Universität Rostock

Traditio et Innovatio

Sicherheitsanalyse durch Entwicklung eines Rogue Device zur Echtzeitmanipulation maritimer Steuerungssysteme

Jakob Engelbert Tomahogh

Betreuer: *M.Sc. Marvin Davieds*
Zweitgutachter: *Prof. Dr. rer. nat. Clemens H. Cap*

14.02.2025

Motivation

- Sicherheit wurde in maritimen Systemen vernachlässigt
- Kommunikationssysteme sind anfällig für Angriffe
- Angriff auf Steuerungssysteme könnte katastrophale Folgen haben
- physischer Zugriff bei Passagierschiffen möglich

Zielsetzung

- Steuerung eines Schiffes durch ein Spielecontroller
- Rogue Device als Schnittstelle zwischen Controller und Schiff
- Unbemerkte Manipulation der Steuerung

Zielsetzung

- Machbarkeit eines solchen Angriffs soll gezeigt werden
- Aufmerksamkeit auf Sicherheitslücken in maritimen Systemen lenken
- Sicherheitslücken sollen durch Steuerung mit Spielecontroller veranschaulicht werden
- Betrachtung möglicher Gegenmaßnahmen

Schiffstechnik

Quelle: https://www.inf.uni-rostock.de/mts/forschungsschiff/forschungsschiff-limanda-1/

Fig. 1: Forschungsschiff Limanda

- zweimotoriger Katermaran
- Länge: 15,73m
- Breite: 6,16m

Schiffstechnik

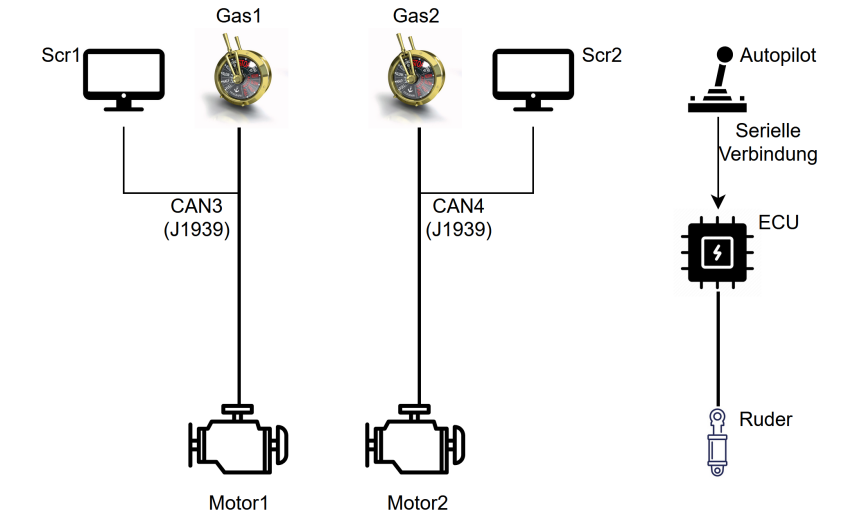


Fig. 2: Vereinfachter Aufbau der Systeme

CAN-Bus

- serielle Netzwerktechnologie, bei dem mehrere Geräte miteinander kommunizieren können
- ermöglicht effiziente Kommunikation zwischen Steuergeräten
- alle Geräte sind gleichberechtigt
- Nachrichten werden nach Broadcast-Prinzip übertragen
- Kommunikation auf dem CAN-Bus ist unverschlüsselt

CAN-Bus Nachricht

Quelle: A Comprehensible Guide to Controller Area Network (Wilfried Voss)

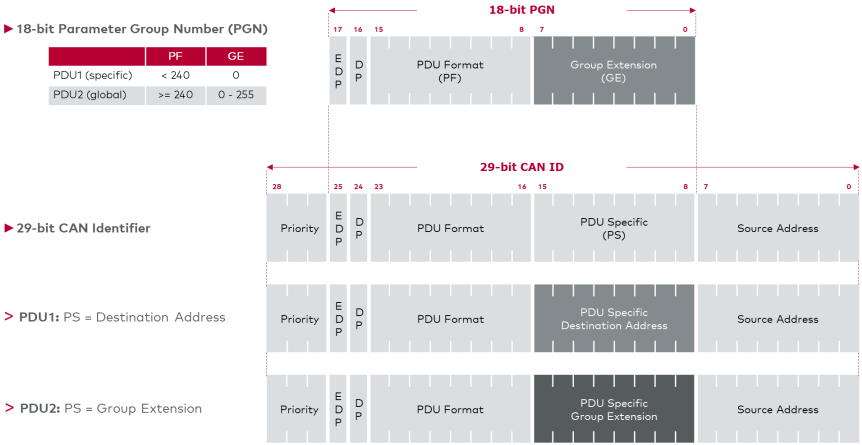
Fig. 3: Aufbau einer CAN-Bus Nachricht

- **SOF**: Start of Frame
- **Arbitration Field**: Nachrichten-ID und Remote Transmission Request
- **Control Field**: Datenlänge
- **Data Field**: Nutzdaten
- **CRC Field**: Prüfsumme
- **EOF**: End of Frame
- **IFS**: Interframe Space, Pause zwischen Nachrichten

J1939

- Standard für die Kommunikation auf dem CAN-Bus
- Nutzt 29 Bit Extended CAN Identifier
- ermöglicht Knotenadressierung

J1939



Quelle: https://www.vector.com/de/de/know-how/protokolle/sae-j1939/

Fig. 4: J1939 Nachrichten-ID

Konzept

- Xbox Controller als Eingabegerät
- Raspberry Pi 5 als Rogue Device
- Anbindung des Raspberry Pi an den CAN-Bus
- Übersetzung der Controllereingaben in CAN-Bus Nachrichten auf Rogue Device

Konzept

- Unterbindung der originalen Steuerung
- Reaktion auf Nachrichten der Gashebel
- Rudersteuerung über Autopiloten

derzeitiger Stand

- Raspberry Pi mit Raspberry Pi OS
- Programmierung in Python
- Bedienungskonzept des Xbox-Controllers

derzeitiger Stand

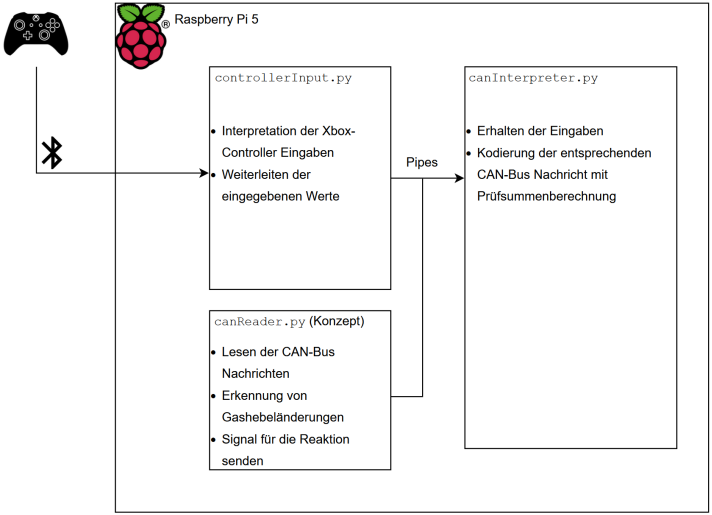
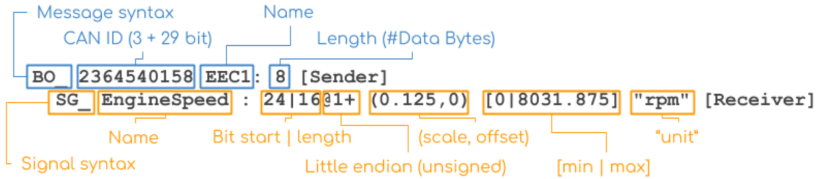


Fig. 5: Programmstruktur auf dem Raspberry Pi

derzeitiger Stand

- Aufbau eines CAN-Bus Netzwerks mit UCAN (USB-zu-CAN Adapter)
- Kodieren der Eingaben in CAN-Bus Nachrichten
  - cantools
  - DBC-Datei

DBC-Datei



Quelle: https://www.csselectronics.com/pages/can-dbc-file-database-intro

Fig. 6: Auszug einer Beispiel DBC-Datei

derzeitiger Stand

- Anhand der DBC-Datei einzelne Signale mit validen Werten erstellt
- Frame-ID einer aufgezeichneten Nachricht entnommen
- Prüfsumme mit Nachrichten-ID, ersten 7 Bytes und Nachrichtenzähler berechnet
- Daraus CAN-Bus Nachricht erstellt

Ausblick

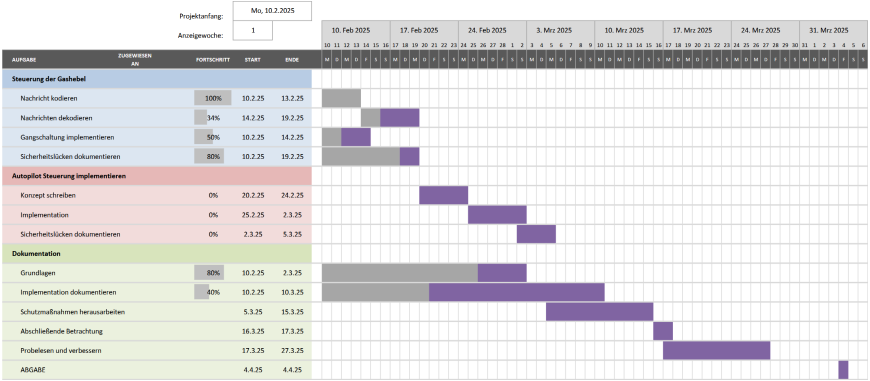


Fig. 7: Zeitplan

Ausblick

Steuerung der Gashebel			
Nachricht kodieren	100%	10.2.25	13.2.25
Nachrichten dekodieren	34%	14.2.25	19.2.25
Gangschaltung implementieren	50%	10.2.25	14.2.25
Sicherheitslücken dokumentieren	80%	10.2.25	19.2.25

Fig. 8: Auszug aus dem Zeitplan

Ausblick

Autopilot Steuerung implementieren			
Konzept schreiben	0%	20.2.25	24.2.25
Implementation	0%	25.2.25	2.3.25
Sicherheitslücken dokumentieren	0%	2.3.25	5.3.25

Fig. 9: Auszug aus dem Zeitplan

Ausblick

Dokumentation			
Grundlagen	80%	10.2.25	2.3.25
Implementation dokumentieren	40%	10.2.25	10.3.25
Schutzmaßnahmen herausarbeiten		5.3.25	15.3.25
Abschließende Betrachtung		16.3.25	17.3.25
Probelesen und verbessern		17.3.25	27.3.25
ABGABE		4.4.25	4.4.25

Fig. 10: Auszug aus dem Zeitplan