames at such a rate that legitimate frames are slowed or prevented: a *flood attack* (in a real-time system, a message arriving late is a failure of the system). The other way to mount an attack is to by-pass the CAN hardware and drive carefully crafted signals directly on to the bus: a CAN *protocol attack*.

A CAN protocol attack uses software to drive signals that exploit low-level features of the CAN protocol itself. For example, the *bus-off attack* uses the CAN error confinement rules to disconnect a targeted ECU from the bus. A CAN protocol attack uses direct access to the standard component that all ECUs contain: the CAN *transceiver*. The transceiver is a chip that converts between the digital TX and RX I/O from the CAN controller and the analog voltages of CAN H and CAN L on the twisted pair CAN wire. Normally the TX and RX pins are driven by a CAN controller but in most ECU electronics this controller is integrated inside a single microcontroller chip alongside the CPU, RAM, etc. But software can disable the CAN controller and take control of the I/O pins directly (using the *pin mux* that is present in some form on all microcontrollers).

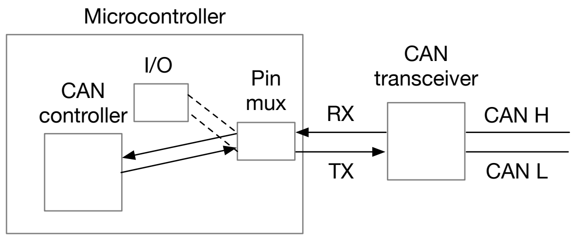

Figure 1: CAN controller, transceiver and pin mux

With carefully written software it is possible to drive the TX pin quickly enough to emulate parts of the CAN protocol and mount protocol attacks.

An attacker must gain access to the CAN bus to attack it. This can be done in two broad ways: a *wired* attack where the attacker directly attaches their own hardware to the CAN bus, and a *hijack* attack where the attacker takes over existing hardware connected to a CAN bus. A hijack is typically carried out by exploiting a remote code execution (RCE) vulnerability, such as a buffer overrun, in a device connected to CAN. There are many vectors for a hijack attack, from exploiting protocol defects in long-range wireless systems [1] to short-range wireless like tire pressure monitoring systems (TPMS) [2].

From the above, we can define a three-axis taxonomy for attacks on a CAN bus.

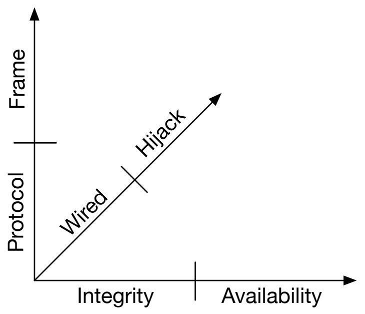

Figure 2: Categories of CAN attack

To illustrate the taxonomy, we will use the Bus-off Attack. This attack was first described by Cho and Shin [3] and operates by software 'bit banging': the RX pin from the CAN transceiver is sampled until the ID field of a CAN frame is detected. If it matches the ID of a frame used by the victim ECU then the software drives the TX pin low for six CAN bit times. This triggers the CAN error handling mechanism, and each time this is done, a counter in the victim ECU's CAN controller is incremented. Eventually the counter reaches a threshold where error confinement rules of the CAN protocol cause the CAN controller to go into 'bus off' mode: the controller refuses to transmit any more frames until it is reset (the process to recover and bring it back online takes quite some time).

The Bus-off Attack can be categorized as follows: it is an Availability attack (denying communication to a victim ECU), and it is a Protocol attack (that exploits the low-level error handling behavior of the CAN protocol) and it works for either wired or hijack access to the CAN bus.

There are several other known CAN protocol attacks [4], including:

- *Double Receive* attack (where a victim is forced to retransmit its frame one or more times)
- *Bus Freeze* attack (where exploiting a legacy feature of the CAN protocol causes the controllers to be held stuck in a protocol loop for an arbitrary time)
- *Error Passive* attack (where a victim's frame can be overwritten by a spoofed payload)
- *Janus* attack (where a single frame can be transmitted but received with different payloads at different controllers)

These attacks can be combined (along with knowledge of how a targeted system behaves) to achieve higher-level attacks.

3. SIMPLE CAN BUS DEFENSES

This section discusses some basic techniques to defend against attacks. Although no single mechanism is sufficient, together they can provide some protection.

The first technique is to protect against CAN protocol attacks where the attacker is accessing the CAN bus via a hijacked ECU. The approach is very simple: design the ECU circuit board to use an external CAN controller rather than an on-chip controller. This removes direct I/O access to the CAN transceiver pins. The CAN bus could still be disrupted by the software setting the CAN controller to use the wrong baud rate, but sophisticated timing attacks like Janus are not possible. A related defense is to lock the pin mux during a secure boot process: an RCE triggered after boot cannot change the pin mux and therefore cannot access the CAN transceiver directly (although not all microcontrollers support pin mux locking – or indeed secure boot).

Another conceptually simple defense is to use an intrusion detection system (IDS): the traffic patterns on a CAN bus should be known at design time because it is an embedded system with a known behavior. Deviation from the known patterns can indicate an attack. Of course, an IDS does not prevent an attack, it only detects an attack with limited confidence. However, providing forensic evidence of an attack and how it took place is essential for hardening a system against a future repeat use of an attack.

The most common technique for defending a CAN bus is the *security gateway*. This is a device like an ECU with two CAN bus interfaces, a *trusted* CAN bus (containing the mission-critical system that needs to be protected) and an *untrusted* side (containing devices that are at the highest risk of being exploited, such as telematics devices with wide area radio network connections). The security gateway is conceptually simple: forward traffic on one bus over to another bus according to a set of rules. Because devices on the untrusted bus have no direct access to the trusted bus, they cannot mount CAN protocol attacks directly, and wired attacks that can access only the untrusted bus also cannot in principle mount attacks on ECUs connected to the trusted bus.

While security gateways do provide some defense against attacks, they also have weaknesses:

- Hijack attacks on ECUs on the trusted bus are still possible by otherwise legitimate traffic containing malware (for example, diagnostic messages designed to exploit commonplace buffering vulnerabilities in the diagnostic stack [5] in a victim ECU).
- Real-time attacks using legitimate messages (of which the flood attack is the simplest) can cause traffic on the trusted side to arrive late and induce a timing fault.
- Implementation problems with the security gateway. For example, buffering problems leading to dropped frames or priority inversion [6].

A security gateway must also be implemented to the highest security levels with no RCE vulnerabilities and there must be a secure mechanism for re-programming its rules.

4. CRYPTOGRAPY ON CAN BUS

A common approach to addressing the 'CIA' triad is to use cryptography: a message is encrypted (keeping its contents *C*onfidential) and a cryptographic message authentication code (MAC) is used to provide message *I*ntegrity. Cryptographic techniques do not assure *A*vailability.

There are specific problems with using cryptographic techniques on CAN:

- CAN frames contain at most 8 bytes of payload, and this is not large enough to hold both a message and a MAC.
- CAN control systems use a 1:*n* or the *publish/subscribe* broadcast model that does not fit well with cryptographic systems designed for 1:1 messaging.
- Fast restart. In powertrain systems it is particularly important that if an ECU goes through a watchdog reset it can recover and return to normal operation very quickly, otherwise an engine could stall.

The CryptoCAN scheme of Canis Labs is designed to fit within the AUTOSAR framework, using AES for encryption and authentication with the SHE standard [7] for hardware security modules (HSMs), and to address these issues. It uses a pair of 8-byte CAN frames to contain the encrypted and authenticated message.

At the transmitter, a 60-bit MAC is obtained from the plaintext message using the AES-CMAC algorithm: a 128-bit block containing the plaintext payload (0-8 bytes), the length of the payload (4 bits) and a 60-bit MAC, computed over the CAN ID, length, plaintext, and a 30-bit 'freshness' value, using the standard AES-CMAC algorithm (Figure 3).

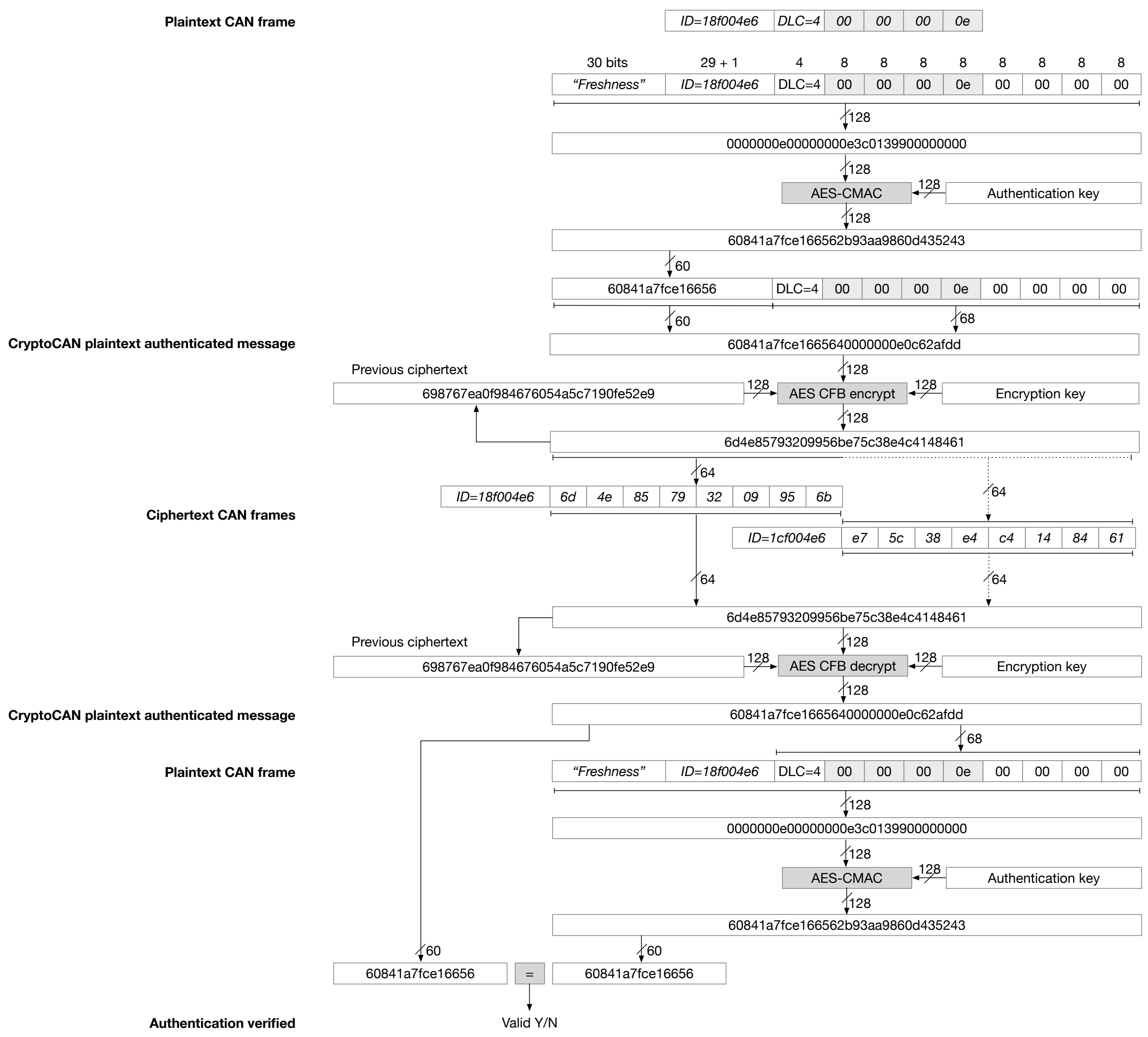

Figure 3: CryptoCAN encryption, decryption, and authentication

The ciphertext is obtained using the cipher feedback (CFB) mode of the standard AES-128 algorithm. The pair of CAN frames are assigned adjacent CAN IDs so that the first of the pair has the higher priority and will be transmitted first. A receiver decrypts and verifies the two-frame message using the reverse process. CFB mode supports random read access so a newly powered ECU can read subscribed messages quickly after power up or reset.

There remain two major problems – common to any symmetric cryptographic scheme. The first is *replay attacks*. With a distributed embedded real-time system, it is straightforward to determine what each CAN message does, even when the contents of the message are kept confidential. It is then straightforward to capture messages that have a known purpose and to re-send them on the CAN bus later when needed. The MAC for such a message will pass verification because of course it was created by the genuine sender. Replay attacks are normally resolved by using sequence numbers or timestamps. While this is straightforward in a 1:1

messaging system it is more difficult in a 1:*n* broadcast system, requiring the transmission of a timestamp to keep ECUs in sync even after an ECU is reset. But that timestamp message itself could be replayed as part of an attack. Resolving this problem is quite difficult and requires a non-cryptographic message authentication technique specific to CAN. This is discussed later.

The second major problem concerns key management. Aside from the normal problems of key management (where unique keys must be reliably and securely programmed into different devices and also stored securely in a central database) there is the specific key problem of 1:*n* broadcast systems.

A MAC guarantees that the transmitter of a message knew the secret key that is shared with the receiver. But in a 1:*n* broadcast system this is a problem: each receiver must know the MAC key to authenticate it. But if one of those receivers were hijacked (exploiting an RCE vulnerability) then it could forge a valid MAC using the key and then transmit a spoof as if sent by the transmitter. This completely undermines the purpose of using cryptography in the first place.

There is a mitigation for the key problem: an SHE HSM stores keys in a secure area of non-volatile memory that the CPU cannot access. It undertakes cryptographic operations requested by the CPU, but the key is kept secret. With the SHE+ extension to the SHE specification, a key can be marked with a flag for *verify-only* and the HSM will reject requests to create a MAC, and only the genuine sender of a message has the key flag set to allow MAC creation. There remains a problem: the HSM must store enough keys so that every sending ECU gets its own key. Unfortunately, the SHE standard defined at most 16 keys (and several of these are reserved for purposes such as secure boot).

In short, encryption defends against spoofing attacks but does not prevent denial-of-service attacks. But without careful implementation it cannot prevent relay spoofing attacks or hijacked ECU spoofing attacks.

5. SIMPLE HARDWARE DEFENSES

Recall from earlier the problem of the message sending a 'freshness' value to be used to prevent replay attacks: this can itself be replayed. A way to solve this problem is to, in effect, mount a bus-off CAN protocol attack on any attempt to spoof the freshness message. The basic idea is simple. The sender ECU listens to the CAN bus (by sampling the RX pin from the CAN transceiver) and looks for a frame with an ID that matches the freshness message. If the ID is seen and did not come from the sender's own CAN controller, then it must be a spoof. Driving the TX pin dominant for six CAN bit times triggers the CAN error mechanism and in effect destroys this spoof frame (and no other ECU will see it). The spoofing ECU's CAN controller will very likely try to re-send it but eventually it will be driven into the bus-off state. Only the genuine freshness message can be sent and therefore replay attacks can be prevented.

This approach can be made more general: it can apply to all the frames sent from an ECU. There are difficulties in implementing this efficiently in software (it requires interrupts to be serviced with very short latencies and leads to a high worst-case CPU load). But it can be implemented in hardware: it monitors the TX and RX signals and is programmed with a list of IDs of frames that are expected to be transmitted. Any frame ID on the list that was received rather than transmitted is destroyed as a spoof. Further, any frame transmitted with an ID not on the list is an attempt to spoof some other ECU and is also destroyed. The TJA115x devices from NXP implement this.

This approach is in effect providing authentication directly in hardware using the atomic multicast feature of the CAN protocol and can in most cases avoid the need for cryptographic solutions.

There are some issues with the hardware approach just described: the ID lists programmed into the hardware must be re-programmable (because the IDs used may change over time with software updates) via a robust and secure mechanism. Furthermore, it does not prevent denial-of-service attacks (the TJA115x devices include a basic anti-flood defense but this does not prevent timed attacks that cause real-time latency problems for targeted messages). But the most important issue is the mixing of security *mechanism* with security *policy*: policy is in general application-specific and needs to be able to activate or deactivate mechanisms. For example, an ECU may need to send firmware updates over CAN but only when in a specific mode (which might be only at a known location, with the vehicle stopped, perhaps with a physical switch or key in place). In general, security policy should be in software.

6. HARDWARE DEFENSES: CAN-HG

CAN-HG is a new hardware augmentation of the CAN protocol, developed by Canis Labs to provide hardware security without the drawbacks of the simple list-based approach described above. It works by adding high-speed out-of-band data into an outgoing CAN frame. Figure 3 shows a timeline of three CAN bits (1, 0, 1).

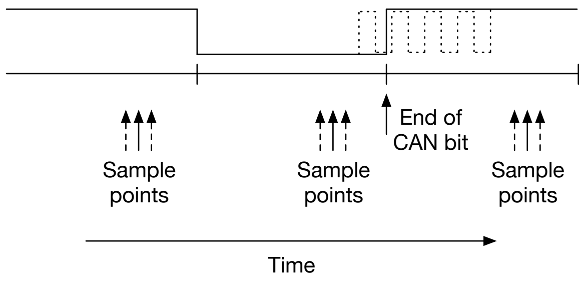

Figure 4: Out-of-band data in CAN

The possible sample points are illustrated. For a given CAN bus system there will be a notional point where each CAN controller samples the transceiver's RX pin (for SAE J1939 the sample point is specified as 87.5% of a bit, and a bus speed of 250kbit/sec). Because there is synchronization jitter and clock drift, there will typically be a window in which controllers will sample the transceiver's RX pin. The CAN-HG augmentation works by adding extra bits in between the sample point windows (shown as dotted line in Figure 4). This extra data is invisible to CAN controllers, but it can be decoded by CAN-HG hardware.

Each ECU to be protected contains a *bus guardian* device as illustrated in Figure 5.

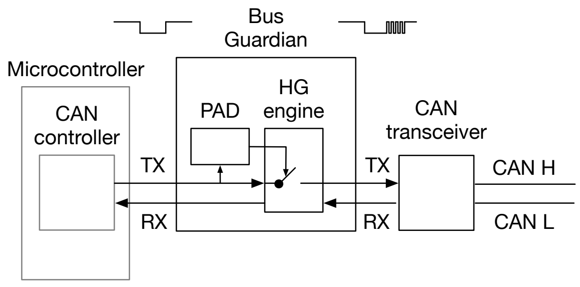

Figure 5: CAN-HG Bus Guardian

The CAN-HG bus guardian augments the outgoing CAN signal on the TX pin from the microcontroller by adding in the *source address* of the device (this is pre-programmed into the Bus Guardian chip and it cannot be changed by software) in out-of-band data and sending the resulting signal on to the CAN transceiver. This 'tag' indicates where the CAN frame physically came from.

The Bus Guardian also contains a *protocol attack detector* (PAD) that detects deviations from the CAN protocol (such as the Janus attack signals). If a protocol attack is detected by the PAD then it disconnects the microcontroller from the CAN bus (by no longer passing the TX signal through).

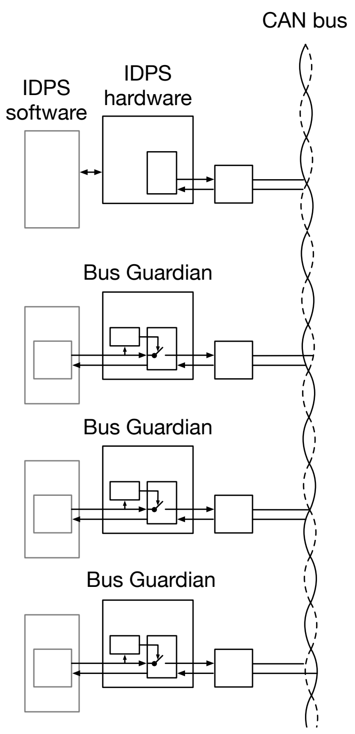

Figure 6: Centralized CAN-HG IDPS

A centralized intrusion detection and *prevention* system (IDPS) detects a spoof: for any given CAN ID there is a corresponding CAN-HG address for where it should come from and if there is a mismatch then the frame can be destroyed using the CAN error mechanism (as discussed earlier). But because the IDPS is a combination of software on a host microcontroller and hardware, the definition of what is a spoof can change with the operation of a system. For example, if there is no diagnostics tester connected then diagnostic frames can be defined as illegal and destroyed. If a tester is connected and the vehicle is stationary, then the frames could be made legal.

Being able to destroy illegal frames is only one part of the CIA triad: ensuring availability of the CAN bus by prevent a device from disrupting it is also essential.

The IDPS can detect crude denial-of-service attacks (like a flood attack) but also subtle timing attacks on specific frames by observing the timestamps of frames on the bus. The IDPS can directly command Bus Guardian chips to disconnect their ECU host from the CAN bus and so a denial-of-service attack detected by the IDPS can be shut down.

This centralized IDPS approach allows sophisticated security policies to be implemented in software and adapted with experience. For example, some failures might not be due to actual attacks but software failures, so a policy that recognized the possibility of failures and did not immediately treat a failure as an attack could be developed (under the Fleming principle "Once is happenstance. Twice is coincidence. Three times is enemy action.").

7. CONCLUSIONS

A taxonomy of attacks on CAN bus has been given and various defenses against these attacks described. The new CAN-HG augmentation hardware offers protection mechanisms against spoofing and denial-of-service attacks and allows for sophisticated security policies to be implemented in software using these mechanisms [8].

The CryptoCAN and CAN-HG technologies are currently being evaluated by the United States Army Combat Capabilities Development Command (DEVCOM) Ground Vehicle Systems Center (GVCS) in the cooperative research and development project "Cyber Security for Military Ground Vehicles Architectures".

8. REFERENCES

[1] S. Checkoway et al, "Comprehensive experimental analyses of automotive attack surfaces." Proceedings of the 20th USENIX Conference on Security, SEC'11, 2011

[2] I. Rouf et al, "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study", USENIX Security 2010, August 2010

[3] K.-T Cho and K. G. Shin, "Error handling of in-vehicle networks makes

them vulnerable", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016

[4] K. Tindell, "Demonstration of attacks on the CAN protocol," Automotive Security Research Group seminar 23 September 2021

[5] G. Litichever and G. Bandel, "Securing SAE J1939 Heavy-Duty Vehicles In-Vehicle Networks," Automotive Security Research Group seminar 14 January 2021

[6] K. Tindell, "CAN priority inversion," https://kentindell.github.io/2020/06/29/can-priority-inversion/ June 2020

[7] Herstellerinitiative Software (HIS) Security Working Group. SHE – Secure Hardware Extension Version 1.1, October 2009

[8] K. Tindell, "Defending CAN", four-part video series, November 2021