

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Reversing a Brake ECU Firmware Update

Ben Gardiner, *National Motor Freight Traffic Association Inc.*

https://www.usenix.org/conference/vehiclesec26/presentation/gardiner

This paper is included in the Proceedings of the
4th USENIX Symposium on Vehicle Security and Privacy.

August 10–11, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-56-4

Open access to the Proceedings of the
4th USENIX Symposium on Vehicle Security and Privacy
is sponsored by USENIX.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Reversing a Brake ECU Firmware Update

Ben Gardiner
National Motor Freight Traffic Association Inc.

Abstract

This paper presents a reverse engineering analysis of a firmware update for the Bendix EC80 Brake ECU, a critical component in heavy-duty vehicles. Unlike prior research focusing on CAN or UDS, this study isolates vulnerabilities in the legacy J2497 Power Line Carrier (PLC4TRUCKS) interface. We analyze the "ID9363" updater executable to extract firmware and perform differential binary analysis of the S12X architecture firmware images. Our research identifies specific changes in the recall that address undocumented vulnerabilities in J1587 PID processing and interrupt handling on J2497.

We demonstrate that the patched functionality contained critical flaws, including a write-where primitive and denial-of-service (DoS) conditions. These vulnerabilities were validated on a test bench and in a moving truck, where exploitation caused the loss of speedometer, dynamic steering, and automatic shifting. The findings confirm the safety recall remediation was also a security patch, mitigating risks in legacy protocol implementations. This work motivates the necessity of scrutinizing legacy interfaces in general in modern heavy vehicle safety systems, and the J2497 interface in particular.

1 Introduction

Introduced circa 2001, the J2497 [1] (aka 'PLC4TRUCKS') powerline databus remains the only industry-standard means to satisfy the trailer ABS warning light requirement of FMVSS 121 [2], S5.1.6.2(b). As such, J2497 has been present in all towing application Class 8 vehicles in North America from 2001 to the present.

Previous research on J2497 ECU security [3] [4] [5] [6] revealed that J2497 is wirelessly reachable and trailer brake controller ECUs implement diagnostics functions inside J1587 Data Link Escapes beyond FMVSS 121 requirements. However, current threat models assume tractor brake controllers only process necessary LAMP messages on J2497.

The 2024 Bendix EC80 safety recall [7] revealed safety impacts to tractor ECUs from J2497 reception, suggesting the EC80 processed J2497 traffic beyond necessary LAMP messages. The safety recall was remediated via a firmware update of the EC80 using the 'ID9363' updater executable.

We performed binary differential analysis on the S12X firmware (FW) pre- and post-recall in 3 types of the EC80 (one for each OEM affected by the recall). Results show extensive processing of J1587 messages received by the tractor brake controller over J2497 in the removed code. This code contained vulnerabilities including hardcoded secrets protecting traction control configuration, buffer overflows, and memory corruption leading to denial of service (of the vehicle) and remote code execution. Finally, we identified the removal of PID handlers discoverable through simple fuzzing. This suggests the safety recall also functions as a security patch, preventing attackers with wireless, 'adjacent' [6], access to J2497 from exploiting these vulnerabilities. This paper focuses exclusively on changes that were 'fixed' by this patch; no '0day' vulnerabilities are revealed as all have been patched.

Contributions. In summary, we make the following contributions:

- **FW Update Process**: We reverse engineered the operations performed by ID9363 from several captures of the UDS [8] traffic (Section 3)
- **EC80 Firmware Partitions**: The flash layout/partitioning and fixed addresses of the EC80 are revealed within the constraints of the S12X architecture (Section 4)
- **S12X Reverse Engineering**: We present novel methods for reversing the S12X banked memory architecture. We detail memory map reconstruction, symbolic analysis, and opportunistic function identification, overcoming significant tooling gaps (Sections 5, 6 and 7)
- **The Security Patch**: The functionality removed by the

patch is confirmed on all three types of EC80 units acquired and the vulnerabilities present in the removed functionality are itemized (Section 8)

- **Exploitability of Removed Functionality**: We evaluate the exploitability of each of the patched vulnerabilities from a mix of: bench and in-motion vehicle testing (Sections 9 and 10).

2 Background

In late 2024, three Class 8 vehicle OEMs in North America issued a safety recall [9] [10] [11]; these were the three OEMs that integrate the Bendix EC80 brake controller. Bendix identified potential memory corruption causing the ECU to go offline and worked with OEMs to deploy a firmware update for affected trucks – those with Electronic Stability Program (ESP) or Automatic Traction Control (ATC) features – estimated at 450,000 units at the time of this writing [12]. The timeline of this recall is summarized below in Figure 1 (see also [13] [14] [15]). See [16] for further details on the recall. The recall was later expanded in October 2025 to cover some units which were sold as aftermarket equipment [17] [18] [1].

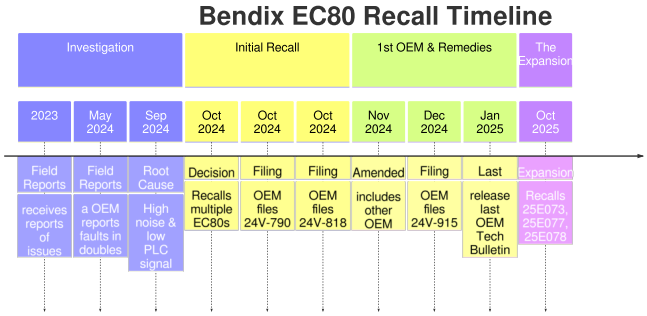


Figure 1: Bendix Recall Timeline.

The firmware update was distributed to fleets via a Windows executable, 'ID9363' [19] [20] [21]. We examine the ID9363 update process and the resulting firmware changes in three EC80 ECUs (one per OEM).

The firmware update, or 'patch', purportedly prevents J2497 noise from triggering memory corruption. However, due to CVE-2022-26131 [6], it is possible to wirelessly inject signals onto J2497. It is also possible to inject from compromised connected devices such as the increasingly common trailer telematics devices. We examine if these memory corruptions could be triggered by malicious actors using wireless (or wired) injection on J2497.

J1587 [22] serves as (roughly) the application layer for J2497, while J1708 [23] defines the original physical and data link layers (i.e. "framing"). The J1708 physical layer is replaced with a Power Line Carrier (PLC) modulated over the 12V auxiliary power line in J2497, but it inherits the J1708 data link and J1587 application layers. A frame can only be a maximum of 21 bytes long and must end with a one-byte checksum (the two's complement of the sum of the preceding bytes). In software, the frames are received byte-by-byte over a standard UART peripheral via a J1708 to J2497 converter chip (the Intellon SSCP485 [24]).

For the J1587 application layer, the first byte is defined by J1708 and is the Message Identification (MID), which identifies the sender or the message type. MIDs 0x0A (LAMP ON), 0x0B (LAMP OFF), and 0x57 (Active Trailer ABS Event) are required by the J2497 standard. For other J1587 messages the bytes following the MID are Parameter Identification (PID), which specify the type of attached data (e.g., VIN, wheel speed, diagnostics) followed by parameters (payload). Some PIDs have a fixed data length (like a 1-byte speed value), while others are variable length. PIDs data enables transmission of vehicle signals, diagnostic commands, and proprietary data.

The Bendix EC80 is a heavy-duty vehicle Electronic Control Unit (ECU) responsible for Anti-lock Braking (ABS), ATC, and ESP functions. It controls pneumatic circuits through external pressure modulation valves connected to PCB-mounted FET drivers. During initialization, absent critical Diagnostic Trouble Codes (DTCs), modulators undergo individual testing, producing a 'roll call' of small air chuffs [2].

Table 1 provides an overview of the EC80 units acquired for this study. The complexity of the EC80 firmware varies with the integrated features. The middle target, `2ec80`, represents a medium complexity unit and was selected as the primary target for reverse engineering in this paper. All findings have been validated across all the target firmwares.

The EC80 units affected were released in April 2020 [20]. Given its role in preventing rollovers and maintaining stability, brake controllers are expected to have a high Automotive Safety Integrity Level (ASIL), likely ASIL C or D, to meet OEM safety targets. While specific ASIL targets for EC80 vehicles are unpublished, Hazard Analysis and Risk Assessment (HARA) of brake controllers [25] identified ASIL D scenarios. While focused on autonomous systems, this analysis notes the architecture applies to non-autonomous vehicles, stating the brake system is critical regardless of autonomy. Furthermore, the literature for a competing product, the mBSP, explicitly states compliance with ISO 26262 [26] up to ASIL D [27].

A Threat and Risk Assessment (TARA) of Class 8 vehicles [28] ranked brake controllers third, after telematics and gateway devices, for cybersecurity priority. The EC80 con-

[1] all recalls for this same issue known at this time: 24V780000, 24V818000, 24V915000, 25E073000, 25E077000, 25E078000.

[2] In a bench environment the same successful power-on results in a clicking of a failsafe 'diagonal' relay.

Target Name	1ec80	2ec80	3ec80
FW version, before	Z228999	Z266494	Z286098
FW version, after	Z300822	Z302578	Z302579
Sticker Name	EC80ESP+	EC80ESP	EC80ESP
Non-sticker Feature	J1708	-	-
Sticker Features	6S/6M	6S/6M	4S/4M
	PLC	PLC	PLC
	2nd CAN	2nd CAN	-
	-	CAN Gateway	-
	Integrated TPMS	-	-
HW Date Code	20220723	20230507	20221019
SW Date Code	3H0922G	3E2323G	2F1423G

Table 1: Information on the three EC80s acquired and tested. While specific part numbers and OEM integrators are omitted, the firmware versions presented are consistent with recall documentation and can be retrieved from the target ECU via a Read Data By Identifier request (0xF194). The order here matches Figure 2 left to right.

nects to multiple interfaces, including "Untrustworthy Network Domains" like J2497 [29]. Because it is not explicitly intended to transport, translate, or filter traffic, it is classified as an "unintended gateway" device [29] [3]. Consequently, it must satisfy cybersecurity requirements NGW-S-001 (Security Assurance), NGW-S-002 through -005 (Won't Transport, Translate, Filter, etc.) [29].

The EC80 uses two S12X microcontrollers (as can be seen in the PCBs pictured in Figure 2). Both are specifically the NXP MC9S12XEQ512 [30], a 16-bit microcontroller from the S12X family; however, the 'left' MCU is in a smaller footprint 80-pin QFP package compared to the larger 'right' MCU in a 144-pin LQFP package.

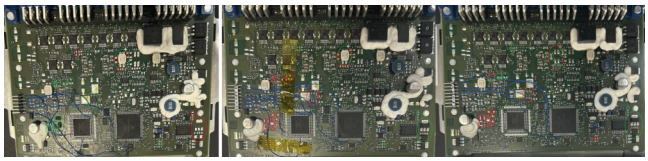


Figure 2: The PCBs of the three EC80s acquired and tested. A 0.1" header is installed for access to the BDM programming pins on the right S12X MCU. The added/removed components are highlighted in green/red, respectively.

The S12X [31] features an XGATE peripheral coprocessor to offload high-speed data transfer and logic processing from the main CPU12X core. The S12X employs a paging architecture using PPAGE, EPAGE, and RPAGE registers to map 16-bit logical windows into a 23-bit global address space. The device includes 512 KiB of PFLASH (Program Flash) residing at global addresses 0x780000 - 0x7FFFFF and 32 KiB of DFLASH (Data Flash) located at 0x100000 - 0x107FFF. Additionally, it features a 4 KiB Emulated EEPROM (EEE) buffer RAM mapped to global addresses 0x13F000 - 0x13FFFF, which allows the firmware to treat the DFLASH as random-access non-volatile memory.

3 ID9363 Update Process

We observed several EC80 firmware updates. The new firmware is readable in cleartext from CAN databus captures; however, analyzing the update requires the pre-update firmware contents. While some ECUs offer a UDS 'upload' service, we confirmed the EC80 does not. From the traffic we reconstructed the seed-key exchange routine in closed-form for all DSC,SA sessions (see Appendix A), confirming no firmware upload service exists in the observed sessions (DSC=2,SA=5) (DSC=3,SA=1) (DSC=3,SA=7) (DSC=3,SA=3) [4].

We used the BDM interface to dump PFLASH, DFLASH, and EEE (emulated in DFLASH) from both MCUs to obtain the pre-update firmware. We found that both the XPROG tool [32] and PROGS12Z [33] worked well. XPROG creates flat binary files of PFLASH, DFLASH and EEE which correspond to their global address extents. PROGS12Z creates .s19 files [34] which contain the global address extents of PFLASH, DFLASH, and EEE.

We found that although the S12X used in the EC80 supports read-out protection, it was not enabled. We confirmed that the data transferred over CAN matches the PFLASH contents after update. Only the 'right' MCU (larger package) receives code changes; the 'left' MCU's PFLASH remained unchanged. It did, however, receive some DFLASH / EEE changes during the update process. We focus solely on the 'right' MCU's code changes.

A visualization of the byte-by-byte difference of PFLASH before and after update of a 2ec80 is presented in Figure 3. Byte differences are marked by pink fill, red outline boxes, and are numerous (this is clearly not a micropatch [35]). The "Drive Block"s 1 and 2 are labeled. The PPAGE numbers corresponding to the PFLASH offset are labeled at the bottom for reference. The same style of visualization will be reused throughout the paper to illustrate successive analysis steps. In this figure function prolog estimates are plotted as green circles (in subsequent figures the functions resulting from firmware analysis will replace these).

Despite extensive byte differences, the high-level source code changes may be concise. Small machine code changes and linking order can cause massive byte differences. Byte-level analysis cannot reveal specific code changes or isolate areas

[3] Although the 'Z' version which implements a 'CAN Gateway' (see Table 1) could be considered a '(intentional) gateway device'

[4] memory leaks, other unintended upload functionality in the bootloader or application are out of the scope of this paper

of interest, as changes span 'Drive Blocks' 1 and 2. Understanding the changes requires disassembling and comparing the pre- and post-update images.

4 EC80 Bootloaders and Application

Global PFLASH byte differences (Figure 3) and ID9363 updater traffic analysis (Table F.1) indicate a J1939 [36] CAN UDS [8] bootloader in the skipped region. We label this the 'late' bootloader as it does not contain the reset vector. Code between the reset vector and this late bootloader is labeled the 'early' bootloader. Its code likely resides in some subset of PPAGE 0xFF (aka "Drive Block 2 of 2") which has a fixed mapping in the S12X memory space.

The bootloaders presumably hand-off to the 'application'. Its code is very likely in the "Drive Block" 1 of 2 region, although it could have some code in the other "Drive Block" as well to make use of the fixed mapping of PPAGE 0xFF.

5 Limitations of Static Analysis

The S12X primarily uses a 16-bit memory space with banked windows (PPAGE, DPAGE, RPAGE) to access global memory [30], rather than direct global addressing (Figure 5) [5]. Disassemblers must model this mapping to correctly resolve page-dependent references to the global address space. Fixed mappings, like PPAGE 0xFF at 0xC000-0xFFFF, are straightforward. The reset vector is in this address so starting analysis with a disassembler is likewise straightforward. In all EC80 firmwares, execution quickly encounters a call instruction with a 3-byte target: a 16-bit offset and a PPAGE. At this point the disassembler needs to have the code in the correct location in its memory model.

[5] although specific, slower execution, instructions exist for direct global address access.

There are not many examples of S12X architecture reversing available. Of course the seminal work of [37] did include S12X reversing of a power assist steering module, and it did include uncovering the use of hardcoded passwords there for UDS [8] seed-key exchange (the security access $27 service). But the 'how' of creating dumps of PFLASH, DFLASH, RAM and assembling it for a reverse engineering tool (e.g. IDA Pro as used by [37]) are not available. The recent analysis by Pulse Security of a motorcycle ECU [38] demonstrates a way to assemble an S12X target binary image for analysis in Ghidra [39] before switching to hunting for numeric tables in Flash. Ghidra does not support the XGATE coprocessor, whereas IDA Pro [40] does; therefore IDA Pro was selected for this analysis. The IDA Pro support team was kind enough to provide a code snippet of how PPAGE is modeled in the IDA linear address space model. It corresponds to how all of Ghidra, the HSW12 open source assembler [41], and the original S12X debugger, HiWave [42] treat it: use PPAGE as the most significant byte in a three-byte address (e.g. the gray boxes in Figure 6). The same is done for RPAGE and EPAGE [6]. Because RPAGE and PPAGE windows are discontiguous in the 16-bit local space this does not cause collisions. This addressing scheme is very useful and will be used in the remainder of the paper (i.e. any three byte addresses other than global addresses imply the first byte is the page and the remaining bytes are the offset within it). IDA also expects to be able to address the global memory space in the exact same address values in its linear memory map (the peach boxes in Figure 6); however, this is only used for instructions that do global memory space access.

We used IDA Pro Python scripts to map PFLASH to global addresses and copy it to high-byte and fixed PPAGE locations in the 16-bit local space (Figure 6, Appendix I.0.1).

[6] This causes IDA linear space collisions between EPAGE 0x10/0x13 and DFLASH global addresses. No workaround exists; users must be cautious if firmware uses both.

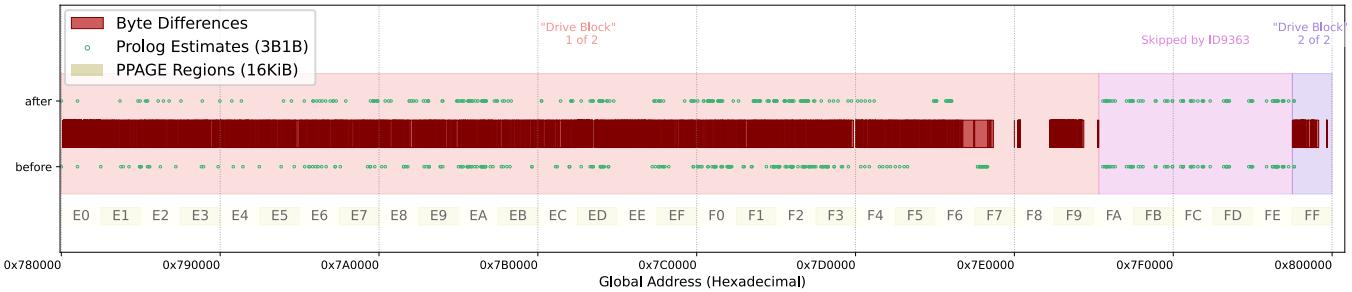


Figure 3: Comparison of the PFLASH of 2ec80 before and after ID9363 update. The unchanged extents illustrate that both the interrupt vector table at the end of PFLASH (0x7E8800) and the region 0x7E8800 - 0x7FC000 were skipped by the firmware update. Many other locations where the PFLASH contents are unchanged were found to have value 0x3F '?', the Software Interrupt ('SWI') instruction, which we believe is used as a padding value. The number of 0x3F bytes increased, suggesting function deletion by the update. Function prolog estimate locations across the update suggest shifting to lower addresses which (also) suggests function deletion by the update (examined in Section 8).

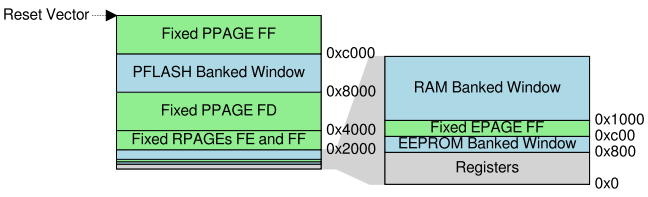


Figure 5: Local 16-bit memory map of S12X by linkerscope. Green for fixed page areas and blue for paged windows.

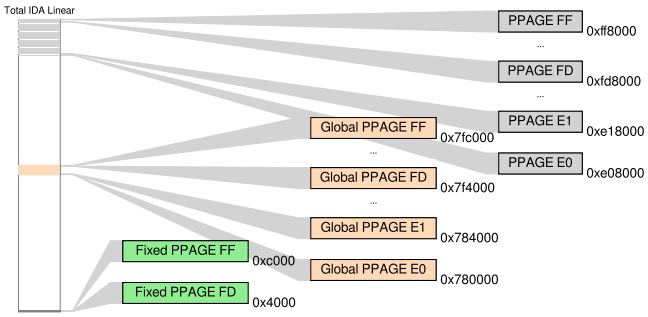


Figure 6: Illustration of the copies of PFLASH placed in the IDA Linear Address space memory map for some success in static analysis.

We then created an entrypoint for every interrupt vector in 0xFF80 - 0xFFFF, including reset, CAN, Programmable Interrupt Timer (PIT), Serial Communications Interface (SCI), and all other peripherals with an interrupt. IDA Pro analysis results are shown in Figure 4, with function extents replacing the prolog estimates used previously in Figure 3.

Analysis proceeds past three-byte calls, confirming correct PPAGE installation. While analysis reaches the late bootloader, it fails to reach application code. Starting from the reset vector often yields incomplete coverage due to unresolved data-dependent jump or call targets in many firmwares and the EC80 is no exception.

We focus on J2497 processing changes, so understanding the entire EC80 bootloader is unnecessary. The EC80 receives J2497 data via an Intellon SSCP485 [24] chip connected to the S12X SCI2 peripheral. However, the SCI2 interrupt vector of the default table points to a dummy function, and no discovered functions reference SCI2 data registers.

6 Limitations of Dynamic Analysis

Since static analysis failed to reveal SCI2 access, we used the unlocked BDM for dynamic analysis.

The debugger software we had access to is HiWave [42]. Breakpoints on SCI2 register reads were never triggered. We found that breakpoints cease functioning after the late bootloader entrypoint (0xFBADB1). The cause is unknown; possibilities include `BGND` instruction execution or resets triggered by the secondary ('left') processor. Attempts to reset both MCUs in tandem failed. Despite evidence of debug-aware firmware (e.g. watchdog feeding after `STOP`), we abandoned further investigation to focus on the firmware patch.

BDM allows reading S12X address spaces without interrupting execution (mostly [7]). We used HiWave's [42] scripting interface (Appendix I.0.10) to create hexdumps and convert

[7] BDM accesses global address space bypassing the MPU. It is non-intrusive only during free bus cycles, otherwise stealing cycles or stalling the CPU.

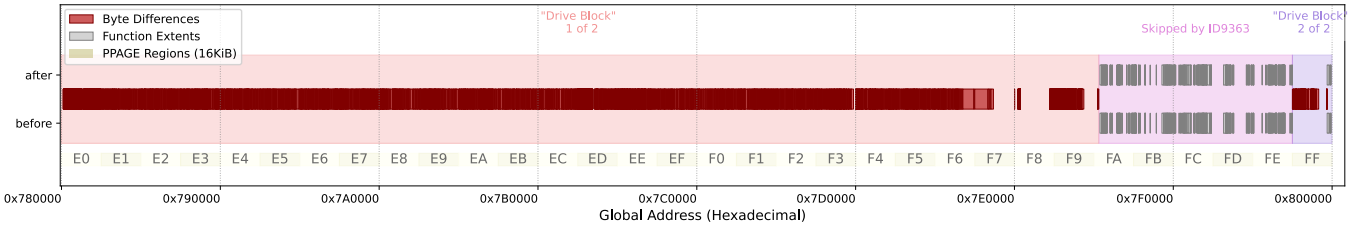


Figure 4: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in the default interrupt vector table. The lack of functions in all but the 'early' and 'late bootloader' regions illustrates that auto-analysis is not able to follow the exeution flow into the 'application' region.

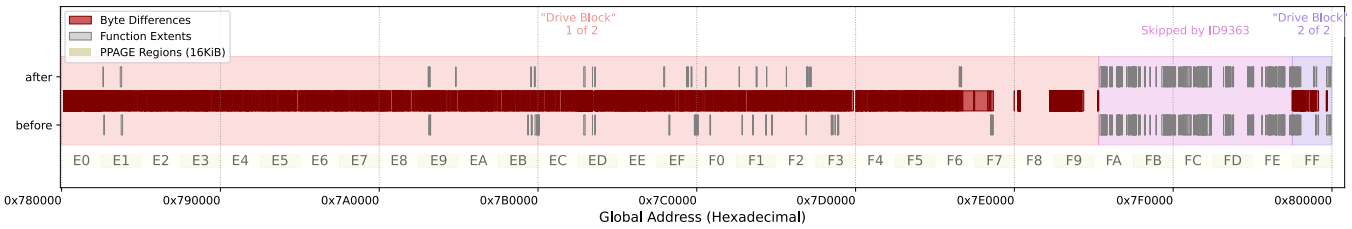


Figure 7: Visualization of what the IDA Pro auto-analysis achieved after creating entrypoints for each of the handlers in an interrupt vector table given by IVBR=0xF7. Compared with Figure 4, this figure shows several functions in the 'application' region; however, they represent a small fraction of the firmware's total application code, the upper bound of which is given by the byte differences. Similar to the prologs in Figure 3, it also shows correlated, left-shifted function extents which (again) suggest code deletion by the update; however, the low overall coverage limits our confidence.

them to .s19 files. This allowed us to read 'live' RAM and register contents without breakpoints and load them into IDA.

Polling registers revealed that the application uses an IVBR of 0xF7, unlike the initial bootloader's default 0xFF. The S12X has the capability to switch the location of its interrupt vector table [8]; although static analysis at the stage shown in Figure 4 missed the IVBR assignment, we had confirmed the application uses an interrupt vector table at 0xF710-0xF7F9.

We updated the IDA Pro database with RAM dumps and entrypoints for the new vectors (Figure 7). Analysis still covers much of the late bootloader, as the application frequently calls back into it (e.g., for programming mode).

We can now access application interrupt handlers. Target `2ec80` supports XGATE [9], PIT, CAN0, CAN4, Enhanced Capture Timer (ECT), SPI1, and SCI2. We focus solely on SCI2. The SCI2 ISR, `sub_C12B`, calls functions where IDA Pro analysis fails due to lack of 'register tracking' for S12X. For now, we manually resolved data-dependent `call` instructions (e.g., `call [-$2662,y]`) that IDA failed to handle. We also manually resolved jump tables, which IDA Pro does not support on S12X.

Then IDA Pro analysis revealed the application's J2497 message reception operation [10]. The details of the receiver architecture are captured in Appendix B; but the dataflow of received packets from the SCI2 peripheral to consumers is simpler: frames are built by appending to a receive buffer at 0x3BF5 and frame reception is signaled with the frame counter / semaphore at 0x3BF4. Figure G.1 illustrates this dataflow. This IDA analysis identified only the 'publish' side. No consumers of the frame counter or receive buffer were found. Firmware typically splits interrupt handling into a lightweight top-half and a main-thread bottom-half to minimize interrupt context cycles.

7 Binary Diffing

Interrupt context execution was explored, but the main execution thread remained invisible. Analysis failed to trace the late bootloader hand-off to the application [11]. Searching for function prolog byte sequences yielded excessive false positives. We observed function pointer arrays in 'Drive Blocks' 1 and 2. Indirect call instructions (e.g., `call [-$2662,y]`) read a 3-byte address (PPAGE offset + PPAGE value) from a resolved memory location. The compiler/linker used here always emits a trailing zero byte. Scanning for these patterns, filtering for valid PPAGE values and offsets (0x8000-0xC000), significantly reduced false positives. We further filtered out isolated function pointers, as they typically occur in arrays. See Appendix I.0.4 for the script used. Then analysis discovered the main-thread bottom-half function for J2497 message processing at sub_EEB989 (see Figure G.2).

The main thread processing in `sub_EEB989`, handles J2497 LAMP ON (0x0A), LAMP OFF (0x0B), and Active Trailer ABS Event (0x57) commands, ignoring payloads. It consumes contiguous 0x0A, 0x0B, or 0x57 messages. Messages with other MIDs are passed to `sub_F096F9`. J1587 messages can contain one or more PID payloads. This function splits J1587 messages into PID payloads and passes the payloads to `sub_F2A3E1`, which dispatches each to a handler from PFLASH tables starting at the 0xDC3D 'context structure' (see Appendix C for details). This data flow processing is illustrated in Figure G.2.

To analyze six firmware images (three pairs) with BinDiff [43], we automated IDB creation (see I.0.1), segment setup,

[8] Except for Power-on, Clock Monitor, and COP Watchdog reset vectors.

[9] We analyzed XGATE code by mapping PFLASH and RAM in IDA. XGATE handled only ECT interrupts (channels 0-7) and was irrelevant to J2497 reception, so analysis ceased. Ghidra would suffice here, but we had at this point invested in IDA-based automation.

[10] One firmware change is observable now but will be detailed in Section 8.

[11] Presumably due to at least a data-dependent call target but possibly due to this and other reasons. The result was good enough to identify the functions that consume the receive buffer 0x3BF5 and Frame Counter 0x3BF4: `sub_EEB98` and `sub_F096F9`; these are functions that check for LAMP messages and then process non-LAMP J1587 messages, respectively.

interrupt vector definition, and function search. We scripted the identification and marking of PID tables and handlers (Appendix I.0.5). We excluded the late bootloader to focus BinDiff on the application firmware.

Lack of 'register tracking' hindered automatic analysis, requiring manual resolution of indirect call targets. We implemented rudimentary concolic analysis to parse disassembly and symbolically track execution, substituting concrete values from PFLASH, DFLASH, or RAM when needed and where available. See Appendix I.0.2 for the script used. We also implemented basic jump table detection and creation by parsing disassembly and creating manual code xrefs (Appendix I.0.3) [12].

To ensure BinDiff fidelity, we aimed for zero disassembly errors. We found numerous disassembler errors in PPAGE 0xFD functions. Hiveplots [44] of IDA Pro xrefs revealed invalid data writes to PPAGE 0xFD from functions in other PPAGEs. These writes are invalid as PFLASH requires sector erasure. Runtime polling of the MMCCTL1 register confirmed that the application maps 0x4000 to fixed RAM, not PPAGE 0xFD. We adjusted the IDA database segment setup, as shown in Figure 8.

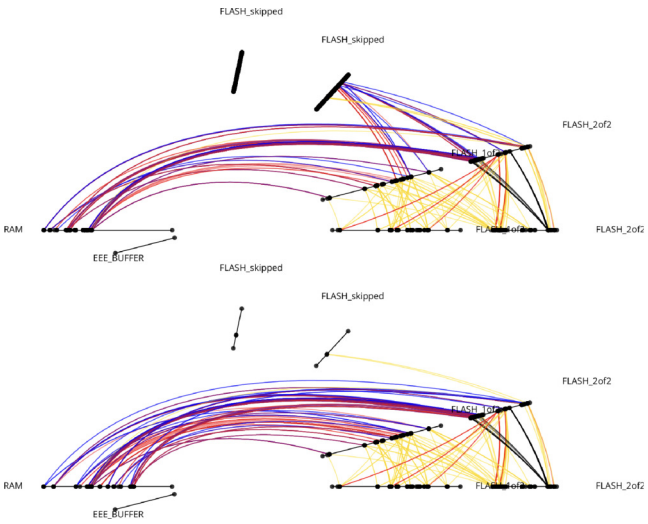


Figure 8: Hiveplot panel comparing IDA Pro analysis achieved before (above) and after (below) a change of the 0x4000 mapping from Flash to RAM. The cross-references (xrefs) are drawn as edges directed counter-clockwise (CCW); each region with executable code is doubled to show intra-region xrefs; the "Drive blocks" 1 and 2 are shown co-linearly and the skipped region is shown lifted above them; yellow edges are code xrefs, red edges are data reads, blue are data writes and black are interrupt vector references.

These steps achieved function coverage of 80% (1ec80), 60% (2ec80), and 75% (3ec80) of byte differences in PPAGEs E0-F9. We then automatically performed BinDiff on all three target pairs. Bindiff was able to support the S12X architecture and give reasonable insight; however, its shortcomings when applied to this firmware caused spurious matches e.g. it matched the SCI2 handler in a before image with the SCI3 handler in an after image.

We ultimately created a QBinDiff [45] analysis, the results of which are in Figure 9. The key to successful QBinDiff analysis was to 'anchor' (force a match) all the interrupt handlers. This yielded improvements over the Bindiff results. Less than 1% of discovered functions were low-confidence matches. Over 98% of functions remained unchanged, while ~110 - ~140 were modified or deleted (Table 2).

	1ec80	2ec80	3ec80
New	0	0	0
Deleted	104	127	123
Modified	11	12	13
Unchanged	1109	1100	1031
Low Confidence	12	13	14

Table 2: QBinDiff results summary of all three targets.

Three-way BinDiff analysis showed that matched functions dropped from 968 (pre-update) to 878 (post-update) across all 'Z' versions, corroborating function deletion. The patch makes very similar changes in all three cases: of the deleted functions, 87 were common to all three updates. The next section, 8, analyzes these changes. Fewer functions were deleted in the 1ec80 update because it retained J1587 Transport Protocol (TP) management functions which were deleted in the others; presumably, because they are used due to the J1708 interface which 1ec80 has and the other targets do not.

Term	Definition	BinDiff Terminology
Function Extents (matched, unchanged)	Functions present in both binaries with identical logic and structure.	Matched, Similarity = 1.0
Function Extents (matched, changed)	Functions present in both binaries but with some modifications to instructions or structure.	Matched, Similarity < 1.0, Confidence >= 0.95
Function Extents (unmatched)	Functions present in one binary but not the other (new or deleted).	Unmatched (Primary/Secondary)
< 95% confidence Function Extents	Functions that might be matched but with low confidence; ignored for high-assurance analysis.	Matched, Confidence < 0.95

Table 3: Table defining the categories of functions used in Figure 9 and the analysis presented in Sections 8 and 9.

[12] because the actual jump table operation "probably can't be scripted" according to IDA Pro support.

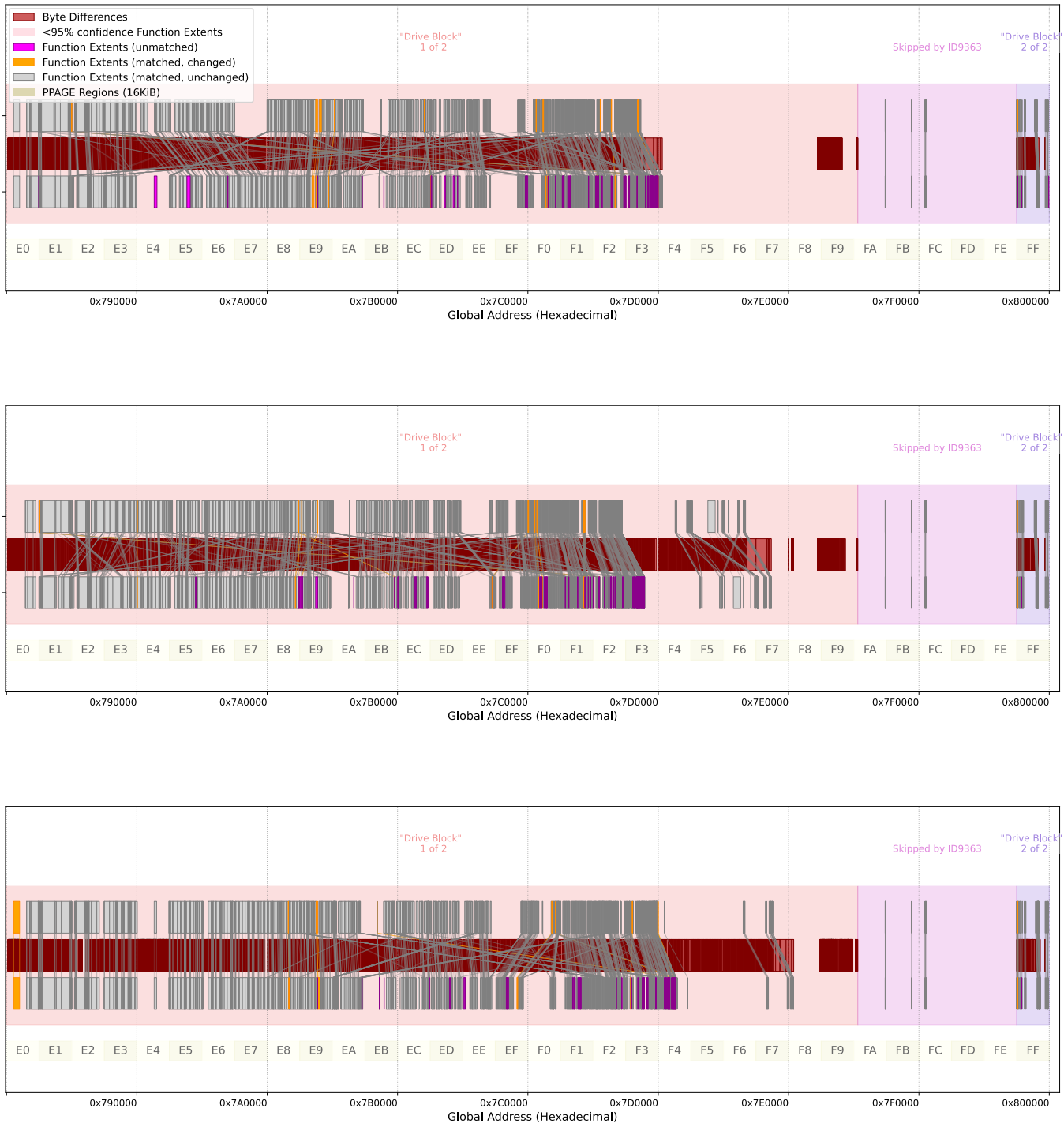


Figure 9: A visualization of the QBinDiff analyses of all three of the firmware updates (when limited to the APPLICATION parts of the PFLASH), overlaid on the byte differences of each. The varying complexity of the firmwares mentioned in the introduction is clearly illustrated here (compare to Table 1); target 3ec80 (top) shows the least complexity while 1ec80 (bottom) shows the most. Links between matched (by QBinDiff) functions are plotted across pre- and post-update. Despite noise from small/null function matches, the links show clear function shifting to lower addresses and numerous unmatched functions in the pre-update images (colored magenta), suggesting multiple function deletions. More conclusively: there are no (known) spurious matches and zero 'new' functions post-update: further suggesting multiple function deletions in the update.

8 Changes Made in Patch

Manual BinDiff analysis, validated by call tree analysis script (Appendix I.0.6), categorized all the deleted functions in all three firmwares as one of:

1. All J1587 PID processing present in the pre-update image (Appendix C), excluding LAMP and active ABS event processing.
2. SCI2 UART EDGE interrupt handling (Appendix B).
3. Secondary J1587 features, including the diagnostic code manager (`sub_E9858A`), J1587 Transport Protocol (TP) connection management (`sub_EF905B`), and TP timeouts (`sub_EC8116`). [13]

Modified functions primarily remove calls to these deleted functions:

1. In `sub_EEB989` (LAMP/ABS check), the call to `sub_F096F9` and subsequent PID processing is removed (Figure 10, EBB 0xEEBA20), consequently removing all linked PID processing tables and functions.
2. In `sub_C12B` (SCI2 ISR), the call to EDGE interrupt handling is removed (Figure 11), effectively eliminating the 'RXEDGIF Branch' (Figure G.3) and related setup/processing.

The bulk of the deletions were due to the removal of all PID processing on J2497 (see Table C.1 for the full list). The disassembly of all deleted functions (for target 2ec80) is provided in Appendix E.

9 Exploitability of Removed Functionality

We reviewed deleted functions (primarily PID handlers, Section 8) for vulnerabilities using static and dynamic analysis. Dynamic analysis was limited to fuzzing due to breakpoint limitations (Section 6). We identified several vulnerabilities in the deleted code.

The PID 0xC2 handler was exploitable for Denial-of-Service (DoS) and Remote Code Execution (RCE); both verified, see below. A DoS condition could be triggered by a J2497 message like `89C2FE`, where a large 'n' parameter (0xFE) caused a crash. For RCE, the handler processes a length parameter specifying the total size of data following the length parameter. The data is copied to a 0x30-byte stack buffer without length check. The return address is 0x31 bytes from this buffer, creating a classic buffer overflow [46] (see Listing of `sub_F1B623` in appendices) with one intervening byte: the count of a loop executed after the copy. While J2497 frame limits restrict direct attacker-controlled data to 18 bytes, the copy size is fully controllable.

[13] J1587 TP was not functional even before the update, confirmed with static analysis and testing.

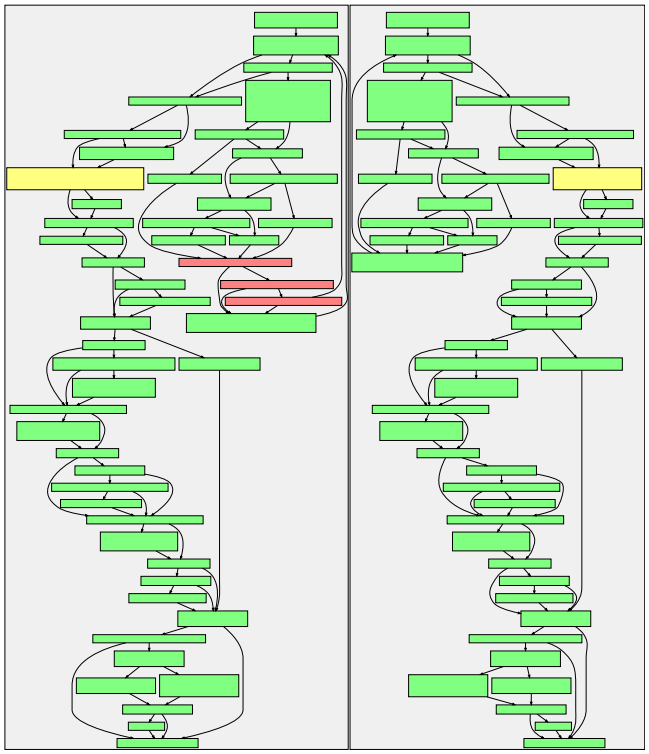


Figure 10: BinDiff of the LAMP etc check function before and after update. The modified EBB has a two-line change: remove a call to `sub_F096F` (Process PID payloads) and remove load of an argument to that call. The deleted EBBs were removed because with the call to PID processing removed there is no longer any need to handle LAMP and Active ABS MIDs at a higher priority.

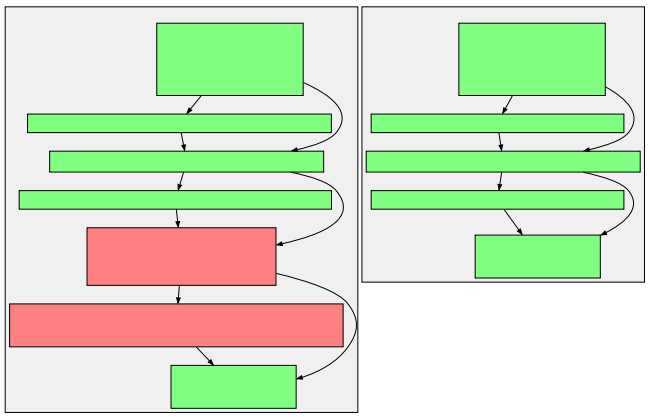


Figure 11: BinDiff of the SCI2 Handler before and after update. The deleted EBBs were responsible for servicing SCI2 EDGE interrupts.

Furthermore, the adjacent region is attacker-controlled, as J2497 frames are contiguous (with an injected frame-length byte) in a 0x54-byte FIFO buffer: an attacker can control the entire 0x53 byte copy by first sending a complete 0x52 byte buffer as a message (results in additional injected length byte and added checksum byte) starting with `89c280` which fills the buffer at 0x3BF5 (but does not get passed to higher application layers because it is longer than 21 bytes) then sending `89c280` by itself. The second message will get written to the same location as the previous `89c280` bytes in the receive buffer and will be passed to higher layers (with the remaining buffer in-tact). The wrinkle here is that any reception of other messages on the J2497 databus will interfere with the layout in the buffer: messages received before will move the start position of the first and/or second messages, reducing available buffer space and messages received in-between will move the start of the second message, removing the adjacency of the attacker-controlled buffer. A typical J2497 databus does contain a stream of LAMP messages and e.g. 0xC2 and other lower priority messages. There is a means to inhibit reception: during the development of the keyhole mitigation [47] the ideal standing sinusoid frequency was discovered to inhibit message reception. An attacker can transmit this sinusoid interference (to empty the buffer) and then transmit their exploit in a blanking of the sinusoid. They can also abuse the Intellon SSCP485 receiver feature where the pre-amble is superfluous to send the two body-only messages in a sequence quicker than 'normal' J2497 databus transmitters can achieve.

The RCE was confirmed on the 2ec80 and 3ec80 targets in a bench environment. This confirms that although the S12X has an MPU which could inhibit execution from data buffer and/or stack areas, it was not configured to do so. The following listing shows a simple proof of concept with NOP placeholders. The first code block is copied to the stack (as described above); the return address is overwritten with the address of the first code block in the receive FIFO 0x3BF5. Execution flows into the receive buffer original copy of the code block and jumps over the return address and intervening byte there to use an additional area for more code in the buffer.

```
c2_cmd          = [0x89, 0xc2,
                   0x30     # buffer size
                   + 1      # 2nd loop limit
                   + 3]     # ret addr on stack
payload_in_buff = ( 0x3bf5  # handy static address
                  + 1  # 1 for the injected length
                  + len(c2_cmd) )  # skip the c2 cmd

code_1 = list(bytes.fromhex(
  "A7"        # 3BF9: NOP          ; needed for jmp
  "1410"    + # 3BFA: SEI          ; Disable ints
  "A7" * 43 + # 3BFF: NOP NOP...
  "2004"      # 3C22: BRA   04     ; to CONTINUE
              #                    ; PC (3C29) + 4
))
sec_loop_lim = [0x00]  # skip handler's 2nd loop
ret_address  = [0xf2,  # PPAGE for later return
                       # to f2a462 via jump
                (payload_in_buff>>8) & 0xFF,
                (payload_in_buff   ) & 0xFF]

code_2 = list(bytes.fromhex(
              # CONTINUE:
  "A7" * 23 + # 3C2D: NOP NOP...
              #       ; Reduce this for resilience
  "10EF"      # CLI           ; Enable Interrupts
  "06A462"    # JMP   $A462   ; Return control
))

tosend = bytearray(c2_cmd + code_1
         + sec_loop_lim + ret_address + code_2)
you_send(tosend)  # fills the entire 0x3bf5 buffer
you_send(tosend[0:20])  # small enough to trigger
                        # reception of 89c2 command
```

This PoC uses the receive FIFO buffer at 0x3BF5 as the location for hosting the executable code and only queues one message (the `[0:20]` slice above). If other bytes are received before the `SEI` is executed they will be written into the buffer after 0x3BF5+20. The interfering sinusoid mentioned above can be reasonably effective at accomplishing this. A more robust approach could use the stack only (if the SP address in the 0xC2 handler were known [14]) or an approach could send multiple messages and skip over the injected length bytes. Long-running payloads will require feeding the watchdog peripheral and there are many examples of this in the firmware.

The PoC leaves 66 bytes of usable payload; this is enough to transmit a valid J1939 [36] CAN frame. The basic form of this is 85 bytes long using MOVB for data move. This can be reduced to 62 bytes by using STD 2,X+ instead. And it can be golfed into a 43 byte payload by abusing the stack for PULX; however, using PULX is not practical on the targets which use the XGATE coprocessor during their runtime (such as 2ec80) because the stack is shared. The payload can also be reduced to 39 bytes by re-using the firmware's CAN-sending function `sub_E9919A`. This was confirmed using the HSW12 assembler [41]. Also possible is misconfiguring the CAN transceiver to a bad bps which will crash that CAN bus segment and this a much smaller payload. This will silence all ECUs on the bus – until bus recovery fixes the problem, if at all. More is possible. As is always the case with RCE: anything is possible [48]. The space limitation here is also only a minor problem since an attacker can find many RAM areas to write longer payloads 'in pieces' as well.

Returning to the vulnerabilities in the other deleted code, the PID 0xED [15] handler can also crash the ECU. Sending a rapid sequence of messages with a large length parameter triggered this vulnerability. Bench and closed-track testing confirmed that greater than 10 repeated `89EDFE` messages trig-

[14] it was not for us. Because there was no dynamic analysis possible this PoC was developed iteratively in a crash/no-crash loop

[15] which is Trailer VIN PID, not the previously allocated PID: Entry Assist Control #1.

gered the crash (128x was used to be sure). The PID 0xED handler (`sub_F2BA46`) causes this by copying the PID payload (length-specified) into a static buffer at 0x5BDC without bounds checking (see Listing for `sub_F2BA46`s). This buffer has a 26-byte safe limit before overwriting critical structures at 0x5BF6 and 0x5BF8. These locations are used by `sub_F38241` for a conditional `memcpy` (`sub_E642`), with the condition seemingly controllable via the PID 0xB4 handler. Like the 0xC2 handler, the attacker controls copy size and can control the adjacent bytes in the receive FIFO buffer.

Theoretically, sequential 0xED and 0xB4 PID payloads (potentially in one J2497 message) could establish a write primitive. The write primitive pointer and data are 26 and 28 bytes from the 0xED handler's buffer start. Successful exploitation requires careful J2497 receive buffer grooming to overcome the 18-byte control limit. This write primitive makes RCE feasible, given unused RAM regions and fixed-address function pointers in the firmware [16].

Deleted SCI2 EDGE interrupt code theoretically presented a write-where primitive. Exploitation involves: 1) Triggering an EDGE interrupt and timeout to partially reset the receive buffer (by `sub_F1B035`), setting a buffer full flag (see Table G.1) but leaving the write offset (0x3C4B) un-reset. 2) Consuming a subsequent message clears the buffer full flag without resetting the offset, potentially exceeding the 0x54 buffer limit. 3) `sub_EC803E` (append data) contained a write-before-check flaw. Data was written to the receive buffer (0x3BF5) at the current offset (0x3C4B) before boundary validation. This allowed writes beyond the limit. 4) Positioning the write offset at +0x54 and sending a 4-byte frame allowed setting arbitrary upper bound (0x3C49) and current offset (0x3C4B) values. Subsequent data would be written to the new offset address.

The PID 0xC7 handler contained a hardcoded password check. Receiving the matching password disables traction control, as per J1587 specification [22]. Bench testing confirmed that message `ACC703004B42` satisfies this password check [17].

Component	Vulnerability	Impact
PID 0xC2	Buffer Overflow	DoS (Verified), RCE (Verified)
PID 0xED	Unbounded Copy	DoS (Verified), RCE (Theoretical)
PID 0xC7	Hardcoded Credential	Auth Bypass (Verified)
SCI2 EDGE	OOB Write	DoS (Theoretical), RCE (Theoretical)

Table 4: Summary of vulnerabilities in removed functionality.

Exploits of these RCE vulnerabilities (verified and theoretical) will depend on the specific 'Z' firmware version memory layout. The ID9363 updater contains 13 'Z' versions (covering a total ~450,000 units [12]); however, exploits of the DoS vulnerabilities and hardcoded credential will not. Testing shows they worked in all three tested 'Z' versions and likely affect all versions.

Reverse engineering is not required to discover all these vulnerabilities. A simple fuzzing script discovered the PID 0xC2 and 0xED crashes. These were verified on the bench and in-motion (Section 10). See Appendix I.0.7 for the script used.

10 Notes on In-Motion Vehicle Tests

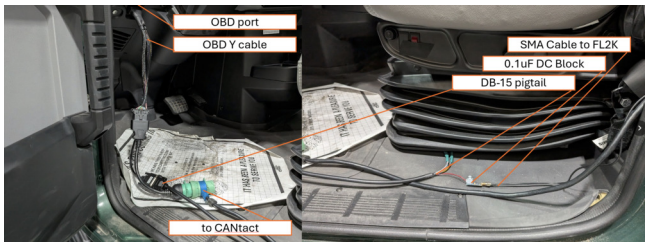


Figure 12: Right: a 10uF DC block resulting in AC-coupling of the FL2K SDR (not pictured) connected to a power amplifier (not pictured). Left: Connection of the same to the in-cab on board diagnostic (OBD) port.



Figure 13: Top Left: failure state of instrument cluster after any and all the ECU crashes described in Section 10 while still in-motion. Wide: in-cab picture of in-motion vehicle test with black bar redacting.

Onsite testing used a host-provided tractor that did not perfectly match the source vehicle of the vulnerable firmware but whose EC80 part number matched perfectly – because the host did not have any EC80s that were not already updated by ID9363. We manually flashed the target ECU with the vulnerable firmware via BDM. Then the target ECU was part-calibrated to the tractor using OEM tools, and baseline ABS functionality was confirmed via hard braking before further testing.

[16] It would also be possible to send CAN data with a write primitive and without RCE.

[17] `4B42` is ASCII 'KB', presumably for Knorr-Bremse, Bendix's parent company (since 2024 after a joint venture starting in 1993).

Testing focused on the crashes in the handlers for J1587 PIDs 0xC2 and 0xED, identified during bench testing. We injected J2497 signals using an FL2K Software Defined Radio (SDR) transmitter [49] [50] and power amplifier. Due to insufficient J2497 filtering at the diagnostic port, we were able to use a simple in-cab setup: AC-coupled via a 10uF capacitor to the diagnostic port's VBAT pin [18] (Figure 12). As shown in [4], this simulates wireless injection [6].

We conducted in-motion tests below 5 mph and at ~9 mph — fast enough to trigger ABS pulsing but safe for occupants. At 9 mph, after observing the ECU crash via CAN, the driver initiated a hard brake to check for ABS pulsing. In all cases, CAN traffic ceased. ECU recovery always required a battery disconnect [19]. The crash consistently caused loss of speedometer, steering assist, and shifting (Figure 13). The results are summarized in Table 5. The scripts used to trigger the 0xED and 0xC2 DoS vulnerabilities can be found in Appendices I.0.9, and I.0.8, respectively.

One difference emerged: in the PID 0xED case, the EC80 stopped streaming signal data but still responded to UDS TP requests. This was not observed in the PID 0xC2 case or during PID 0xED bench testing.

We repeated these in-motion tests on a second vehicle: owned by a fleet and from a different OEM than the previous. This EC80 was 'factory installed' and was not yet updated by 'ID9363'. During these tests we were able to test also the 0xC7 'backdoor' impacts. In this vehicle the impact of crashing the ABS controller did not include loss of speedometer nor steering assist nor shifting; but other impacts were the same. This was also true for the 0xC7 'backdoor'.

Vuln.	Target	Speed	UDS	Notes	ABS Disabled
0xED	1ec80	$<$ 5 mph	Yes	Loss of shift, steering assist, speedo. Multiple cluster faults	–
0xED	3ec80	$<$ 5 mph	No	Mulitple Cluster faults.	–
0xED	1ec80	$>$ 5 mph	Yes	Loss of shift, steering assist, speedo	**Yes**
0xED	3ec80	$>$ 5 mph	No	Mulitple Cluster faults.	**Yes**
0xC2	1ec80	$<$ 5 mph	No	Loss of speedo, steering assist	–
0xC2	3ec80	$<$ 5 mph	No	Mulitple Cluster faults.	–
0xC2	1ec80	$>$ 5 mph	No	Loss of shift, steering assist, speedo	**Yes**
0xC2	3ec80	$>$ 5 mph	No	Mulitple Cluster faults.	**Yes**
0xC7	3ec80	$>$ 5 mph	No	Multiple Cluster faults including "Dyno Mode". Loss of shifting in Reverse	**Yes**

Table 5: In-motion test results (In all cases streaming data stopped).

11 Random Noise Sufficiency

The EDGE SCI2 interrupt vulnerability could obviously have been triggered by noise: constant framing errors and/or parity errors could cause false 0x00 bytes to fill the receive buffer and edge interrupts firing coincident with these could cause the partial reset and eventual overwrite of pointers. It is not as clear for the PID processing handlers' vulnerabilities.

As was mentioned in Appendix B, the only receiver error checking is if the checksum is consistent. These checksums are the two's complement of the sum of all bits in the message. The checksum is not able to detect many-bit errors, and e.g. fails to detect 'symmetric' errors (pairs of bit flips). It is easy to imagine the checksum failing to prevent reception of invalid messages in noisy conditions.

There are not any established models of bit errors or of spurious data reception of the Intellon SSCP485 [24] that we are aware of. However, from previous research [53] it is known to be possible for a message on the bus to have several of its bits flipped and/or be truncated on reception in the presence of noise.

It is very typical of trailer brake controllers to stream PID 0xC2 and so it could be imagined that bit flips of an existing 0xC2 message could cause a large value of 'n' and hence the same DoS crash we observed in Section 10.

Trailer brake controllers do not typically stream 0xED messages [20]. Nor do they stream 0xC7 messages. It is, however, possible that the 0xC2 (or other streaming messages) get bit flipped to 0xED or 0xC7 and truncated to the right length. This could not be reasonably described as a likely possibility though.

12 Related Work

Binary analysis of bare-metal firmware often faces unique constraints. While [54] uses emulation, we employ minimum-viable concolic analysis where emulation was unavailable. We address BinDiff's small-function mismatch issues [55] by applying QBinDiff [56] with manual interrupt handler anchoring to enable useful results. Furthermore, our findings confirm that silent security patching [57] is occuring also in heavy vehicle firmware.

[18] this was confirmed by observing Trailer ABS fault dash light status and 0xF001 PGN with `python -m can.viewer -i cantact -c 0 -b 500000 --filter=00F00100:00FFFF00` (using python-can [51] and CANtact [52]).

[19] This is a relevant condition because some tractors are configured without a battery disconnect switch, requiring a manual removal of the battery leads or a manual cycling of a fuse to recover the ECU.

[20] although they can send them on power up and on request at power-up of the tractor

Seminal work [58] identified "standard" vulnerabilities like buffer overflows that, we show, persist in modern components. We extend J1708 security research [59] to J2497 and apply the wireless write attack [53] to tractor equipment. Complementing J1939 diagnostic research [60], we demonstrate that mission-time (as distinct from diagnostics) data processing defects in J2497 (as opposed to J1939) directly impact tractor safety.

13 Generalizable Contributions

Our work provides several contributions that generalize across different architectures and target devices. First, we demonstrate that hiveplots can be effectively used to survey the current disassembly of a binary. This visualization technique is particularly useful for identifying 'configuration' issues and spotting segmented areas in bare-metal firmware images. Second, we found that the use of manual anchoring in QBinDiff for each of the interrupt handlers is a critical step in achieving useful and accurate binary diff results in its application to bare metal firmware. Finally, our approach of performing live RAM dumping is generally applicable to any case where a debugger is not fully functional. This technique is critical for discovering dynamically resolved functions, particularly when function pointers or their offsets are stored exclusively in RAM.

14 Conclusions

This work demonstrates binary differential analysis on legacy automotive architectures, using heuristics and symbolic analysis to overcome tooling limitations. Our analysis reveals several vulnerabilities were patched by the ID9363 update.

While the vendor emphasized safety compliance regarding random noise, our analysis confirms a broader security impact. Random line noise is unlikely to generate the valid J1587 frames and frame sequences required to trigger these vulnerabilities. Specifically, the patch removes the tractor ECU's J1587 parsing stack (except LAMP/ABS event handling), eliminating vulnerable handlers for PIDs 0xC2, 0xED, and 0xC7.

We classify the removed functionality as critically flawed, containing unauthenticated memory corruption (PIDs 0xC2, 0xED) and hardcoded weak authentication (PID 0xC7). In-motion testing validated the severity: exploiting the PID 0xC2 buffer overflow caused immediate ABS Denial of Service (DoS), physically manifesting as loss of steering assist, speedometer, shifting and ABS pulsing. The PID 0xED handler induced the same DoS via memory corruption and the 0xC7 handler also via abuse of weak authenticated functionality. Bench testing confirmed RCE via the memory corruption in the PID 0xC2 handler.

The ID9363 update mitigates risks from attackers with wireless adjacency or other forms of J2497 network access, improving the compliance to requirements NGW-S-001 through -005 [29]. Bendix's proactive recall and firmware update represent a commendable and responsible action. This is particularly important as users are believed to prioritize safety recalls over security patches. However, the lack of distinct CVE assignments for these flaws may obscure the patch's security criticality. What is recommended – by ISO/IEC 29147 [61] and CISA [62] – is that vulnerabilities which are patched should be communicated to users so that they can make their own informed risk calculations.

The features removed align with the security architecture for tractor J2497 reception [21] detailed in all of:

1. the ATA TMC position paper on next generation tractor trailer interfaces [63]: "Pertaining to PLC communications as described in SAE J2497: only the MID 10 and 11 lamp messages, MID 125 J2497 identification, and MID 87 active ABS event shall be permitted on new tractor and trailer equipment…"
2. the CISA mitigations in advisory ICSA-25-021-03 [64]: "To most effectively mitigate general vulnerabilities of the powerline communication, any [tractors] utilizing J2497 technology should disable all features where possible, except for backwards-compatibility with LAMP ON detection only."
3. the latest version of SAE J2497 [1]: "For tractor PLC units, it is recommended that the software implements only reception of the LAMP ON and OFF messages … All other reception of messages should be performed on other network types."

Given the potential for a) malicious J2497 signal injection and b) compromised J2497-capable telematics units, we strongly advocate that all tractor J2497-receiving equipment, especially brake controllers, adopt the security-conscious approach demonstrated by the EC80 update.

Acknowledgments

We would like to thank our colleagues at the National Motor Freight Traffic Association Inc. whose expertise and dedication have been invaluable to the completion of this research. Particularly Anne Zachos for the continuous assistance in analysis and onsite testing.

We would like to thank AIS for access to their Class 8 vehicle multiple times during this research. Thank you to Hannah Silva and Jesse Norton for sharing S12X development materials and EC80 experience. Thank you to Chris York for the

[21] for tractor J2497 transmission the same sources recommended that the tractor mitigate attacks on older trailer equipment; this patch does not appear to add any such mitigations.

trailhead. Thank you to Jonatan Mars for critical support at a critical time. We would also like to thank many industry engineers for their support of this research – the rest of whom would prefer not to be named. Without the support of the engineers at these vehicle OEMs and suppliers this work would not be possible. Last but not least, this work would also not be possible without the support of the member fleets of the NMFTA Inc.

We used various large language models (Gemini [65] 2, 2.5-pro, 3-pro) for tasks such as: drafting Python code and Mermaid diagrams. We acknowledge the open-source tools used: Pandoc [66], Matplotlib [67], Hiveplotlib [68], Zynamics BinDiff [43], Quarkslab QBinDiff [45], Quokka [69], Python 3 [70], python-can [51], py-hv-networks [71], RP1210 python [72], Osmocom FL2K [50], Linkerscope [73], Mermaid [74], and Graphviz [75].

Open Science - Availability

The complete code listings of scripts mentioned in the paper are available in the Artifact Appendix of this paper https://doi.org/10.5281/zenodo.19664811. With the exception of the QBinDiff analysis which will be made available later as 'howto' document pull request to the Quarkslab QBinDiff repository docs.

None of the firmware images used in the analysis will be made available. The reverse engineering of this work focused only on the machine code patched by the firmware update. This work makes no claims on the presence nor absence of vulnerabilities in the remainder of the firmware domains; hence, there is insufficient guarantee of safety of sharing the firmwares. In lieu of the firmware, we provide the following in the Artifact Appendix:

1. disassembly listings (with instruction bytes included) for the deleted and select modified functions only
2. PID Handler and Context Tables deleted in the update
3. an .s19 file containing only the above plus the interrupt vectors for the application

The ID9363 updater executable will not be made available because it contains copies of all the new firmware images.

The ID9363 firmware update CAN logs will not be made available because they contain the entire updated firmware in the clear as well as seed-key exchange pairs which cannot be guaranteed to be insufficient to reconstruct the seed-key routines in closed-form.

Ethical Considerations

This research prioritizes disclosure and minimizes potential harm. Prior to commencing the technical analysis, in early 2025, the manufacturer of the brake ECU was contacted and informed of our intent to conduct research on their firmware update for the safety recall and requested to collaborate. Again, as the analysis neared completion, the manufacturer was contacted in an attempt to provide them with an overview of our findings. Two of the three OEMs have received the results in this paper. NHTSA and Transport Canada have also been informed of this research and results.

Crucially, this paper discusses only functions and vulnerabilities addressed by the manufacturer's safety recall. Only deleted function code is shared. Limited side-by-side code modifications are shared to limit disclosure of code in the field. The PoC provided can be used to confirm RCE but does not disclose the details of CAN bus or other physical control by the ECU. The updated firmware has been deployed on ~310,000 units [12] at the time of this writing. These vulnerabilities are confirmed absent in updated firmware versions Z300822, Z302578, Z302579 (one for each affected OEM). Thus, disclosing these vulnerabilities introduces no new risks. By focusing on already-patched issues, this research aims to contribute to the broader understanding of automotive security without inadvertently exposing users to unmitigated risks.

Finally, the in-vehicle testing was conducted in a controlled environment, utilizing a closed test track for in-motion vehicle testing. All testing was performed with the explicit cooperation and support of an OEM, ensuring that safety protocols were rigorously followed and that no public roads or operational vehicles were subjected to unverified or potentially hazardous conditions.

References

[1] SAE Truck and Bus Control and Communications Network Committee, "J2497 power line carrier communications for commercial vehicles." Warrendale, PA, 2026.

[2] *Federal motor vehicle safety standard no. 121, air brake systems*. Washington, D.C.: National Highway Traffic Safety Administration.

[3] MITRE Corporation, "CVE-2020-14514." 2020. Accessed: Nov. 2025. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14514

[4] B. Gardiner, "Disclosure of confirmed remote write to J2497 aka PLC4TRUCKS," *NMFTA, Alexandria, VA, Letter, March*, 2022.

[5] National Motor Freight Traffic Association, "Actionable mitigations options v9." 2022. Accessed: Nov. 2025. [Online]. Available: https://nmfta.org/wp-content/media/2022/11/Actionable_Mitigations_Options_v9_DIST.pdf

[6] MITRE Corporation, "CVE-2022-26131." 2022. Accessed: Nov. 2025. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26131

[7] Bendix Commercial Vehicle Systems LLC, "24E086 chronology." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RMISC-24E086-5355.pdf

[8] *ISO 14229: Road vehicles – unified diagnostic services (UDS)*. Geneva, Switzerland: International Organization for Standardization, 2020.

[9] PACCAR Incorporated, "Safety recall report 24V-915." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V915-6438.PDF

[10] Navistar, Inc., "Safety recall report 24V-818." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V818-4283.PDF

[11] Volvo Trucks North America, "Safety recall report 24V-790." 2024. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2024/RCLRPT-24V790-3386.PDF

[12] National Highway Traffic Safety Administration, "NHTSA recalls by manufacturer." Accessed: Nov. 2025. [Online]. Available: https://datahub.transportation.gov/Automobiles/NHTSA-Recalls-by-Manufacturer/mu99-t4jn

[13] National Highway Traffic Safety Administration, "Technical service bulletin 10194446." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10194446

[14] National Highway Traffic Safety Administration, "Technical service bulletin 10176745." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10176745

[15] National Highway Traffic Safety Administration, "Technical service bulletin 10222229." 2024. Accessed: Nov. 2025. [Online]. Available: https://dot.report/bulletins/10222229

[16] National Motor Freight Traffic Association, "Bendix EC80 recall: Safety and security implications." 2025. Accessed: Jan. 2026. [Online]. Available: https://nmfta.org/bendix-ec80-recall-safety-and-security-implications/

[17] Bendix Commercial Vehicle Systems LLC, "Safety recall report 25E-073." 2025. Accessed: Nov. 2025. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2025/RCLRPT-25E073-3346.pdf

[18] Land Line Media, "Defective bendix ECUs have prompted recall of nearly half a million trucks with latest paccar recall." 2025. Accessed: Nov. 2025. [Online]. Available: https://landline.media/defective-bendix-ecus-have-prompted-recall-of-nearly-half-a-million-trucks-with-latest-paccar-recall/

[19] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-007." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_007_en_000.pdf

[20] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-006." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch_27_006_en_000.pdf

[21] Bendix Commercial Vehicle Systems LLC, "Technical bulletin TCH-27-008." 2025. Accessed: Nov. 2025. [Online]. Available: https://www.bendix.com/media/services-and-support/product-action-center-pdfs/tch-27-008_en_000.pdf

[22] *J1587: Electronic data interchange between microcomputer systems in heavy-duty vehicle applications*. Warrendale, PA: SAE International, 2013.

[23] *J1708: Serial data communications between microcomputer systems in heavy-duty vehicle applications*. Warrendale, PA: SAE International, 2016.

[24] Intellon Corporation, *SSC P485 PL transceiver IC data sheet*. 1997.

[25] S. Behere, X. Zhang, V. Izosimov, and M. Törngren, "A functional brake architecture for autonomous heavy commercial vehicles." 2016. Available: https://legacy.sae.org/publications/technical-papers/content/2016-01-0134/

[26] *ISO 26262: Road vehicles – functional safety*. Geneva, Switzerland: International Organization for Standardization, 2018.

[27] ZF Friedrichshafen AG, "mBSP XBS factsheet." 2024. Accessed: Nov. 2025. [Online]. Available: https://www.zf.com/public/org/ZF_CVS_mBSP_XBS_Factsheet_EN_296135.pdf

[28] Vehicle Cybersecurity Working Group (VCRWG), National Motor Freight Traffic Association, "NMFTA vehicle cybersecurity requirements." 2024. Accessed: Nov. 2025. [Online]. Available: https://github.com/nmfta-repo/nmfta-vehicle_cybersecurity_requirements

[29] B. Gardiner, J. Maag, and K. Tindell, "Security requirements for vehicle security gateways," SAE International, 2024-01-2806, 2024. Accessed: Nov. 2025. [Online]. Available: https://www.sae.org/papers/security-requirements-vehicle-security-gateways-2024-01-2806

[30] NXP Semiconductors, “MC9S12XEQ512 data sheet.” Accessed: Nov. 2025. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100.pdf

[31] NXP Semiconductors, “MC9S12XE family reference manual.” Accessed: Nov. 2025. [Online]. Available: https://www.nxp.com/docs/en/reference-manual/MC9S12XERM.pdf

[32] ELDB, “XPROG-box.” Accessed: Nov. 2025. [Online]. Available: https://www.eldb.eu/

[33] PEmicro, “PROGS12Z flash programmer software.” Accessed: Nov. 2025. [Online]. Available: https://www.pemicro.com/

[34] Motorola, “Motorola s-record description (PDF).” Accessed: Nov. 2025. [Online]. Available: https://deramp.com/downloads/mfe_archive/060-Standards%20and%20Specifications/Hex%20Data%20Formats/Motorola%20S%20Record.pdf

[35] DARPA, “Assured micropatching (AMP).” Accessed: Nov. 2025. [Online]. Available: https://www.darpa.mil/program/assured-micropatching

[36] *J1939: Serial control and communications heavy duty vehicle network*. Warrendale, PA: SAE International, 2018.

[37] C. Miller and C. Valasek, “Adventures in automotive networks and control units,” IOActive, 2014. Accessed: Nov. 2025. [Online]. Available: https://www.ioactive.com/wp-content/uploads/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf

[38] Pulse Security, “Reversing the ducati 696 ECU part 2.” 2022. Accessed: Nov. 2025. [Online]. Available: https://pulsesecurity.co.nz/articles/ducati-696-part2

[39] National Security Agency, *Ghidra*. (2025). Available: https://ghidra-sre.org/

[40] Hex-Rays, *IDA pro*. (2026). Available: https://hex-rays.com/ida-pro/

[41] hotwolf, *HSW12*. Accessed: Nov. 2025. [Online]. Available: https://github.com/hotwolf/HSW12

[42] NXP Semiconductors, *HiWave debugger*. 2010.

[43] Zynamics, *BinDiff*. Available: https://www.zynamics.com/bindiff.html

[44] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra, “Hive plots—rational approach to visualizing networks,” *Briefings in bioinformatics*, vol. 13, no. 5, pp. 627–644, 2011.

[45] R. Cohen, R. David, R. Mori, F. Yger, and F. Rossi, “Improving binary diffing through similarity and matching intricacies,” in *Proc. Of the 6th conference on artificial intelligence for defense*, 2024.

[46] A. One, “Smashing the stack for fun and profit,” *Phrack Magazine*, vol. 7, no. 49, 1996, Available: http://phrack.org/issues/49/14.html

[47] National Motor Freight Traffic Association, “j2497-keyhole.” Accessed: Nov. 2025. [Online]. Available: https://github.com/nmfta-repo/j2497-keyhole

[48] Masterjun, “Super mario world "total control" TAS.” 2014. Accessed: Nov. 2025. [Online]. Available: https://tasvideos.org/2513M

[49] T. Yapo, “FL2K experiments.” Accessed: Nov. 2025. [Online]. Available: https://hackaday.io/project/164346-fl2k-sdr

[50] Osmocom, “Osmo-FL2k project.” Accessed: Nov. 2025. [Online]. Available: https://osmocom.org/projects/osmo-fl2k

[51] python-can Developers, *Python-can*. (2024). Available: https://python-can.readthedocs.io/

[52] E. Evenchick, “CANtact.” Accessed: Nov. 2025. [Online]. Available: https://cantact.io/

[53] B. Gardiner, “Mitigating PLC4TRUCKS remote write,” in *Proceedings of the 9th escar USA conference*, Ypsilanti, MI: escar, 2022. Available: https://escar.info/downloads

[54] R. Tsang, D. Joseph, S. Salehi, *et al.*, “FFXE: Dynamic control flow graph recovery for embedded firmware binaries,” in *33rd USENIX security symposium (USENIX security 24)*, 2024, pp. 5573–5590.

[55] A. Marcelli, M. Graziano, X. Ugarte-Pedrero, Y. Fratantonio, D. Balzarotti, and A. Francillon, “How machine learning is solving the binary function similarity problem,” in *USENIX security symposium*, 2022.

[56] E. Mengin and R. David, “QBinDiff : Un outil de diffing binaire flexible et modulaire,” in *SSTIC 2020 (symposium sur la sécurité de l’information et des communications)*, 2020.

[57] J. Ruge, J. Classen, F. Gringoli, and M. Hollick, “Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets,” in *USENIX security symposium*, 2020.

[58] K. Koscher *et al.*, “Experimental analysis of a modern automobile,” in *2010 IEEE symposium on security and privacy*, 2010.

[59] B. Hass, Y. Burakova, L. Millar, and A. Weimerskirch, “Truck hacking: An experimental analysis of the SAE J1939 standard,” in *10th USENIX workshop on offensive technologies (WOOT 16)*, Austin, TX: USENIX Association, 2016.

[60] R. Chatterjee, C. Green, and J. Daily, “Exploiting diagnostic protocol vulnerabilities on embedded networks in commercial vehicles,” in *Symposium on vehicles security and privacy (VehicleSec)*, 2024.

[61] *ISO/IEC 29147: Information technology – security techniques – vulnerability disclosure*. Geneva, Switzerland: International Organization for Standardization, 2018.

[62] Cybersecurity and Infrastructure Security Agency, "Shifting the balance of cybersecurity risk: Principles and approaches for security-by-design and -default," 2023. Accessed: Nov. 2025. [Online]. Available: https://www.cisa.gov/resources-tools/resources/shifting-balance-cybersecurity-risk-principles-and-approaches-security-design-and-default

[63] Technology & Maintenance Council, "Position paper 2024-3: Next generation tractor-trailer technical needs," American Trucking Associations, 2024. Accessed: Nov. 2025. [Online]. Available: https://tmc.trucking.org/sites/default/files/TMC_PP-2024_3_NEXTGEN_TRACTOR_TRAILER_TECHNICAL_NEEDS%20.pdf

[64] Cybersecurity and Infrastructure Security Agency, "ICS advisory (ICSA-25-021-03) bendix EC-80." 2025. Accessed: Nov. 2025. [Online]. Available: https://www.cisa.gov/news-events/ics-advisories/icsa-25-021-03

[65] Google DeepMind, "Gemini: A family of highly capable multimodal models." 2023. Accessed: Nov. 2025. [Online]. Available: https://deepmind.google.com/technologies/gemini/

[66] J. MacFarlane, *Pandoc*. (2026). Available: https://pandoc.org/

[67] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[68] Hiveplotlib Developers, *Hiveplotlib*. (2025). Available: https://github.com/hiveplotlib/hiveplotlib

[69] Quarkslab, *Quokka*. (2024). Available: https://github.com/quarkslab/quokka

[70] Python Software Foundation, *Python programming language*. (2026). Available: https://www.python.org/

[71] TruckHacking organization (originally by Haystack), *Py-hv-networks*. (first published 2016). Available: https://github.com/TruckHacking/py-hv-networks

[72] dfieschko, *RP1210*. (2019). Available: https://github.com/dfieschko/RP1210

[73] LinkerScope Developers, *LinkerScope*.

[74] K. Sveidqvist, *Mermaid*. (2026). Available: https://mermaid.js.org/

[75] Graphviz Authors, *Graphviz*. (2026). Available: https://graphviz.org/

[76] Wireshark Foundation, *Wireshark*. Available: https://www.wireshark.org/

A Appendix: ID9363 Update Process Further Details

The update process is executed by the ID9363 updater executable, where we captured the J1939 [36] traffic sent between it and the target ECU. The update process uses UDS [8] on J1939 [36]; in all cases we observed the traffic was unicast, default priority, between the vehicle diagnostic adapter at address 0xF9 and the brake controller at 0x0B: resulting in the two extended (29-bit) CAN IDs 0x18DA0BF9 and 0x18DAF90B. The CAN traffic was filtered on those two IDs and processed with Wireshark [76] to reconstruct the ISO-TP multi frame messages used by UDS and then these were analyzed to reconstruct the ID9363 update process, a simplified summary of which is reproduced in Table F.1 (the remainder of this appendix is available in appendix F at https://doi.org/10.5281/zenodo.19664811).

B Appendix: J2497 Reception Architecture of EC80

The reception is driven by events from the SCI2 and PIT hardware peripherals as illustrated in Figure G.3, PIT being used to manage timeouts. SCI2 is a UART that supports several interrupt event types and the receiver state machine makes use of 'data ready' (RDRF), IDLE, and edge (RXEDGIF) events. The entire state machine is distributed over several functions which all access address 0x3C52, a set of control bitfields, summarized in Table G.1 (the remainder of this appendix is available in appendix G at https://doi.org/10.5281/zenodo.19664811).

C Appendix: J1587 PID Processing

Table C.1 shows all the PID processing which are removed by the firmware update. These were discovered and compiled by walking PID handler data structures found in the firmware and detailed in the remainder of this appendix – which is available in appendix H at https://doi.org/10.5281/zenodo.19664811

D Appendix: Code Listings

The remainder of this appendix is available in appendix I at https://doi.org/10.5281/zenodo.19664811.

E Appendix: Disassembly Listings

The remainder of this appendix is available in appendix J at https://doi.org/10.5281/zenodo.19664811.

J1708 message	PID / PID Request Description	Data Input Size
mm0031 / mm803188	Request for 0x31 49 ABS Control Status	one byte data (broadcast req) / zero bytes data (unicast req)
mm003E / mm803E88	Request for 0x3E 62 Retarder Inhibit Status	one byte data / zero bytes data
mm0054 / mm805488	Request for 0x54 84 Road Speed	one byte data / zero bytes data
mm0097 / mm809788	Request for 0x97 151 TC Control State	one byte data / zero bytes data
mm009E / mm809E88	Request for 0x9E 158 Battery Potential (V), Switched	one byte data / zero bytes data
mm00A8 / mm80A888	Request for 0xA8 168 Battery Potential (V)	one byte data / zero bytes data
mm00C2 / mm80C288	Request for 0xC2 194 Transmitter System Diagnostic Code and Occurrence Count Table	one byte data / zero bytes data
mm00C4 / mm80C488	Request for 0xC4 196 Diagnostic Data/Count Clear Response	one byte data / zero bytes data
mm00C7 / mm80C788	Request for 0xC7 199 Traction Control Disable State	one byte data / zero bytes data
mm00D1 / mm80D188	Request for 0xD1 209 ABS Control Status, Trailer	one byte data / zero bytes data
mm00D6 / mm80D688	Request for 0xD6 214 Vehicle Wheel Speeds	one byte data / zero bytes data
mm00E9 / mm80E988	Request for 0xE9 233 Unit Number (Power Unit)	one byte data / zero bytes data
mm00EA / mm80EA88	Request for 0xEA 234 Software Identification	one byte data / zero bytes data
mm00ED / mm80ED88	Request for 0xED 237 Vehicle Identification Number	one byte data / zero bytes data
mm00F3 / mm80F388	Request for 0xF3 243 Component Identification	one byte data / zero bytes data
mm2A	42 Pressure Switch Status	1 byte data
mm46	70 Parking Brake Switch Status	1 byte data
mm54	84 Road Speed	1 byte data
mm75	117 Brake Primary Pressure	1 byte data
mm76	118 Brake Secondary Pressure	1 byte data
mmA9	169 Cargo Ambient Temperature	2 bytes data
mmB4	180 Trailer Weight	2 bytes data
mmC2	194 Transmitter System Diagnostic Code and Occurrence Count Table	variable bytes data. byte count, followed by sets of diagnostic data
mmC303	195 Diagnostic Data Request / Clear	3 bytes data
mmC7	199 Traction Control Disable State	variable length data. byte count, state flags, ASCII 'access code' of 0-15 characters selected by the manufacturer ...
mmD1	209 ABS Control Status, Trailer	variable bytes data, up to 3x5 bytes for 5 trailers byte count, followed by the status bytes
mmED	237 Vehicle Identification Number	variable bytes data. byte count, followed by the VIN ASCII
mmF5	245 Total Vehicle Distance	4 bytes data
mmF7	257 Total Engine Hours	4 bytes data
mmFE88C5 / mmFE88C6	254 Proprietary DLE	permitted lengths of only 0xC5 and 0xC6
mmFF73	115 Trailer Pneumatic Supply Line Pressure	one byte data
mmFF7B	123 Door Status	one byte data

Table C.1: J1587 PIDs supported by the before update firmware. 'mm' refers to any MID value except 0A, 0B, or 57, which are handled earlier in processing.

ARTIFACT EVALUATED
USENIX
AVAILABLE

Artifact Appendix
Reversing a Brake ECU Firmware Update

Ben Gardiner, National Motor Freight Traffic Association Inc.

A Artifact Appendix

A.1 Abstract

The purpose of the artifact(s) is to share IDA Pro / python scripts which we hope will be useful to others and also to support the claim that the firmware update to remediate the recall was in fact a security patch. There are some artifacts which we cannot safely release: i.e. the firmware files. The before and after firmware files of the ECUs would be required for functional and/or reproducible badges. And these firmware files will not be released because – as mentioned in the paper – we cannot make any guarantees on the absence of other vulnerabilities in neither the before nor after ECU firmwares. For this reason and the fact that the scripts require a paid license to IDA Pro 9.1, we request only confirmation of availability of the artifacts.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifacts submitted contains documentation and disassembly of vulnerabilities that have been patched in the wild through a safety recall. The provided disassembly is limited to the functions removed during the update to minimize the risk of exposing code from the updated, currently deployed firmware. All provided scripts are for offline static analysis or safe, controlled test-bench use.

A.2.2 How to access

The artifacts are submitted alongside this appendix document.

- **IDA Scripts: `ida-*.py`** Reverse engineering scripts (IDA Python). These are standalone scripts produced by inlining functions from an internal EC80-specific analysis library (which is not released in full to prevent leaking potentially attacker-beneficial information):
  - `ida-pflash-s12x-setup.py`: Sets up the memory map for the S12X processor in IDA Pro, handling the PFLASH and PPAGE banking.
  - `ida-concolic-s12x-resolve.py`: Performs a basic concolic analysis to resolve indirect calls.
  - `ida-jumptables-s12x-resolve.py`: Detects and creates jump tables by parsing the disassembly and creating manual code xrefs.
  - `ida-funcptr-s12x-discovery.py`: Scans the firmware image for potential function pointers to identify additional functions.
  - `ida-pid-ec80-naming.py`: Walks the PID handler tables and names the handler functions based on the PID and context.
  - `ida-deletedfn-report.py`: Validates the call tree to identify deleted functions.
- **Disassembly:** The disassembly of the deleted/modified functions (`disassembly-of-changed-functions.md`).
- **S19 File: `changes-2ec80.s19`** An S19 memory dump containing the global addresses of the deleted functions, PID and context tables, and interrupt vectors from the pre-update `2ec80` target. Note that this file contains only the global addresses of these items in PFLASH; it has not been 'expanded' to the paged format expected by many tools.
- **Testing Scripts: `j*.py`** scripts for fuzzing the J2497 ingress of the target and for testing susceptibility to simple PoCs of the vulnerabilities.
  - `j1587-pid-fuzzer.py`: Used to fuzz the J1587 PID handlers and discover the vulnerabilities in the 0xC2 and 0xED PID Handlers.
  - `j2497-sdr-badones-transmit.py`: Used to trigger the 0xC2 and 0xED vulnerabilities during in-motion tests. It emits the test signal on an attached FL2K SDR. (Expects the `j2497-keyhole` project to be installed as a package or run from the same directory).

These files are available at Zenodo DOI https://doi.org/10.5281/zenodo.19664811.

A.2.3 Badges

We apply for the **Artifacts Available** badge only. We cannot apply for the Functional or Reproducible badges because the full firmware files cannot be released as mentioned in the paper.

A.2.4 Hardware dependencies

- A standard desktop or laptop computer is sufficient for offline static analysis and reviewing the disassembly and scripts. A Windows 10 or 11 x64 machine is required for the `j1587-pid-fuzzer.py` test script.
- A FL2K VGA adapter with a SMA connector adapter is required for the `j2497-sdr-badones-transmit.py` test script
- A RP1210 VDA with J2497 support (or one with J1708 support and a converter) and any python-can CAN adapter is required for the `j1587-pid-fuzzer.py` test script.
- A generic USB HID relay switch (VID: 0x16C0, PID: 0x05DF) is required for power cycling the target in the `j1587-pid-fuzzer.py` test script.

A.2.5 Software dependencies

- **Python 3.12 or later**
- **IDA Pro** (with S12X architecture support) for running the IDA Python scripts, if desired.
- **py-hv-networks, RP1210, python-can, scapy, bitstring, hidapi, numpy, tabulate, and bincopy** pypi packages must be installed for the test and setup scripts.
- **Your own S12X firmware .s19 file** is required for running the IDA Pro scripts.
- **osmo-fl2k** is required to provide the `fl2k_file` executable used by the `j2497-sdr-badones-transmit.py` script.

A.3 Evaluation Workflow

The provided `changes-2ec80.s19` file can be loaded into an S12X disassembler to verify the structure of the deleted code. Please note that it is not sufficient for reproducing the execution of any of the scripts. The provided `disassembly-of-changed-functions.md` file contains disassemblies which can be used for reference if a working S12X disassembler is not available.

A.3.1 Major Claims

- **(C1):** The ID9363 update removed substantial J1587 PID processing functionality from the EC80 firmware [1].
- **(C2):** The removed functionality contained critical vulnerabilities, including buffer overflows and out-of-bounds writes (e.g., in handlers for PIDs `0xC2` and `0xED`). This is proven by the experiment (E1).
- **(C3):** The removed functionality contained a hardcoded credential (PID `0xC7`). This is proven by the experiment (E1).

A.3.2 Experiments

- **(E1):** [Verifying Removed Functionality] [30 human-minutes + 0 compute-hour + 0GB disk]: Reviewing the disassembly allows manual verification of claims C2 and C3.
  - **How to:** Review the disassemlby of the .s19 file provided or text of disassembly file provided.
  - **Preparation:** Ensure the `changes-2ec80.s19` file can be disassembled or the `disassembly-of-changed-functions.md` can be referenced.
  - **Execution:**
    1. Open the provided disassembly files.
    2. Observe the `Context at 0xDC3D` and `PID Table at 0xD9DD` sections to confirm that the following functions are PID handlers.
    3. Locate `sub_F1B623` and `sub_F2BA46` to verify claim C2.
    4. Locate `sub_F2AB7A` to verify claim C3.
    5. Load the `.s19` file to confirm byte sequences match the provided disassembly.
  - **Results:** Reviewing the text and `.s19` file is sufficient to corroborate the claims manually.

A.4 Notes on Reusability

This artifact uses standard Python 3 and IDA Pro environments. The scripts can be adapted for analyzing similar S12X-based firmware images.

Additionally, for users wishing to extract their own firmware via the HiWave debugger, memory can be dumped with the following script/commands (note: set the command window's cache limit to unlimited first):

```
DMM CACHINGOFF
DMM WRITEREADBACKOFF
HCS12X_MAP4000 RAM
db 0x4000'L..0x7FFF'L
db 0x0f8000'G..0x0FBFFF'G
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.

[1] Note that we cannot provide an artifact to confirm the claim that the code is deleted because that would require releasing the 'after' firmware image.