600 Jahre Universität Rostock

Traditio et Innovatio

Bachelorarbeit zum Thema

Sicherheitsanalyse durch Entwicklung eines Rogue Device zur Echtzeitmanipulation maritimer Steuerungssysteme

Studiengang:	Informatik
Vorgelegt von:	Jakob Engelbert Tomahogh
Matrikelnummer:	221201101
Bearbeitungszeitraum:	15. November 2024 – 04. April 2025
Erstgutachter:	M. Sc. Marvin Davieds
Zweitgutachter:	Prof. Dr. rer. nat. Clemens H. Cap

Lehrstuhl für Informations- und Kommunikationsdienste
Institut für Informatik
Fakultät für Informatik und Elektrotechnik
Universität Rostock

Gedruckt auf säurefreiem und elementar-chlorfrei gebleichtem Papier.

Stand:: 5. März 2025

Inhalt

Kurzzusammenfassung

Im traditionellen Sinne bezieht sich der Begriff Typografie auf die Gestaltung von Druckwerken mit beweglichen Lettern (Typen). Anfänglich fand dies insbesondere im Bleisatz bzw. dem Satz mit Holzlettern statt.

In der Medientheorie steht Typografie für gedruckte Schrift in Abgrenzung zu Handschrift (Chirografie) und elektronischen sowie nicht literalen Texten.

Heute bezeichnet Typografie meist den medienunabhängigen Gestaltungsprozess, der mittels Schrift, Bildern, Linien, Flächen und Leerräumen alle Arten von Kommunikationsmedien gestaltet. Typografie ist in Abgrenzung zu Kalligrafie, Schreiben oder Schriftentwurf das Gestalten mit vorgefundenem Material.

Abstract

In the traditional sense, the term typography refers to the design of printed works with movable letters (types). Initially, this was done in lead typesetting or wood typesetting.

In media theory, typography stands for printed type in contrast to handwriting (chirography) and electronic as well as non-literal texts.

Today, typography usually refers to the media-independent design process that uses type, images, lines, surfaces and empty spaces to create all kinds of communication media. In contrast to calligraphy, writing or type design, typography is the design with found material.

Kapitel 1

Einleitung

In diesem Kapitel

1.1 Motivation

Jedes Jahr werden zahlreiche Passagiere mit Schiffen befördert. Dabei kann es vorkommen, dass bösartige Personen ein Teil dieser Passagiere sind. Ein Angriff auf die Steuerung eines Schiffes könnte katastrophale Folgen haben. Dennoch wird die Sicherheit von Schiffen oft vernachlässigt. Wenn die Kommunikation auf einem internen Systemen nicht ausreichend geschützt ist, könnte ein Angreifer durch physischen Zugriff auf das Schiff die Kommunikation manipulieren und so die Steuerung des Schiffes übernehmen. Auf großen Passagierschiffen ist das Risiko besonders hoch, da es schwierig ist, einen solchen Angriff zu bemerken. Ein bösartiger Angreifer kann sich als Passagier tarnen. Es gibt dort zu viele Menschen, um verdächtige Aktivitäten sofort zu bemerken.

1.2 Ziel der Arbeit

Dadurch soll aufgezeigt werden, wie wichtig es ist, die Kommunikation den Systemen eines Schiffes zu schützen und Aufmerksamkeit auf die Sicherheit dieser zu lenken. Die Arbeit erforscht die Möglichkeit eines Angriffs mit Steuerungsübernahme. Dazu wird ein

Rogue Device entwickelt, welches in der Lage ist externe Steuerungsbefehle zu erhalten und mit diesen Kontrollnachrichten auf die Schiffskommunikation zu senden. Dabei wird auf Netzwerke zugegriffen, welche die Kommunikation zwischen den verschiedenen Steuerungssystemen des Schiffes ermöglichen. Als Protokolle spielen dabei NMEA 2000, NMEA 0183 und J1939 eine wichtige Rolle. Unter anderem wird auch auf eine serielle Schnittstelle zugegriffen, um mehr Kontrolle über das System zu erlangen.
Hier soll die Machbarkeit eines solchen Angriffs gezeigt werden. Zusätzlich werden die Auswirkungen auf die Steuerung des Schiffes analysiert. Es soll ein Bewusstsein für die Sicherheit von Schiffen geschaffen werden und mögliche Gegenmaßnahmen vorschlagen.
Um die Sicherheitslücken zu veranschaulichen, soll das Schiff mit einem Gaming-Controller gesteuert werden. Dies soll zeigen, wie einfach es ist, die Steuerung eines Schiffes zu übernehmen, wenn die Kommunikation nicht ausreichend geschützt ist.

Kapitel 2

Grundlagen

In diesem Kapitel

2.1 CAN-Bus

Bei einem Can-Bus handelt es sich um eine serielle Netzwerktechnologie, welche mehrere Geräte mit einem Draht verbindet. Die Entwicklung des Can-Bus wurde von Bosch im Jahr 1983 begonnen. 1986 wurde der erste Can-Bus Standard veröffentlicht. Die Motivation der Entwicklung war die effiziente Kommunikation zwischen den Steuergeräten in einem Auto. Als günstige Nebenwirkung konnte damit die Kabelmenge reduziert werden, da alle Geräte mit einem Bus verbunden werden können. Durch die höhere Zuverlässigkeit und Funktionalität des Can-Bus, wurde dieser schnell in der Autoindustrie etabliert. Aber auch in anderen Sektoren, wie z.B. der Medizintechnik, der Gebäudeautomation oder der Luftfahrt, spielt der Can-Bus mittlwerweile eine wichtige Rolle. [Vos08, Seiten 2-10]
Man sprich von einem Bussystem, da alle Geräte gleichberechtigt sind. Dass heißt, dass jedes Gerät Nachrichten senden und empfangen kann. Die Nachrichten werden nach Broadcasting-Prinzip übertragen. Jede Nachricht wird von allen Knoten empfangen, aber nur die Knoten, die die Nachricht benötigen, verarbeiten sie. Diese werden nicht Einhaltung der Protokollregeln überprüft, da dies zu einer größeren unnötigen Last auf dem Bus führen würde. Jedoch wird die Integrität der Nachrichten durch eine Prüfsumme sichergestellt. Das wird auch mit einer Acknowledge-Nachricht(ACK) bestätigt. Wird eine Nachricht

nicht bestätigt, wird von dem Sender eine Fehlermeldung auf den Bus gesendet. Bei einer fehlerhaften Nachricht reagieren die Knoten mit einer Fehlermeldung, die wieder der gesamte Bus empfängt. Wenn ein Knoten dauerhaft fehlerhafte Nachrichten sendet, wird dieser vom Bus getrennt. Die auf dem Bus gesendeten Daten werden mit einer Nachrichten-ID versehen, die die Priorität der Nachricht angibt. Die Nachrichten mit der niedrigsten ID haben die höchste Priorität. Die Maximale Länge einer Nachricht beträgt 8 Byte. Durch die vergleichsweise geringe Länge der Nachrichten, kann eine geringe Latenz erreicht werden. Dabei kann eine höchste Baudrate von 1Mbit/s gesetzt werden. [Vos08, Seiten 13-19]
Alle Knoten in einem Can-Bus sind mit einem Zweiadrigen Kabel verbunden. Diese werden als High(CAN_H) und Low(CAN_L) bezeichnet. Der Bus ist an beiden Enden mit einem Widerstand von 120 Ohm abgeschlossen um Reflexionen zu vermeiden. [Vos08, Seite 132]
Eine Nachricht auf einem CAN-Bus ist im allgemeinen in einem Can-Frame verpackt. Dieser beginnt mit einem Startbit, welches Start of Frame (SOF) genannt wird. Darauf folgt das Arbitration Field, in dem die Nachrichten-ID und ein Bit RTR (Remote Transmission Request) gesetzt wird. Das RTR-Bit wird gesetzt, wenn der Sender eine Antwort auf die Nachricht erwartet. Das Control Field wird für die Datengröße und die Nachrichtenlänge verwendet. Im Data Field sind die eigentlichen Nutzdaten kodiert. Das CRC Field enthält eine Prüfsumme, welches die Richtigkeit der Nachricht überprüft. Hier folgt ein ACK Field, welches die Prüfsumme bestätigt. Die Nachricht endet mit einem End of Frame (EOF). Danacht folgt ein Interframe Space (IFS) von 3 Bit, welches eine Pause zwischen den Nachrichten darstellt. [Vos08, Seite 36]

2.1.1 Erweitertes Can-Protokoll

Das Standard Can-Protokoll hat einen 11 Bit Identifier, während das erweiterte Can-Protokoll 29 Bit Identifier besitzt. Die Society of Automotive Engineers (SAE) hat das J1939-Protokoll entwickelt, um die Kommunikation in Nutzfahrzeugen zu verbessern. Der Fokus liegt auf der Kommunikation mit dem Antrieb des Fahrzeugs. Dafür wird der 11 Bit Identifier mit J1939 auf 29 Bit erweitert um mehr verschiedene Nachrichten zu ermöglichen.
Auf einem Can-Bus können der Standard 11 Bit Identifier und der erweiterte 29 Bit Identifier gleichzeitig verwendet werden. Wenn zwei Nachrichten den gleichen 11 Bit Identifier haben, wird die Nachricht mit dem 11 Bit Identifier bevorzugt immer die höhere Priorität haben. Die spezifierten Baudraten sind hier 250kbit/s und 500kbit/s. Die Hauptkomponenten des erweiterten Identifier sind die Priorität, eine Parameter Group Number (PGN) und eine Quell-Adresse. Das PGN-Feld ist aus 1 Bit Data Page (DP), 1 Bit Extended Data Page (DP), einer Protocol Data Unit (PDU) und einem PDU spezifischen Feld (PS) aufgebaut. Die DP und EDP bestimmen mit welchem Standard die Nachricht gesendet wird. Für SAE J1939 werden aber nur zwei Kombinationen verwendet. Das PS-Feld kann entweder eine Zieladresse oder eine Gruppenerweiterung (GE) sein. Welches PS-Feld verwendet wird, hängt von der PDU ab.

Wenn diese einen Wert von unter 240 hat, wird das PS-Feld als Zieladresse verwendet. In dem anderen Fall wird das PS-Feld als Gruppenerweiterung verwendet. [MG18]

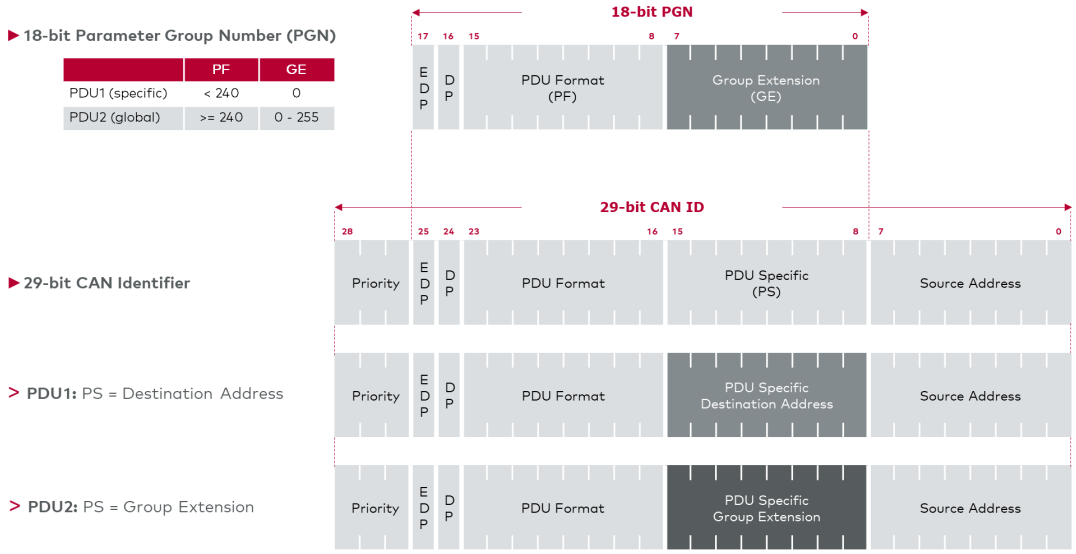


Abbildung 2.1: Header einer J1939-Nachricht auf dem CAN-Bus [Gmbb](letzter Zugriff: 28.01.2025)

Auch wenn in einem CAN-Bus jedes Gerät immer jede Nachricht empfängt, soll es durch die Zieladresse möglich sein, dass nur das Gerät die Nachricht verarbeitet, welches die Nachricht benötigt. Um eine Nachricht an alle Knoten zu senden, wird die Zieladresse auf 255 gesetzt. Mit der Gruppenerweiterung ist es nicht möglich eine Nachricht zielgerichtet an bestimmte Geräte zu senden. [MG18]

2.2 Raspberry Pi

Ein Raspberry Pi ist ein Einplatinencomputer, der von der Raspberry Pi Foundation entwickelt wurde. Dieser den Prozessor mit Grafikeinheit, Arbeitsspeicher, Speicher und Anschlüsse auf einem einzigen Board integriert. Es handelt sich um einen vollwertiger Computer, der kleiner als normale PCs ist. Für die Größe ist der Raspberry Pi leitsungsstark, während er einen recht günstigen Preis von unter 100€ hat. Er kann mit einem normalen Betriebssystem betrieben werden. Trotzdem gibt es ein spezielles Betriebssystem, das für den Raspberry Pi entwickelt wurde, das Raspberry Pi OS (ehemals Raspbian). Dieses basiert auf der Linux-Distribution Debian. Um viele Funktionen erfüllen zu können, hat der Raspberry Pi viele Anschlüsse. Der Raspberry Pi 5 hat 4 USB-Anschlüsse, 2 Micro-HDMI-Anschlüsse, 1 Ethernet-Anschluss, 1 USB-C-Anschluss für die Stromversorgung und einen Micro-SD-Kartensteckplatz.

2.2.1 Raspberry Pi als Rogue Device

Unter einem Rogue Device versteht man ein Gerät, welches sich unautorisiert und unauffällig in ein Netzwerk integriert. [Sca+08] Dies kann ein Raspberry Pi sein, der sich in ein Netzwerk einbindet und Daten abfängt oder manipuliert. Hierüber können sich Angreifer Zugriff auf das Netzwerk verschaffen. Ein solches Gerät kann dazu verwendet werden, um eigenen Code auszuführen, der beispielsweise Daten verändert oder weitere Angriffe vorbereitet. Zudem kann es von Angreifern aus der Ferne gesteuert werden, wodurch gezielte Manipulationen oder Spionageangriffe möglich sind. Darüber hinaus lässt sich ein Rogue Device nutzen, um Informationen über das Netzwerk zu sammeln, etwa durch das Mithören von Kommunikation oder die Analyse von Sicherheitsmechanismen. Das ermöglicht es, Schwachstellen im Netzwerk aufzudecken und gezielt auszunutzen.

2.3 State of the Art

Die Technologie des CAN-Bus ist seit einiger Zeit etabliert und wird in vielen Bereichen eingesetzt. Daher gibt es auch viele Tools und Bibliotheken, die es ermöglichen, mit dem CAN-Bus zu arbeiten. In diesem Abschnitt wird der aktuelle Stand der Technik vorgestellt.

2.3.1 Anbindung von Raspberry Pi an CanBus

In der folgenden Arbeit soll ein Raspberry Pi an einen CAN-Bus angeschlossen werden. Dafür gibt es verschiedene Möglichkeiten. Im Folgenden werden einige Möglichkeiten vorgestellt.
Die erste Möglichkeit ist die Verwendung eines Microchip MCP251x, welcher als CAN-Controller dient. Dieser wird mit dem Microchip MCP2551 ergänzt, welcher als CAN-Transceiver dient. Diese Kombination ermöglicht es, den Raspberry Pi mit dem CAN-Bus zu verbinden. Der MCP251x wird über die SPI-Schnittstelle(Serial Peripheral Interface) des Raspberry Pi angeschlossen. Dazu wird ein Treiber benötigt, der die Kommunikation zwischen dem Raspberry Pi und dem MCP251x ermöglicht. Ein solcher Treiber ist bereits seit dem 05.05.2015 in das Raspberry Pi OS (ehemals Raspbian OS) integriert. [SKJ16] Um diese Möglichkeit zu vereinfachen, gibt es auch verschiedene HATs (Hardware Attached on Top), die solche Microchips bereits integriert haben. Ein Beispiel dafür ist das PiCan2 von SK Pang Electronics. Dieses Board wird auf den GPIO-Pins (General Purpose Input/Output) des Raspberry Pi gesteckt. Es erlaubt damit dem Raspberry Pi mit dem CAN-Bus zu kommunizieren. Das PiCan2 unterstützt eine Geschwindigkeit von bis zu 1Mbit/s. [PL19]
Es besteht auch die Möglichkeit, einen USB-CAN-Adapter zu verwenden. Damit ist die Anbindung eines beliebigen Linux-System an den CAN-Bus möglich. Unter Umständen ist es auch mit Windows und MacOS möglich. Ein Beispiel für einen solchen Adapter ist der UCAN

von Fysetc. Dieser Adapter ist mit einer open-source Firmware ausgestattet. Zusätzlich ist auch die Hardware open-source. Die Verbindung zum Raspberry Pi erfolgt über USB-C. Zum CAN-Bus müssen lediglich CAN_H und CAN_L und die Masse verbunden werden. [Fys] (letzter Zugriff: 17.02.2025)

2.3.2 Überseztung von CAN-Nachrichten

Es gibt viele Software-Tools, die es ermöglichen, mit dem CAN-Bus zu arbeiten. Um einen Überblick zu bekommen, werden hier einige vorgestellt. Diese Tools unterscheiden sich in ihrem Funktionsumfang, ihrer Benutzerfreundlichkeit und den unterstützten Anwendungsfällen. Einige Programme sind speziell für die Analyse und das Debugging von CAN-Nachrichten ausgelegt, während andere umfassende Entwicklungs- und Simulationsmöglichkeiten bieten. Im Folgenden werden die wichtigsten Eigenschaften, Einsatzbereiche und Besonderheiten der einzelnen Tools vorgestellt.

- **canCommander**: Ein Tool, das es ermöglicht, CAN-Nachrichten zu senden und zu empfangen. Es bietet eine einfache Benutzeroberfläche und umfangreiche Analysefunktionen. Mit canCommander können CAN-Nachrichten aufgezeichnet, analysiert und bearbeitet werden. Dabei können auch Nachrichten gesendet und injiziert werden. Es wird auch eine schon vorbereitete Platine verkauft, die es ermöglicht, mit dem CAN-Bus zu arbeiten. Trotzdem ist canCommander ein open-source Projekt und kann auch auf anderen Plattformen verwendet werden. Diese Plattformen beschränken sich aber auf Mikrocontroller, wie Arduino Uno oder ESP32. [KuK] (letzter Zugriff: 15.02.2025)
- **cantools**: Eine Python-Bibliothek, die es ermöglicht, CAN-Nachrichten zu dekodieren. Mit cantools können CAN-Daten in ein lesbares Format umgewandelt werden, das von Menschen verstanden werden kann. Die Bibliothek bietet eine Vielzahl von Funktionen und unterstützt verschiedene Protokolle und Datenformate. Damit ist cantools ist ein nützliches Werkzeug für die Analyse und Verarbeitung von CAN-Nachrichten und eignet sich für die Entwicklung von Anwendungen, die mit dem CAN-Bus arbeiten. [can] (letzter Zugriff: 15.02.2025)
- **CANoe**: Ein eigentständiges kommerzielles Tool, das von Vector Informatik GmbH. entwickelt wurde. Es bietet eine vielzahl an verschiedenen Funktionen. CANoe ermöglicht die Entwicklung, Simulation und Analyse von CAN-Netzwerken. Es bietet eine umfangreiche Funktionalität und unterstützt verschiedene Protokolle und Datenformate. [Gmba] (letzter Zugriff: 15.02.2025)
- **can_decoder**: Ein Open-Source-Projekt, das es ermöglicht, CAN-Nachrichten zu dekodieren. Es wurde von der Firma CSS-Electronics entwickelt und ist auf GitHub

verfügbar. Allerdings ist das Projekt nicht mehr aktiv und wird nicht mehr weiterentwickelt. Trotzdem kann es ein nützliches Werkzeug für die Analyse und Verarbeitung von CAN-Nachrichten sein. [Fal] (letzter Zugriff: 15.02.2025)

- **SavvyCAN**: Eine C++ Anwendung, mit der CAN-Nachrichten aufgezeichnet und durch Reverse-Engineering analysiert werden können. Hiermit können auch einzelne Nachrichten dekodiert werden. SavvyCAN ist ein Open-Source-Projekt und kann auf GitHub gefunden werden. Zusätzlich gibt es Unterstützung auf Linux und Windows. [Kid] (letzter Zugriff: 15.02.2025)

- **can-utils**: Ein Set von Linux-Tools, die es ermöglichen, mit dem CAN-Bus zu arbeiten. Damit ist es möglich, CAN-Nachrichten zu senden und zu empfangen. Es ist ein Open-Source-Projekt und kann auf GitHub gefunden werden. Diese Tools bieten eine Grundlage für die Entwicklung von Anwendungen, die mit dem CAN-Bus arbeiten. [lin] (letzter Zugriff: 15.02.2025)

Um der Nutzlast der CAN-Bus Nachrichten Informationen zuzuordnen, ist eine DBC-Datei essentiell. Diese Datei schlüsselt auf, welche Informationen in den verschiedenen CAN-Nachrichten enthalten sind. Da auch sensitive Informationen in den Nachrichten enthalten sein können, wird diese Datei in den meisten Fällen nicht öffentlich verfügbar gemacht. Ohne diese Datei können zwar Brute-Force Ansätze verwendet werden, allerdings ist dies sehr aufwendig und nicht zielführend. [Cho+21]
Um die vorher genannten Tools zu verwenden, ist es also notwendig, die DBC-Datei zu haben. Bei canCommander sind einige DBC-Dateien bereits vorinstalliert. Bei cantools kann die DBC-Datei in das Programm geladen werden. Allerdings kann es eine Schwierigkeit darstellen, die bestimmte DBC-Datei zu erhalten.

Kapitel 3

Konzept und Systemdesign

In diesem Kapitel

3.1 Aufbau Schiffsysteme

Ein Schiff besteht aus vielen verschiedenen Systemen, welche verschiedene Aufgaben haben. In diesem Fall wird das Steuerungssystem des Schiffes betrachtet. Dabei werden die wichtigen Systeme für die Arbeit abstrahiert, um die Funktionsweise des Systems zu verstehen.

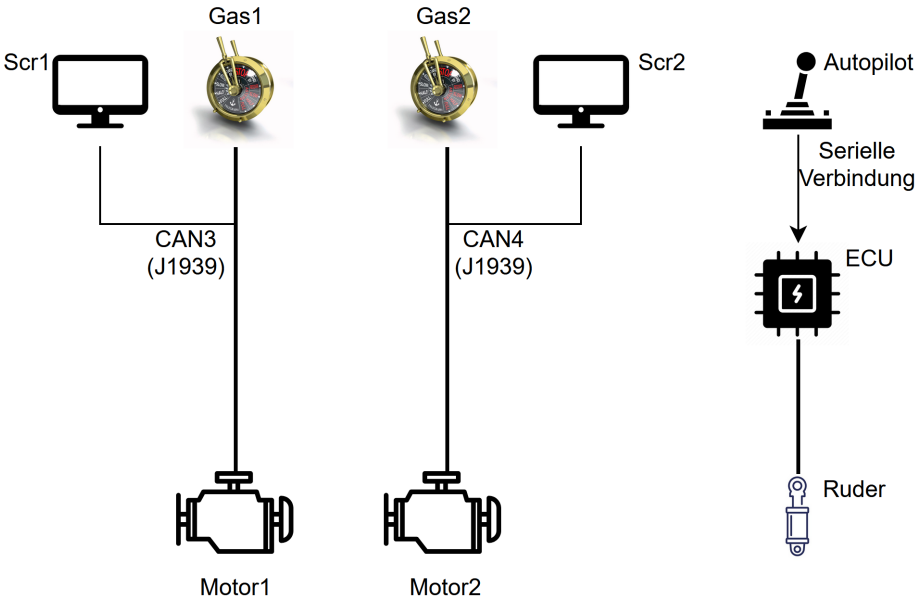


Abbildung 3.1: Vereinfachte Darstellung der Systeme auf der Limanda

Die wichtigen Systeme für diese Arbeit begrenzen sich auf die Gashebel und den Autopiloten. Die beiden Gashebel sind jeweils mit einem CAN-Bus verbunden. An diesen Bussen werden die Steuerbefehle für die Motoren gesendet. Dort sind unter anderem auch Bildschirme für den Schiffsführer angeschlossen. Als Higher-Layer-Protokoll wird J1939 genutzt.
Der Autopilot ist ein eigenständiges System, welches über eine serielle Schnittstelle mit dem Ruder verbunden ist. Dieser hat keine Verbindung zu den Gashebeln oder Motoren. Er kann lediglich Signale an die Rudersteuerung senden. Dabei ist dieser mit einer seriellen Verbindung an die Rudersteuerung angeschlossen. Diese Verbindung ist unverschlüsselt. Die Rudersteuerung ist mit dem Ruderstellmotor über eine unbekannte Verbindung angeschlossen. Diese Verbindung ist jedoch nicht relevant für diese Arbeit.

3.2 Steuerungslogik des Controllers

Der benutzte Controller ist ein Xbox Series X Controller. Dieser wurde gewählt, da er eine gute Haptik hat und weit verbreitet ist. Zusätzlich ist er kabellos und kann somit frei bewegt werden. Um die Steuerung des Schiffes zu ermöglichen, müssen die Eingaben des Controllers in Steuerbefehle umgewandelt werden. Dies passiert auf dem Raspberry Pi. Der Controller wird über Bluetooth mit dem Raspberry Pi verbunden. Dort werden die Eingaben des Controllers ausgelesen und in einem Python-Programm in Steuerbefehle umgewandelt.
Um eine einfache Steuerung zu ermöglichen, wird im folgenden die Tastenbelegung aufgeschlüsselt. Um alle gewünschten Funktionen umzusetzen, werden nicht alle Tasten benötigt.

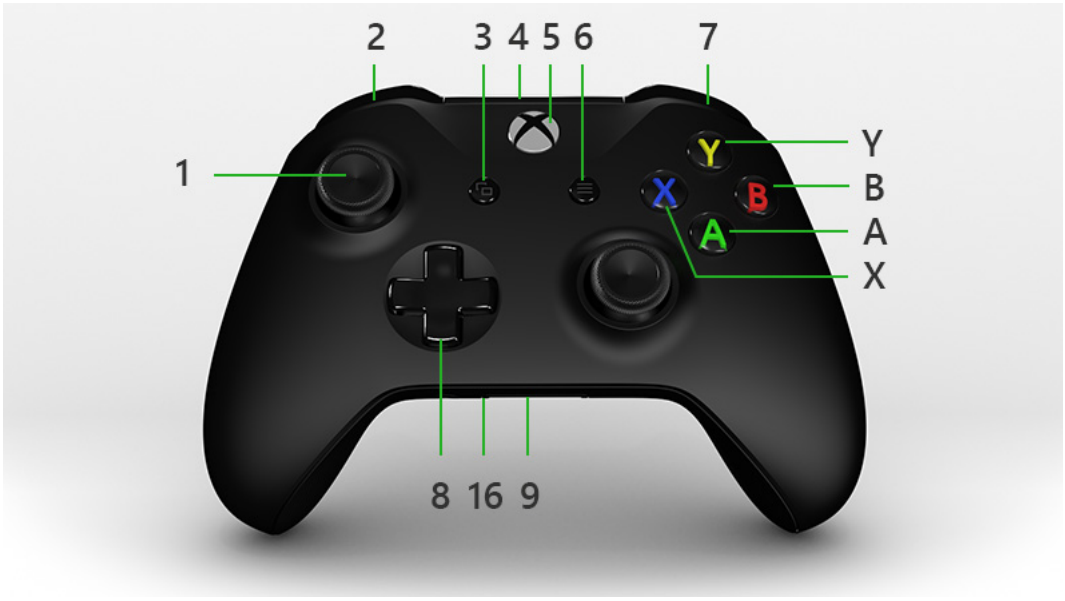


Abbildung 3.2: Vorderseite des Controllers [Mic](letzter Zugriff: 24.01.2025)

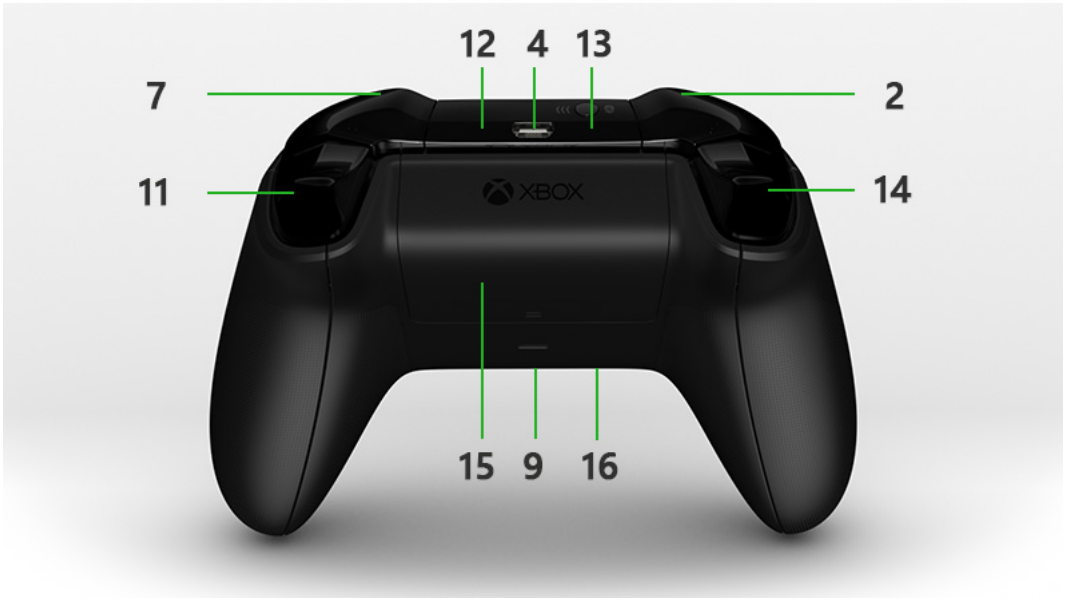


Abbildung 3.3: Rückseite des Controllers [Mic](letzter Zugriff: 24.01.2025)

Nummerierung der Taste	Funktion
1	Bewegung des Ruders
2	Reduzierung der linken Gashebelposition
7	Reduzierung der rechten Gashebelposition
11	Erhöhung der rechten Gashebelposition
14	Erhöhung der linken Gashebelposition
B + 2	Umschalten des Rückwärtsgangs am linken Motor
B + 7	Umschalten des Rückwärtsgangs am rechten Motor
B + 2 + 7	Umschalten des Rückwärtsgangs an beiden Motoren (basierend auf dem derzeitigen Gang am rechten Motor)

Das Einlegen des Rückwärtsgangs ist durch eine Tastenkombination so gewählt, dass es nicht aus Versehen passieren kann. Mit jeweils der Taste 2 oder 7 wird die Gashebelposition reduziert. Mit der zusätzlichen Betätigung der Taste B wird der Rückwärtsgang eingelegt an dem jeweiligen Motor. Wenn die Tasten 2, 7 und B gleichzeitig betätigt werden, wird der Rückwärtsgang für beide Motoren gleichzeitig umgeschalten. Dies ist so gewählt, da eine Verzögerung bei dem Umschalten eines Rückwärtsgangs von 10 Sekunden eingebaut ist. Das soll dem Getriebe ermöglichen, den Gang zu wechseln ohne eine direkte Gaseingabe. Nun muss es aber auch möglich sein, beide Rückwärtsgänge gleichzeitig umzuschalten, daher die vergleichsweise komplexe Tastenkombination.

3.3 Integration des Rogue Device

Damit der Controller die Steuerbefehle an das Schiff senden kann, muss das Rogue Device in das System integriert werden. In diesem Fall ist das Rogue Device der Raspberry Pi. Damit dieser möglichst unbemerkt in das System integriert werden könnte, muss der Controller drahtlos verbunden werden. Um eine unentdeckte Integration zu ermöglichen, muss die Art der Stromversorgung unabhängig von dem Schiff sein. Dafür könnte ein Akku genutzt werden. Je nach Art der Anwendung kann der Akku Um die Kommunikation von dem Rogue Device zu dem Schiff zu ermöglichen, müssen die einzelnen Systeme angesteuert werden. Um die Gashebelposition zu verändern, wird der Raspberry Pi mit dem Can-Bus des Schiffes verbunden.
Die grobe Struktur des Rogue Devices soll wie folgt aussehen:

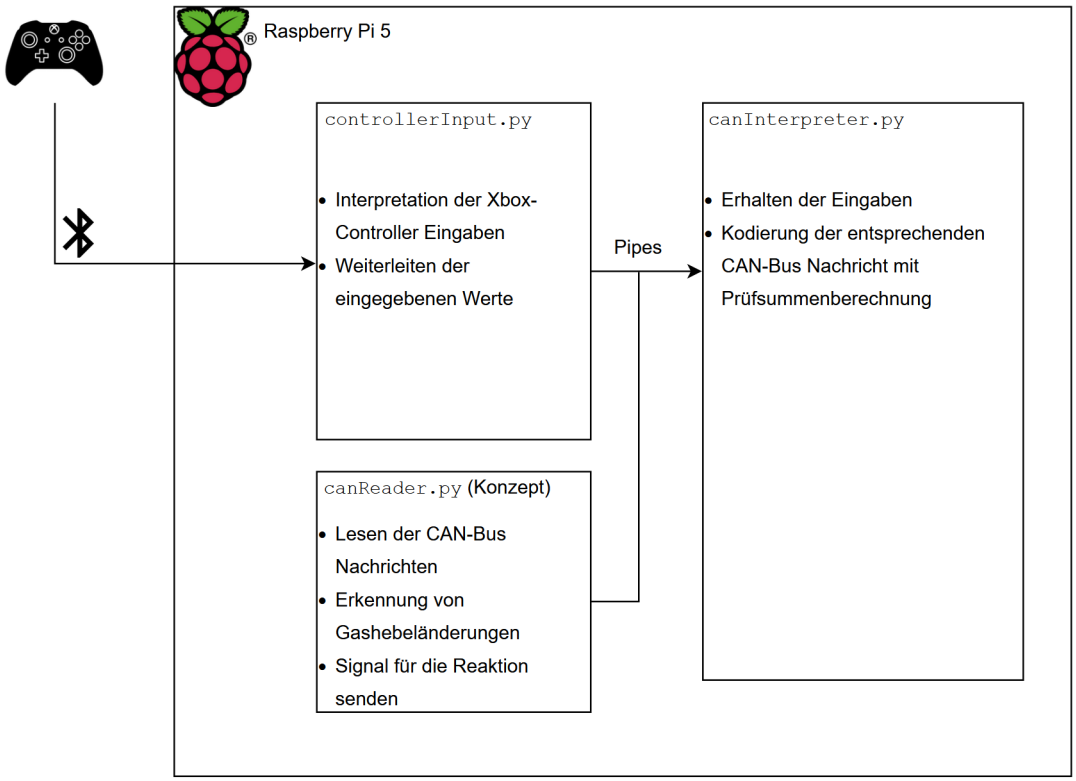


Abbildung 3.4: Konzept der Programmstruktur auf dem Rogue Device

Hier wird der Controller über Bluetooth mit dem Raspberry Pi verbunden. Dieser liest die Eingaben des Controllers aus und sendet die entsprechenden Steuerbefehle an das Schiff. Dafür wird der Can-Bus des Schiffes genutzt. Sollte der Gashebel in der normalen Benutzung vom Schiffsführer benutzt werden, wird ein Signal an den Can-Bus gesendet. Dieses Signal wird dann an die Motoren weitergeleitet. Dafür wird `canReader.py` genutzt. Dieses Programm liest die Nachrichten des Can-Bus reagiert auf die wichtigen Nachrichten für diese Arbeit. Um diese Eingabe zu verhindern, muss auf diese Nachricht geachtet und reagiert werden. Mit dem Lesen des Can-Bus kann die entsprechende Nachricht entdeckt werden. Dann kann eine Nachricht von dem Rogue Device gesendet werden, um die Gashebelposition zu überschreiben. Eigene Nachrichten müssen dazu erkannt werden, um eine Endlosschleife von eigenen Nachrichten zu verhindern. Dafür könnte der Nachrichtenzähler überwacht werden. Aus den aufgezeichneten Nachrichten hat sich herausgestellt, dass der Nachrichtenzähler wenige verschiedene Werte hat. Daher sind diese Werte zu den eigenen erstellten Nachrichten verschieden. Dies ist eine einfache Methode, um die eigenen Nachrichten zu erkennen. Eine weitere Möglichkeit wäre, eine eigene CAN-ID zu berechnen. Diese könnte dann genutzt werden, um die eigenen Nachrichten zu erkennen.
konzept der Autopilot-Steuerung

- Autopilot kommuniziert seriell mit Rudersteuerung

- Autopilot kann nur das Ruder steuern
- Autopilot-Nachrichten abfangen und analysieren
- evtl replayen in eigener Reihenfolge
- evtl eigene Nachrichten nachbauen

Wie in der Abbildung 2.1 zu sehen, besteht der Header aus einer PGN, einer Quelladresse und einer Priorität. Um nun Nachrichten an den Motor zu senden, kann eine Nachricht des Gashebels abgefangen werden. Die PGN kann für die eigene Nachricht genutzt werden. Die Quelladresse kann man auch kopieren. Die Priorität sollte möglichst klein gewählt werden, damit die eigene Nachricht bevorzugt wird. In der eigenen Nachricht kann dann die gewünschte Gashebelposition gesendet werden.

3.3.1 Rückmeldung der Eingaben

Was muss ich dabei beachten? Muss eine Rückmeldung für die Eingaben geschehen? Wenn ja, wie? (kleiner OLED-Bildschirm oder App) Es muss eine Art der Rückmeldung geben, um in etwa die Eingaben im Vergleich zum momentanen Zustand zu sehen. Dabei sollte die Rückmeldung möglichst unauffällig sein. Ein kleiner OLED-Bildschirm könnte benutzt werden, allerdings muss dieser physisch an den Raspberry Pi angeschlossen werden. Das würde die Versteckung des Rogue Devices erschweren. Eine App auf einem Handy könnte die Rückmeldung geben. Diese App könnte dann die gewünschten Positionen anzeigen. Dafür muss der Raspberry Pi mit dem Handy verbunden werden. Dies könnte über Bluetooth geschehen. Hier ist zu beachten, dass die Verbindung stabil ist und nicht von anderen Geräten gestört wird. Jedoch kann ist hier mit einem größeren Aufwand zu rechnen, da die App erst entwickelt werden muss. Eine weitere Möglichkeit würden Vibrationen im Xbox Controller sein. Diese könnten genutzt werden, wenn die derzeitige Eingabe ein Maximum oder Minimum erreicht hat. Allerdings ist dies nicht so genau wie eine Anzeige auf einem Bildschirm. Es könnten auch nur wenige bestimmte Positionen mittgeteilt werden.
Eine solche Rückmeldung ist wichtig, um eine sogenannte Pilot Induced Oscillation (PIO) zu verhindern. Das Phänomen tritt vorallem bei Flugzeugen auf, wenn der Pilot zu stark gegensteuert. Dabei reagiert das Flugzeug auf die Steuerbefehle des Piloten und der Pilot reagiert auf die Reaktion des Flugzeugs. Dies führt zu einer Schwingung, die sich immer weiter verstärkt. [McR95]
Ein vergleichbares Phänomen kann auch bei Schiffen auftreten, wenn auch nicht im gleichen Ausmaß. Bei Schiffen sind die Bewegungen im Wasser langsamer und weniger stark. Dennoch kann es zu einer Schwingung kommen, wenn der Schiffsführer zu stark gegensteuert. Um dies zu verhindern, ist eine Rückmeldung der Eingaben wichtig. Diese Rückmeldung kann dann dazu genutzt werden, um die Eingaben genauer zu wählen.

Kapitel 4

Implementierung

In diesem Kapitel

4.1 Umsetzung des Rogue Device

Wie vorher schon beschrieben, wird ein Raspberry Pi 5 als Rogue Device benutzt. Als Betriebssystem wird Raspberry Pi OS benutzt. Dieses Betriebssystem ist eine auf Debian basierende Distribution, die speziell für den Raspberry Pi entwickelt wurde. Da der Raspberry Pi mit Linux betrieben wird, ist es die Bedienung vergleichbar mit einem normalen Linux-System. Der Standard Package Manager ist apt. Unter dessen Benutzung werden Python 3.12 und die benötigten Bibliotheken installiert. Damit sind die Vorraussetzungen für die weitere Implementierung gesetzt.

4.2 Verbindung Rogue Device - Controller

Im ersten Schritt wird der Controller mit dem Raspberry Pi verbunden. Der Raspberry Pi wird mit Raspberry Pi OS betrieben. Dieses Betriebssystem ist eine auf Debian basierende

Distribution, die speziell für den Raspberry Pi entwickelt wurde. Als Standard-Bluetooth-Treiber wird BlueZ verwendet. Dieser ist bereits vorinstalliert und muss nicht extra installiert werden. Da es sich in diesem Fall um einen XBox Controllers handelt, müssen die richtigen Treiber (xboxdrv) installiert werden. Diese können im apt-Repository gefunden werden. Zusätzlich muss der Enhanced Re-Transmission Mode (ERTM) deaktiviert werden. Dieser Modus ist standardmäßig aktiviert und verhindert die korrekte Verbindung des Controllers. Zum Schluss soll der Controller auf die aktuelle Firmware geupdatet werden. Dies kann über die Xbox Accessories App gemacht werden, allerdings ist dies nur auf Windows möglich. Nun kann der Controller mit dem Raspberry Pi verbunden werden. Dies geschieht über die Bluetooth-Einstellungen des Raspberry Pi. Um die Verbindung zu testen, kann eine Webapplikation benutzt werden. Diese zeigt die Eingaben des Controllers an. Zum Beispiel: hardwaretester (abgerufen am 02.01.2025)
Als Programmiersprache wird Python 3 benutzt. Dies ist eine einfache Sprache, die viele Bibliotheken hat. Für die Controller-Eingaben wird die Bibliothek Pygame benutzt. Diese Bibliothek ist einfach zu benutzen und hat viele Funktionen. Sobald der Controller mit dem Raspberry Pi verbunden ist, wird dieser von Pygame erkannt. Die Eingaben des Controllers können dann in Variablen gespeichert werden. Dabei gilt es zu beachten, dass der Gashebel oder die Ruderposition nicht zu schnell verändert werden. Dies könnte zu unerwünschten Ergebnissen führen. Daher werden diese Werte mit Tasten des Controllers eingegeben, welche nicht nur eine binäre Eingabe haben. Diese werden als Achsen bezeichnet. Diese ermöglichen eine stufenlose Eingabe. Bei einer vollständigen Eingabe soll die Gashebelposition nach 20 Sekunden 100% erreichen. So ist sichergestellt, dass die Gashebelposition nicht zu schnell verändert wird. Die Ruderposition soll nach 2 Sekunden in jede Richtung jeweils 100% erreichen. Dies ist ein Kompromiss zwischen Geschwindigkeit und Genauigkeit.

4.3 Übersetzung Signale Controller - Schiff

Es gibt zuerst ein Programm, welches die Signale des Controllers erhält und in passende Variablen in Python interpretiert. Diese Variablen müssen dann in Nachrichten für den Can-Bus umgewandelt werden. Hierfür gibt es ein weiteres Programm. Damit die beiden Programme miteinander kommunizieren können, wird Inter-Process-Communication (IPC) benutzt. Als Methode werden hierbei Pipes benutzt. Diese sind einfach zu implementieren und haben eine automatische Synchronisierung zwischen den Prozessen. Das bedeutet, dass die Prozesse nicht aufeinander warten müssen, sondern einfach weiterarbeiten können. Es wird durch den Puffer der Pipe sichergestellt. Wenn dieser voll ist, wird der schreibende Prozess angehalten, bis der lesende Prozess den Puffer geleert hat. Dies ist ein einfaches und effizientes Verfahren, um die beiden Prozesse zu synchronisieren [VJ15].

4.3.1 CAN-Bus

Die Übertragenen Eingabewerte werden in dem nächsten Programm `canInterpreter.py` erkannt. Basierend auf den Eingaben wird dann die entsprechende Nachricht erstellt. Um eine Nachricht zu kodieren, wird die Bibliothek `cantools` benutzt. Mit dieser Bibliothek können DBC-Dateien gelesen und Nachrichten erstellt werden. Mit einer solchen Datei kann eine bestimmte Nachricht mit ihren Signalen definiert werden. Es ist dafür notwendig, diese DBC-Datei zu verstehen.

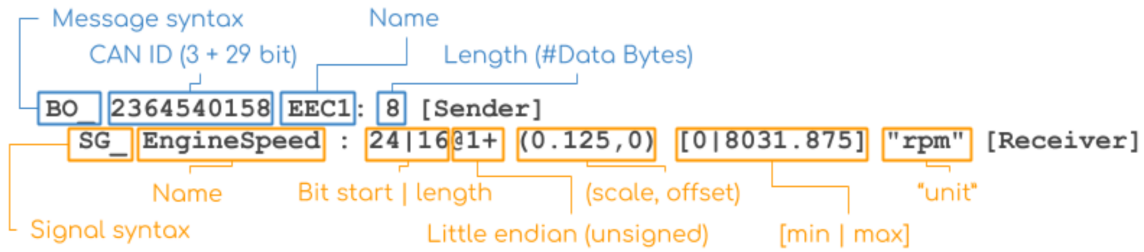


Abbildung 4.1: Auszug aus einer Beispiel DBC-Datei [Ele](letzter Zugriff: 18.02.2025)

In dieser sind einzelne Nachrichten aufgelistet. Jede Nachricht hat eine ID, eine Länge und Signale. Diese Signale haben eine Länge, einen Offset und einen Faktor. Mit diesen Informationen kann eine Nachricht erstellt werden. Im Anschluss an die Nachrichtentypen werden in der DBC-Datei für bestimmte Signale die möglichen Eingaben definiert. Das hilft bei der richtigen Wahl der Eingabe.
In der benötigten Nachricht ist eine Prüfsumme von 4 Bit notwendig. Jedoch gibt es zwei Methoden der Prüfsummenberechnung nach dem SAE J1939 Standard [Gmbb] (letzter Zugriff: 18.02.2025). Die richtige Methode für das System der Limanda wurde durch aufgezeichnete Nachrichten ermittelt. Am Ende wird die Prüfsumme der Nachricht hinzugefügt und die Nachricht wird an den Can-Bus gesendet.
Zusätzlich hilft das Programm `canReader.py` bei der Überwachung des Can-Bus. Es kann die Nachrichten auf dem Can-Bus lesen und in Echtzeit dekodieren. Dazu wurde wieder die Bibliothek `cantools` benutzt. Auch hier wird mit der gleichen DBC-Datei gearbeitet, da es sich um die gleichen Nachrichten handelt. In den dekodierten Nachrichten können die Signale und deren Werte gesehen werden. Dies hilft bei der Überwachung des Systems. In den aufgezeichneten Nachrichten wurde entdeckt, dass Nachrichten des Gashebels immer eine PGN von 0 haben, obwohl in der DBC-Datei eine andere PGN definiert ist. Das kann darauf zurückzuführen sein, dass es nicht unbedingt die richtige DBC-Datei ist. Es ist jedoch die einzige verfügbare DBC-Datei. Da nur die Gashebelposition mit der PGN gesendet wird, kann nach dieser PGN gefiltert werden. Die gefundenen Nachrichten werden dann dekodiert und spezifisch nach dem Signal für die Gashebelposition durchsucht. Wenn dieses Signal

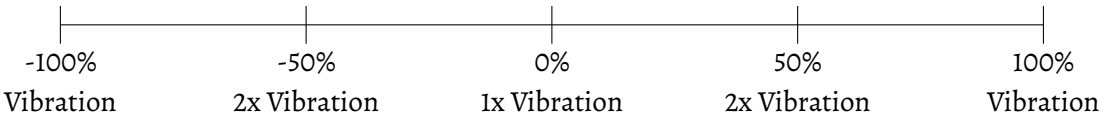
entdeckt wird, dann muss eine eigene Nachricht gesendet werden, um die wahren Eingaben zu verhindern. Dies passiert wieder durch `canInterpreter.py` nach einem über Pipes übermittelten Signal. Um hier zu verhindern, dass eigene Nachrichten vom eigenen System erkannt werden, wird ein eigener Identifier benutzt. Um diesen zu berechnen, wurde mit einem Port von `canboatjs` auf Python gearbeitet [Ben+24]. In diesem Programm kann eine bestehende CAN-ID in Quelle, Priorität und PGN aufgeteilt werden. Diese Werte können dann einzeln verändert werden. Mit diesen Werten kann dann eine neue CAN-ID berechnet werden. Nun können eigene Nachrichten anhand der CAN-ID erkannt werden.
Für die Gangschaltung musste eigene Nachricht in der DBC-Datei definiert werden. Das konnte anhand der Bedienungsanleitung der Motoren gemacht werden. Die Nachrichten wurden dann in `canInterpreter.py` erstellt und gesendet. Eine Schwierigkeit dabei hat sich ergeben, als die ID nicht als erweiterter J1939-Standard erkannt wurde. Dafür wurde eine Lösung gefunden, indem die ID mit der Hexadezimalen Zahl 80000000 addiert wurde. Die neu erzeugte ID wird nun ohne Probleme als erweiterte ID erkannt. Diese Lösung konnte nicht in einer offiziellen Dokumentation für den DBC-Standard gefunden werden. Es wurde jedoch in einem Forum gefunden, dass diese Lösung funktioniert [jul21] (letzter Zugriff: 04.03.2025). Laut der gleichen Quelle hängt der Grund dafür damit zusammen, dass ein Bit für die ID in dem Dateiformat zweckentfremdet für die Erkennung einer erweiterten ID benutzt wird. Allerdings kann dies nicht bestätigt werden, aber es scheint eine plausible Erklärung zu sein.
Da in der Nachricht für das Getriebe auch die aktuelle maximale zulässige Last gesendet wird, reicht es nicht aus, diese Nachricht nur bei einer Änderung des Gangs zu senden. Daher wird die Nachricht bei jeder Änderung der Gashebelposition gesendet. Damit hat die aktuelle maximale zulässige Last immer den derzeitigen Wert.

4.3.2 Serielle Schnittstelle

4.3.3 Eingabe-Interface

Durch die begrenzte Zeit dieser Arbeit, wurde die Rückmeldung mit Vibrationen im Xbox-Controller implementiert. Dabei wird bei Erreichen des Maximums oder Minimums der Gashebelposition eine Vibration ausgelöst. Dies ist eine einfache Methode, um dem Benutzer eine Rückmeldung zu geben. Auch bei der hälfte der Gashebelposition wird eine kurze und leichte Vibration ausgelöst. Das soll ermöglichen, dass es eine ungefähre Vorstellung der Gashebelposition gibt. Bei der Ruderposition wird eine Vibration ausgelöst, wenn die Ruderposition 100% erreicht hat in jeweils beide Richtungen erreicht hat. Bei der Mitte der Ruderposition wird eine kurze und leichte Vibration ausgelöst. Wenn die Ruderposition auf 50% in eine Richtung ist, werden zwei kurze und leichte Vibrationen ausgelöst. Dies soll die Bedienung erleichtern.

Die Ruderstellung wird durch den linken Stick des Controllers gesteuert. Wenn die Eingabe eine bestimmte Schwelle überschreitet, wird dies wie Folgt durch Vibrationen signalisiert:



Dabei ist die Schwelle so gewählt, dass die Vibrationen nicht zu oft ausgelöst werden. Es ist wichtig zu wissen, dass der Wert basierend auf der Eingabe des Joysticks kontinuierlich berechnet wird. Die Eingabe beträgt -1 bis 1. Dabei ist -1 die maximale Position nach links und 1 die maximale Position nach rechts. Damit die Ruderstellung nicht zu schnell verändert wird, benötigt es eine volle Eingabe von 1 oder -1 für 2 Sekunden, damit die Ruderstellung 100% erreicht. Entsprechend werden diese 100% langsamer erreicht, wenn die Eingabe geringer ist.

Kapitel 5

Sicherheit

In diesem Kapitel

5.1 Schwachstellen

Die durchgeführte Arbeit konnte in diesem Format nur durch einige Schwachstellen in der Kommunikation durchgeführt werden. Um einen solchen Angriff möglichst zu vermeiden, ist es wichtig, dass die Schwachstellen aufgedeckt und behoben werden. Dafür werden in dem folgenden Abschnitt die aufgedeckten Schwachstellen dieser Arbeit aufgeführt und diskutiert.

5.1.1 Schwachstellen am CAN-Bus

Der CAN-Bus ist eines der zentralen Systeme in modernen Fahrzeugen und Schiffen. So auch auf der Limanda. Nach dem CAN-Standard sind die Nachrichten auf dem Bus nicht verschlüsselt. Zusätzlich werden die Nachrichten nur auf ihre Richtigkeit überprüft, jedoch nicht auf ihre Authentizität. Das bedeutet, dass ein Angreifer Nachrichten auf den Bus senden kann, die von anderen Systemen als legitim angesehen werden.[Vos08] (Seiten 13-19) Durch das Bus-Netzwerk wird die einfache Integration weiterer Geräte in das System ermöglicht. Ein Angreifer kann so durch physischen Zugriff auf den Bus ein Gerät hinzufügen, welches im schlimmsten Fall die Kontrolle über das gesamte System

übernehmen kann. Die einzige Hürde, ein Gerät in ein CAN-Bus Netzwerk zu integrieren, ist das Beschaffen der richtigen DBC-Datei. Diese Datei wird häufig von den Herstellern der Geräte nicht öffentlich zur Verfügung gestellt. Es sind jedoch einige DBC-Dateien im Internet verfügbar, die von anderen Nutzern erstellt wurden. Durch das Ausprobieren von dem Dekodieren der CAN-Nachrichten mit verschiedenen DBC-Dateien, kann eine passende Datei gefunden werden.
In dieser Arbeit wurde zuerst ausgenutzt, dass die Nachrichten durch physischen Zugriff auf den Bus mitgelesen werden können. Die Nachrichten wurden aufgezeichnet und mit einer passenden DBC-Datei dekodiert. Nachdem die Nachrichten verstanden wurden, konnte eine Nachricht mit dem gleichen Syntax und eigenen Werten erstellt werden. Die aufgezeichnete Nachrichten-ID wurde dabei in der Quelle verändert. Die Priorität ist gleich geblieben, da es bereits die höchste Priorität war. In anderen Fällen ist es möglich, die Priorität einfach zu verändern. Mit der eigenen Nachrichten-ID konnten nun die echten Nachrichten von den eigenen unterschieden werden. Das ist wichtig, um echte Nachrichten mit den eigenen zu überschreiben. Alle Geräte in einem CAN-Bus sind gleichberechtigt[Vos08] (Seite 14). Dadurch wird die Überschreibung der älteren Nachrichten möglich.
Die vorher beschriebenen Schwachstellen sind allgemein für CAN-Bus Netzwerke. Wobei nicht alle eine Schwachstelle beschreiben, sondern Eigenschaften des Standards sind, welche zusammen mit den Schwachstellen ausgenutzt werden können.

5.1.2 Schwachstellen am J1939-Protokoll

Das J1939-Protokoll ist ein Protokoll, welches nicht für die Sicherheit, sondern für die Kommunikation zwischen den Steuergeräten entwickelt wurde. Es ist ein Protokoll, welches auf dem CAN-Bus aufbaut. Das Protokoll ist nicht verschlüsselt und die Nachrichten sind nicht authentifiziert. Allerdings hat das Protokoll auch spezifische Schwachstellen, welche aber nicht alle in dieser Arbeit ausgenutzt wurden.
Eine Schwachstelle ist, dass Nachrichten mit der niedrigsten ID die höchste Priorität haben. Zusätzlich ist es möglich, eine eigene Priorität so zu setzen, dass die Nachrichten mit der eigenen ID die höchste Priorität haben. Durch das Senden von vielen Nachrichten mit hoher Priorität kann ein Denial-of-Service-Angriff durchgeführt werden. Dabei wird der Bus so stark belastet, dass die echten Nachrichten nicht in einem angemessenen Zeitraum gesendet werden können. Jedes Gerät in einem J1939-Netzwerk benötigt eine eindeutige Adresse und einen einzigartigen Namen [Jun10]. Da ein Gerät vollständige Kontrolle über den eigenen Namen und die eigene Adresse hat, kann ein Angreifer sich als ein anderes Gerät ausgeben. Die Namensübernahme führt zunächst zu einem Konflikt, da zwei Geräte den gleichen Namen haben. Diesen Konflikt gewinnt das Gerät, welches die niedrigere Adresse hat. Daher kann der Name recht einfach übernommen werden. Dadurch kann ein originelles Gerät aus dem Netzwerk ausgeschlossen werden. Mit der übernommenen Adresse kann der Angreifer nun

Nachrichten senden, die von anderen Geräten als legitim angesehen werden. Die einzige Sicherheitsmaßnahme, die das J1939-Protokoll bietet, ist eine Zuweisungstabelle für die Adressen und Namen. Allerdings bietet diese bei der Namensübernahme keinen Schutz, wenn der Angreifer auch die Adresse übernimmt. Diese Schwachstelle wurde in dieser Arbeit nicht ausgenutzt, da es zu unvorhersehbaren Folgen hätte führen können. Das spricht für die Schwere der Schwachstelle.
Eine weitere Möglichkeit den Bus zu stören, ist die globale Anfrage nach PGNs. Bei einer solchen Anfrage sollen alle Geräte mit der eigenen PGN antworten. Die Spezifikation rät dazu, maximal 3 Anfragen pro Sekunde für eine Parametergruppe zu senden. Es gibt aber keine feste obere Grenze für die Anzahl der Anfragen. Daher gibt es auch keine Gegenmaßnahmen bei zu vielen Anfragen. Ein Angreifer kann also durch das Senden von vielen Anfragen alle Geräte auf dem Bus dazu bringen, zu antworten. Damit ist ein Distributed-Denial-of-Service-Angriff (DDoS) möglich, weil die Überlastung von mehreren Geräten ausgeht.
[MG18]

5.2 Schutzmaßnahmen

Schutzmaßnahmen sind notwendig, um die Schwachstellen zu schließen und Angriffe zu verhindern. Eine Schwierigkeit bei dem Zugriff auf den CAN-Bus ist, dass dieser physisch zugänglich sein muss. Dies ist auch auf der Limanda der Fall. Wenn man allerdings unbemerkten Zugriff erhält, kann das Rogue Device versteckt in dem System integriert werden. Das stellt die einzige Schutzmaßnahme dar, die in dieser Arbeit umgangen wurde. Jedoch wird der Zugriff nur durch eine Abdeckung zu der Technik des Schiffes verhindert. Mit dem richtigen Werkzeug kann diese Abdeckung einfach entfernt werden. Eine weitere Hürde ist, dass die richtige Baudrate für den CAN-Bus des Schiffes gewählt werden muss. Ohne vorherige Kenntnisse muss dabei ausprobiert werden, welche Baudrate die richtige ist. Das stellt aber keine Schutzmaßnahme dar, sondern einfach eine wichtige Eigenschaft des CAN-Standards.

Was sind weitere Möglichkeiten?

- Intrusion Detection Systeme (IDS) für den CAN-Bus
- Dabei muss der CAN-Standard in keiner Form modifiziert werden
- auch keine Implementation in jedem Gerät eines Netzwerkes notwendig

[GGT16]

- zwei Arten von IDS für den CAN-Bus
- Signatur/Regel basiertes Detektionssystem
- dabei sind werden gewisse Kombinationen oder Seqeunzen von Nachrichten von bekannten Angriffen erkannt
- Anomalie basiertes Detektionssystem
- dabei wird das normale Verhalten des Systems überwacht
- Abweichungen von diesem Verhalten werden als Angriff erkannt, eigene Lösung für Schiffe notwendig, da unterschiedliches Verhalten
- Studie dazu [Dav+24]
- Anomalie basiertes System weniger wartungsintensiv, da keine ständige Aktualisierung der Regeln notwendig um neue Angriffe zu erkennen
- Anomalie basiertes System auch besser für unbekannte Angriffe, aber auch mehr Fehlalarme
- Regeln basiertes System sehr zuverlässig für bekannte Angriffe

[HKD09]

- probleme mit J1939 vorallem durch das Fehlen kryptographischer Authentifizierung
- aufgrund der verschiedenen Hersteller ist es schwierig, eine einheitliche Lösung zu finden
- daher public key infrastructure (PKI) notwendig

[MG18] Feasability of PKI in J1939:

- notwendigkeit von PKI in Fahrzeug-Fahrzeug Netzwerken
- Support für PKI in AUTOSAR (Automotive Open System Architecture) crypographic specification
- viele Open-Source libraries mit PKI support
- zumindest für Autoindustrie keine ungewöhnliche Forderung, schon einige Implementierungen mit PKI

[MG18]

5.3 Relevanz für andere Schiffe

Gibt es solche Angriffsmöglichkeiten auch auf anderen Schiffen?

- Can-Bus häufig genutzt, häufig nicht authentifizierte nachrichten
- besonders auf größeren Schiffen ist das Ruder nicht mehr mechanisch, sondern elektronisch
- Angriff auf das Steuerungssystem könnte katastrophale Folgen haben

Kapitel 6

Abschließende Betrachtung

In diesem Kapitel

6.1 Fazit

Was wurde geschafft? Was kann damit ausgesagt werden?

6.2 Ausblick

Wo kann noch weiter geforscht werden? Was wurde nicht geschafft?

Kapitel VII

Anhang

In diesem Kapitel

VII.1 Quellcode

VII.2 Schaltpläne

Abbildungen

Literatur

[Ben+24] Scott Bender, Kees Verruijt, Jouni Hartikainen und Teppo Kurki. *Canboatjs*. GitHub. Aug. 2024. URL: https://github.com/canboat/canboatjs/releases/tag/v2.10.0 (siehe S. 18).

[can] cantools. *cantools*. GitHub. URL: https://github.com/cantools/cantools/releases (siehe S. 7).

[Cho+21] Wonsuk Choi, Seyoung Lee, Kyungho Joo, Hyo Jin Jo und Dong Hoon Lee. „An Enhanced Method for Reverse Engineering CAN Data Payload“. In: *IEEE Transactions on Vehicular Technology* 70.4 (2021), S. 3371–3381. DOI: 10.1109/TVT.2021.3063261 (siehe S. 8).

[Dav+24] Marvin Davieds, Thomas Mundt, Richard Dabels, Noah Garkisch und MA Al-Mamun. „Anomaly Detection in Uncharted Ship-Borne Field Buses: A case study“. In: *2024 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE. 2024, S. 1–9 (siehe S. 23).

[Ele] CSS Electronics. *CAN DBC File Explained - A Simple Intro*. Website. URL: https://www.csselectronics.com/pages/can-dbc-file-database-intro (siehe S. 17).

[Fal] Martin Falch. *can_decoder*. GitHub. URL: https://github.com/CSS-Electronics/can_decoder (siehe S. 8).

[Fys] Fysetc. *UCAN*. GitHub. URL: https://github.com/FYSETC/UCAN/ (siehe S. 7).

[GGT16] Mabrouka Gmiden, Mohamed Hedi Gmiden und Hafedh Trabelsi. „An intrusion detection method for securing in-vehicle CAN bus“. In: *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. 2016, S. 176–180. DOI: 10.1109/STA.2016.7952095 (siehe S. 22).

[Gmba] Vector Informatik GmbH. *CANoe*. Website. URL: https://www.vector.com/de/de/produkte/produkte-a-z/software/canoe/#c386458 (siehe S. 7).

[Gmbb] Vector Informatik GmbH. *SAE1939*. Website. URL: https://www.vector.com/de/de/know-how/protokolle/sae-j1939/# (siehe S. 5, 17).

[HKD09] Tobias Hoppe, Stefan Kiltz und Jana Dittmann. „Applying intrusion detection to automotive it-early insights and remaining challenges“. In: *Journal of Information Assurance and Security (JIAS)* 4.6 (2009), S. 226–235 (siehe S. 23).

[jul21] juleq. *cantools Issue*. GitHub. März 2021. URL: https://github.com/cantools/cantools/issues/301#issuecomment-808726550 (siehe S. 18).

[Jun10] Markus Junger. *Introduction to J1939*. Techn. Ber. Vector Informatik GmbH., 2010. URL: https://www.vector.com/int/en/download/introduction-to-j1939/ (siehe S. 21).

[Kid] Collin Kidder. *SavvyCAN*. GitHub. URL: https://savvycan.com/index.php (siehe S. 8).

[KuK] Matthew KuKanich. *CAN Commander*. GitHub. URL: https://github.com/MatthewKuKanich/CAN_Commander (siehe S. 7).

[lin] linux-can. *can-utils*. GitHub. URL: https://github.com/linux-can/can-utils (siehe S. 8).

[McR95] Duane T McRuer. *Pilot-induced oscillations and human dynamic behavior*. Techn. Ber. 1995. URL: https://ntrs.nasa.gov/citations/19960020960 (siehe S. 14).

[MG18] Pal-Stefan Murvay und Bogdan Groza. „Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol“. In: *IEEE Transactions on Vehicular Technology* 67.5 (2018), S. 4325–4339. DOI: 10.1109/TVT.2018.2795384 (siehe S. 5, 22, 23).

[Mic] Microsoft. *Xbox One Wireless Controller*. Xbox Support. URL: https://support.xbox.com/de-DE/help/hardware-network/controller/xbox-one-wireless-controller (siehe S. 11).

[PL19] Sudarshan Pant und Sangdon Lee. „Design and Implementation of a CAN Data Analysis Test Bench based on Raspberry Pi“. In: *Journal of Multimedia Information System* 6.4 (Dez. 2019), S. 239–244. ISSN: 2383-7632. DOI: 10.33851/jmis.2019.6.4.239 (siehe S. 6).

[Sca+08] Karen Scarfone, Murugiah Souppaya, Amanda Cody und Angela Orebaugh. „Technical guide to information security testing and assessment“. In: *NIST Special Publication* 800.115 (2008), S. 2–25. URL: https://git.hsbp.org/six/PTD/raw/commit/9e642f4aa921f2bc83b012326ce0ed23c0af4e53/methodology_documents/NIST_SP800-115.pdf (siehe S. 6).

[SKJ16] A. A. Salunkhe, Pravin P Kamble und Rohit Jadhav. „Design and implementation of CAN bus protocol for monitoring vehicle parameters“. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, Mai 2016, S. 301–304. DOI: 10.1109/rteict.2016.7807831 (siehe S. 6).

[VJ15] Aditya Venkataraman und Kishore Kumar Jagadeesha. „Evaluation of inter-process communication mechanisms“. In: *Architecture* 86.64 (2015). URL: http://pages.cs.wisc.edu/~adityav/Evaluation_of_Inter_Process_Communication_Mechanisms.pdf (siehe S. 16).

[Vos08] Wilfried Voss. *A comprehensible guide to controller area network*. Copperhill Media, 2008. ISBN: 9780976511601 (siehe S. 3, 4, 20, 21).

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Die Arbeit ist noch nicht veröffentlicht und ist in ähnlicher oder gleicher Weise noch nicht als Prüfungsleistung zur Anerkennung oder Bewertung vorgelegt worden.

Rostock, 5. März 2025