AEVisionLab: Manipulating In-vehicle Ethernet Networks with All-round Vision

ANTHONY YEO KEE TECK, Singapore University of Technology and Design, Singapore
MATHEUS E. GARBELINI, Nanyang Technological University, Singapore
SAI SATHIESH RAJAN, Singapore University of Technology and Design, Singapore
JIANYING ZHOU, Singapore University of Technology and Design, Singapore
SUDIPTA CHATTOPADHYAY, University of Missouri-Kansas City, United States

With the increasing adoption of Advanced Driver Assistance Systems (ADAS) in modern cars, the use of vision systems for autonomous vehicles, driving assistance, and in-vehicle entertainment has introduced new risks and attack vectors to existing In-Vehicle Networks (IVNs), thus bringing considerable concerns to the automotive cybersecurity space. Prior works have focused on analyzing functional or partial security aspects of vision systems during ADAS simulation using specialized Automotive Ethernet (AE) equipment or requiring expensive vehicle-in-the-loop setups. These approaches are either inaccessible to independent security researchers or do not offer comprehensive insights to help researchers understand the practical implications of attacks in a realistic car employing Automotive Ethernet IVNs for vision-related use cases. To this end, we build an embedded system, AEVisionLab, using automotive commercial off-the-shelf (COTS) electronic control units (ECUs) and a workstation. AEVisionLab is a standalone, reusable, and comprehensive test platform for Automotive Ethernet (AE) networks that allow manipulation of messages over a realistic All-round Vision IVN of a BMW G20 via 100BASE-T1 Ethernet bridging. AEVisionLab allows replication of driving test scenarios directly with COTS ECUs and collection of key network performance metrics, facilitating the design, evaluation, and impact analysis of concrete attacks in the laboratory. We demonstrate the capability of AEVisionLab by designing and evaluating concrete attacks scenarios including eavesdropping and hijacking of SOME/IP services, manipulation and delaying video feed, among others. We envision AEVisionLab as a flexible platform for designing and evaluating both attack and mitigation techniques (e.g., intrusion detection) on AE network, which can be easily extended to support other automotive ECUs, machine learning models for ADAS, or sensors for assisted driving.

Additional Key Words and Phrases: Automotive, Testbed, ECU, Fuzzing, Attack, Ethernet

ACM Reference Format:
Anthony Yeo Kee Teck, Matheus E. Garbelini, Sai Sathiesh Rajan, Jianying Zhou, and Sudipta Chattopadhyay. 2018. AEVisionLab: Manipulating In-vehicle Ethernet Networks with All-round Vision. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 38 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The Automotive Ethernet (AE) protocol is expanding due to its need for high-speed data transmission in applications like real-time sensor communication, Advanced Driver Assistance Systems (ADAS), and in-vehicle entertainment.

Authors' Contact Information: Anthony Yeo Kee Teck, anthony_yeo@sutd.edu.sg, Singapore University of Technology and Design, Singapore, Singapore, Singapore; Matheus E. Garbelini, Nanyang Technological University, Singapore, Singapore, Singapore, matheus.garbelini@ntu.edu.sg; Sai Sathiesh Rajan, Singapore University of Technology and Design, Singapore, Singapore, Singapore, sai@rajan.cc; Jianying Zhou, Singapore University of Technology and Design, Singapore, Singapore, Singapore, jianying_zhou@sutd.edu.sg; Sudipta Chattopadhyay, University of Missouri-Kansas City , Kansas City, Missouri, United States, schattopadhyay@umkc.edu.

Manuscript submitted to ACM

Furthermore, AE can provide quick and secure remote services to support over-the-air software upgrades and real-time troubleshooting. AE is also essential for software-defined vehicles (SDVs), where a central control unit is connected by Ethernet links to multiple controllers for specific functions such as Advanced Driver Assistance Systems (ADAS) and infotainment. Considering the importance of AE protocol in connected cars, security testing of cars employing AE protocol is crucial.

In this paper, we conceptualize, design and evaluate an embedded system, AEVisionLab, using automotive commercial off-the-shelf (COTS) electronic control units (ECUs) and a workstation. AEVisionLab is a comprehensive test platform targeted for security testing of AE protocols. In contrast to other automotive test platforms, the distinction of AEVisionLab is illustrated in Figure 1. Figure 1(a) represents a test platform that performs Vehicle-in-the-Loop (VIL) testing for ADAS system. However, such test platforms are expensive as they require the use of mechanical equipment attached to the car such as a Roller Bench for the car wheels. Moreover, this setup is mostly performed by the car manufacturer who has expertise of the behaviour of the car, In-Vehicle-Network and signals to be read or written during road scenario simulation via a Hardware-in-the-Loop (HIL) Rack. Similarly, Camera-in-the-loop (Figure 1(b)) is specifically designed to test the ADAS system without the need of the entire car, however, it still requires knowledge of the entire ADAS system, which is proprietary to the manufacturer and not accessible to cybersecurity experts. These test platforms are considered to connect indirectly to the ECUs since the ADAS or ECU testing is performed end-to-end and does not allow controlling every connection between all ECUs (i.e., direct connection). In contrast, Figure 1(c) illustrates hybrid platforms that either simulate IVNs or operate real COTS ECUs in conjunction with simulation. However, such works do not consider protocols or ECUs that are employed in ADAS and All-round Vision systems of autonomous vehicles (AVs). To bridge this gap, AEVisionLab platform offers extensive and *direct* control of COTS Cameras, Camera ECUs, Ethernet switches, thus making AV systems more accessible to cybersecurity research and providing sufficient control for comprehensive security testing. Moreover, this is accomplished without incurring significant cost as in the platforms employed in Figures 1(a)-(b).

While designing AEVisionLab, we face several technical challenges. Firstly, one of the key design considerations for AEVisionLab is to maintain modularity and extensibility of the test platform. To this end, we carefully decomposed the software and hardware components of AEVisionLab platform. This allowed us to extend the test platform with different ECUs without affecting the software part. Likewise, AEVisionLab facilitates development of advanced attacks, monitoring and defense techniques without any change in the hardware. Secondly, since AEVisionLab employs COTS ECUs, we had to reverse engineer to understand unknown message and protocols, which, in turn allowed us to investigate and manipulate the flow of messages. Thirdly, we employed various techniques to collect, manipulate and inject videos of real road scene efficiently. *To the best of our knowledge, AEVisionLab is the first test platform to facilitate arbitrary manipulation of in-vehicle AE network messages including video and using COTS ECUs.*

We present the following contributions in AEVisionLab:

(1) We design and implement a realistic test platform targeted towards security analysis of automotive ethernet (AE) IVNs and protocols using low-cost, yet Commercial-off-the-shelf (COTS) ECUs.
(2) We design and implement AEVisionLab software and hardware platform in a fashion to perform arbitrary attacks in the IVN. This is accomplished via (i) active tap of AE messages for sniffing, frame injection and fuzzing; and (ii) video feed simulation and manipulation.
(3) We propose a methodology to perform reverse engineering and fingerprinting of proprietary AE protocols. This is instantiated to extract video feed from a real all-round vision system IVN of a BMW (G20) car model.

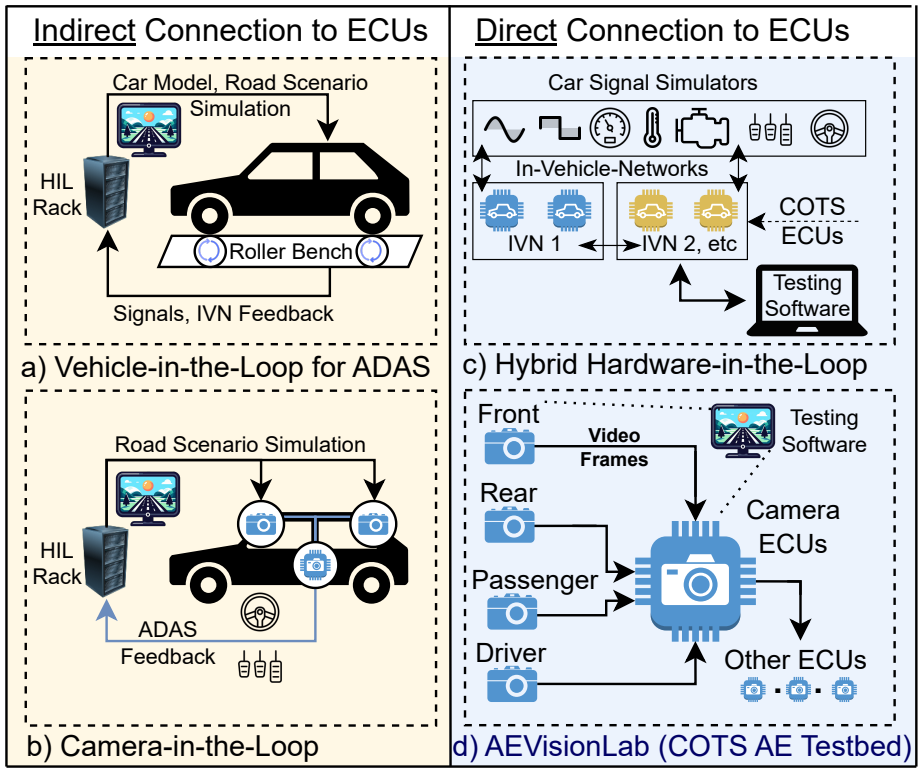


Fig. 1. Automotive test platforms

Subsequently, we show that AEVisionLab can be used to simulate or modify video stream of road scenes within BMW G20 IVN in real-time.

(4) Our security analysis of BMW G20 IVN reveals vulnerabilities in the all-round vision system. This is then exploited by five attacks including eavesdroping and hijacking of IVN services, delay of camera feed and ECU identity forgery. These attacks leverage AEVisionLab platform to modify the flow of messages encapsulating known AE protocols (e.g., SOME/IP, SOME/IP-SD) as well as a proprietary BMW UDP protocol.

(5) We extensively evaluate the functionality and overall impact of AEVisionLab in the IVN by measuring key metrics such as AE message modification latency. We also measure the image similarity between the original and the altered video feed. The results reveal that AEVisionLab can perform analysis and attacks of the IVN with negligible impact in its functionality and performance.

The remainder of the paper is organized as follows. We discuss the motivation behind AEVisionLab in Section 2. After providing a brief overview of AEVisionLab architecture and the requirements in Section 3, we present the design and implementation details of AEVisionLab in Section 4. In Section 5, we present a comprehensive evaluation of AEVisionLab including its capabilities and multiple case studies. After discussing related work in Section 6, we present the future outlook and potential usage of AEVisionLab in security testing and evaluation in Section 7. We conclude the paper in Section 8.

2 Motivation

AEVisionLab uses general communication equipment for AE networks without the need for specialized car equipment. It does not need an actual car or driving, and uses real ECUs, including the cameras, are connected in an AE network. Therefore, AEVisionLab can modify any AE messages from a connected ECU and conduct attacks in a laboratory. Concretely, Table 1 evaluates the different aspects of existing test platforms with respect to AEVisionLab.

Test Platform	No specialised car equipment	Car free	AE message			No ECU simulation	No driving	Laboratory Test		
			Sniff	Inject	Bridging			Real camera	IVN attack	Support AE
Keysight SA8710A (VIL)	❍	❍	●	●	❍	●	◗	●	❍	●
KT-CAMULATOR (CIL)	❍	●	❍	❍	❍	●	●	❍	❍	●
AutoCrypt CSTP (HIL)	❍	●	❍	❍	❍	❍	●	❍	●	❍
VitroBench	❍	●	❍	❍	❍	●	●	❍	●	❍
AEVisionLab	●	●	●	●	●	●	●	●	●	●

Table 1. AEVisionLab with respect to similar works. ●: supported, ◗: partially supported, ❍: generally not supported, VIL: Vehicle-in-the-loop, CIL: Camera-in-the-loop, HIL: Hybrid Hardware-in-the-loop.

Keysight SA8710A[1] is a comprehensive cybersecurity test platform and is typically used for the entire car test or message penetration test. Although it can test subcomponents, its main purpose is to validate the end-to-end systems against known cyberattacks. Only trained personnel are able to operate this high-performance system, and it generally needs a car driving on a Roller Bench in a workshop environment to test different scenarios. In contrast, the setup of AEVisionLab uses COTS ECUs in a laboratory and it involves minimal cost (8000 USD). AEVisionLab can tap any ECU to modify the message in the IVN as a bridge connection or to simulate an internal attack.

Konrad Technologies (KT) has developed an ADAS Sensor fusion test platform for Radar, Camera, Ultrasonic, and Lidar sensors. Its KT-CAMULATOR [2] is a virtual simulator that emulates the cameras of a car to an ECU. Their purpose is to test the ADAS function by injecting the camera's video. By itself, it does not monitor the messages from the AE network. However, AEVisionLab can manipulate the messages of the ECUs, including the actual video of the cameras.

AutoCrypt CSTP [3] has three products: Compliance, Fuzzer, and Functional Tester. Their platform can test a car's single or multiple ECUs and with connected hybrid hardware or software simulation. The products support the CAN protocol, but are limited in extended support for AE. Furthermore, there are limited HIL test platforms available in the market for AE research. AEVisionLab fills the research gap by connecting all required ECUs to an AE network.

Finally, VitroBench [29] is similar to AEVisionLab as a research test platform, except that VitroBench is developed to manipulate broadcast messages of CAN networks. It is unable to handle high-speed point-to-point AE communication links. Moreover, VitroBench leverages Python programming, which is too slow for the high-speed AE video stream, whereas AEVisionLab uses C programming for the high speed AE messages.

3 Test Platform Overview

The automotive industry is rapidly evolving, particularly in using AE communication protocol in the realm of Advanced Driver Assistance Systems (ADAS) and infotainment technologies. There is a growing demand to evaluate the security of these systems. We have developed the AEVisionLab test platform, which is specifically designed to accommodate the bandwidth requirements of AE communication protocol.

One application of ADAS systems is the all-round vision camera system, which enhances a vehicle's awareness of its surroundings and contributes to various safety features. For this test platform, we target an all-round vision camera

[1] https://www.keysight.com/sg/en/product/SA8710A/automotive-cybersecurity-penetration-test-platform.html
[2] https://www.konrad-technologies.com/products/adas/camulator-camera-direct-injection
[3] https://autocrypt.io/products/cstp/

system, and we have selected real car Electronic Control Units (ECUs) such as the camera ECU, four cameras, and interfacing ECUs that govern the functionality of such camera systems. These ECUs are integrated into the platform, which serves as a comprehensive testing environment.

The study focuses on the AE protocol, examining its behavior and potential security vulnerabilities by following several key aspects:

(1) **Observation of AE Messages:** AE messages are analyzed to gain insights into the behavior of the four cameras and the camera ECU. We aim to decipher patterns, identify functionalities, and understand the interactions between different components of the system.
(2) **Fingerprinting:** Our objective is to establish a fingerprint of the system, which includes gathering information such as the ECU's IP address, decoded message protocols, and timing characteristics. This fingerprint aids in subsequent analyses and testing procedures.
(3) **Reverse Engineering:** We decipher unknown protocols encountered within the AE communication network. By deconstructing and analyzing the structure of these protocols, we gain an understanding of the decoding of some unknown messages.
(4) **Execution of Fuzzing and Attacks:** We test the security posture of the AE communication links via random fuzzing and various targeted attacks. These attacks include simulated scenarios aimed at disrupting communication or exploiting vulnerabilities in the system.

In the following, we will first illustrate the overall architecture of AEVisionLab and subsequently, discuss the requirements of our design.

3.1 AEVisionLab architecture

The AEVisionLab platform consists of an automotive testbed, communication interfaces, and software components (see Figure 2). For the automotive testbed, we study an all-round vision camera testbed. The communication interfaces intercept the AE links between the ECUs and send the intercepted messages to the workstation, to serve two functions:

(1) Message sniffing via Capture Module, and
(2) Message injection and modification via Bridging

The workstation (see Figure 2) provides traffic control and analysis, enabling us to monitor and assess the security of the ADAS system under various conditions. In addition to hardware components, the software components are programs and applications that are tailored to support security testing and analysis tasks. Concretely, using such software components, we *monitor*, *inject*, and *modify* messages. Additionally, we also analyze the behavior of the ADAS system during testing. Message injection and modification are supported by building a bridge from an ECU to the workstation and back to the other connecting ECU.

Finally, the platform has an external screen that playbacks videos of road scenes. The ADAS camera can then capture the simulated road scene. This functionality enables us to modify or analyze the video stream messages and evaluate the security of the camera system in different scenarios.

3.2 AEVisionLab design requirements

The key consideration in designing the AEVisionLab platform is to make it open and adaptable to various car architectures. Concretely, such a platform must be able to monitor all selected AE communication links, repeat test cases, and record results for reproduction and analysis. In addition, the platform can target arbitrary ECUs, enabling

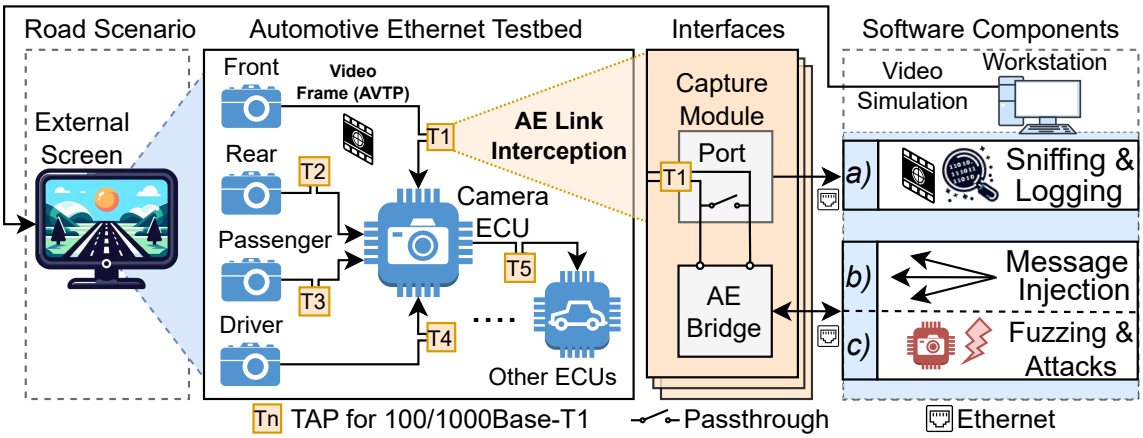


Fig. 2. Architecture of AEVisionLab Test Platform.

message injection and modification for behavior fingerprinting or fuzzing. In the following, we first discuss the attack model targeted in AEVisionLab. Subsequently, we outline the operational requirements to realize the targeted attack model.

Attack Model. To perform any attack, AEVisionLab considers a Dolev-Yao adversary (i.e., attacker), which is able to eavesdrop, modify, inject or drop legitimate messages between a pair of automotive Ethernet ECUs. Such adversary resides within a compromised ECU or is inserted (e.g., bridged) in the automotive Ethernet link, hence acting as a Man-in-the-Middle attacker. For both cases, we assume an already compromised component by means of physically bridging an external hardware to the automotive Ethernet link such as bridge or by means of exploiting the firmware of the target ECU (e.g., via BLE vulnerabilities in the BDC [28]). In practical terms, a *Compromised ECU* can perform malicious actions in all Ethernet links connected to it, while a *Compromised Link* can only affect the pair of ECUs connected to such link, therefore requiring multiple taps to compromise the link of all cameras in the All-round vision IVN. Nonetheless, the actions carried by the Dolev-Yao adversary are the same in both scenarios. This is because we assume that the adversary has no knowledge of any encryption during AE communication and hence she can only modify plain-text messages. From such starting point, the adversary can choose which component to affect in the IVN. For example, Figure 3 exemplifies attack surfaces where an adversary can be introduced within the All-round vision IVN (see Table 2 for ECU details). Concretely, messages originating from a compromised *Rcam* (or its link to the TSRVC) can affect other ECUs in the path of processing or forwarding video frames such as *TSRVC*, *BDC* and *HU*.

To showcase the impact to multiple ECUs of the All-round vision IVN, we perform practical Man-in-the-Middle attacks by individually compromising different links (i.e., bridging) of the IVN as illustrated in Figure 3. The impact of each attack is discussed in our evaluation (Section 5).

Requirements for Automotive Ethernet Testbed. The ECUs of the testbed should include modules for camera control, cameras, and relevant interfaces to form a functional all-round vision camera system. Such modules are required to create meaningful test scenarios. Additionally, there should be provisions to intercept camera video streams and control messages. Moreover, the system should allow for disconnecting targeted ECUs to bypass messages via the workstation

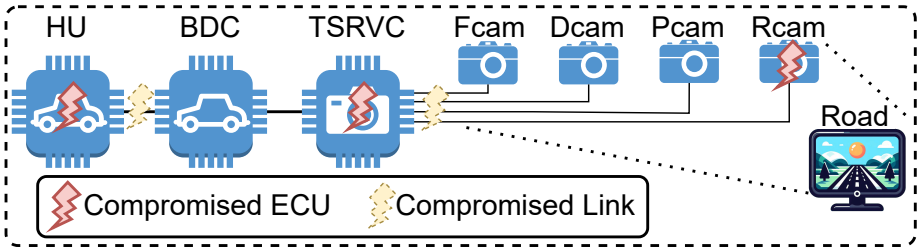


Fig. 3. A Dolev-Yao adversary can eavesdrop or manipulate within the All-round vision system.

and resend them, possibly after modifications, to other connected ECUs. Finally, the test platform should be versatile enough to replace the testbed for different car models.

Requirements for Message Sniffing. The Capture Module must be able to tap the AE links and sniff the messages from the ECUs of the test platform. The common protocols used in the current car's architecture [14] are 100BASE-T1 and 1000BASE-T1 [23]. We specify our requirements for the Capture Module based on these two protocols:

- The data rate is 100Mbit/s or 1000Mbit/s.
- The tapping of the link should have no impact on the functionality of the automotive testbed.
- At least six AE links can be supported simultaneously with a common data rate.
- The connected links are inter-changeable from the testbed's ECUs and each link can be configured independently.
- The messages of the tapped links are forwarded to the workstation for real-time monitoring and logging.

Due to the limited number of links in most Capture Modules, the sniffing should be extensible by combining several Capture Modules for more AE links.

Requirements for Message Injection and Modification. The system should allow AE link interception for message injection and modification. This functionality is crucial for our test platform, as we target security testing and analysis. To accomplish this, an ECU can be isolated by disconnecting its link and the intercepted messages are routed through the workstation. They will be modified by software components before being forwarded to the other connected ECUs. This interception will be performed by an AE bridge, i.e. two compatible AE adapters converting from 100BASE-T1 or 1000BASE-T1 protocol to the workstation Ethernet protocol.

Requirements for Video Simulation. Notably, among various message types, the video stream is expected to have the highest bandwidth, capped at 30 frames per second (fps). Any modification of intercepted video frames must occur within a single frame duration, which is approximately 33 milliseconds, to ensure real-time processing. The test platform should enable viewing and analysis of individual camera feeds, assessing the effects of different test scenarios on both video and control messages. It should support real-time capture of video, playback of captured video, and logging of real-time video messages. Additionally, it should facilitate offline analysis of video messages and modify these messages. Moreover, the platform should allow for the injection of modified video messages back into the communication network.

Requirements for Software Component. The software component for the AEVisionLab test platform is designed with several key features and requirements:

(1) *Sniffing & Logging*: This is to capture Ethernet messages without disrupting the car's normal operation and storing them in an appropriate format for subsequent inspection.

(2) *Message Injection and Modification*: The software component needs to be capable of injecting (modified) messages into targeted AE links at arbitrary time intervals.
(3) *Interception*: The intercepted messages need to pass through the workstation bidirectionally for the purpose of testing and analysis. Thus, such interception should incur minimal processing time to ensure efficiency.
(4) *Fuzzing & Attacks*: Such a component needs to facilitate message fuzzing (e.g., modification). Likewise, different targeted attacks can be launched by message interception, modification and injection.

In summary, the software component in AEVisionLab enables interception and modification of arbitrary ECU messages while ensuring efficient performance to avoid impacting normal car functions on the test platform. It offers options to pass through, modify, block, or inject additional messages.

4 Implementation of AEVisionLab Test Platform

This section elaborates the design and implementation details of various components (see Figure 2) discussed in the preceding section.

4.1 Automotive Ethernet Testbed Implementation

The implementation of the automotive ethernet testbed involves ECUs responsible for controlling the cameras of the all-round vision system (see Figure 4) in a BMW Series 3 Seventh generation (G20) car model[4]. These ECUs include the Headunit (HU), Body Domain Controller (BDC), Camera Controller (TSRVC), Front camera (Fcam), Driver camera (Dcam), Passenger camera (Pcam), and Rear camera (Rcam). Table 2 describes the ECUs. These components are arranged on a test board as shown in Figure 30.

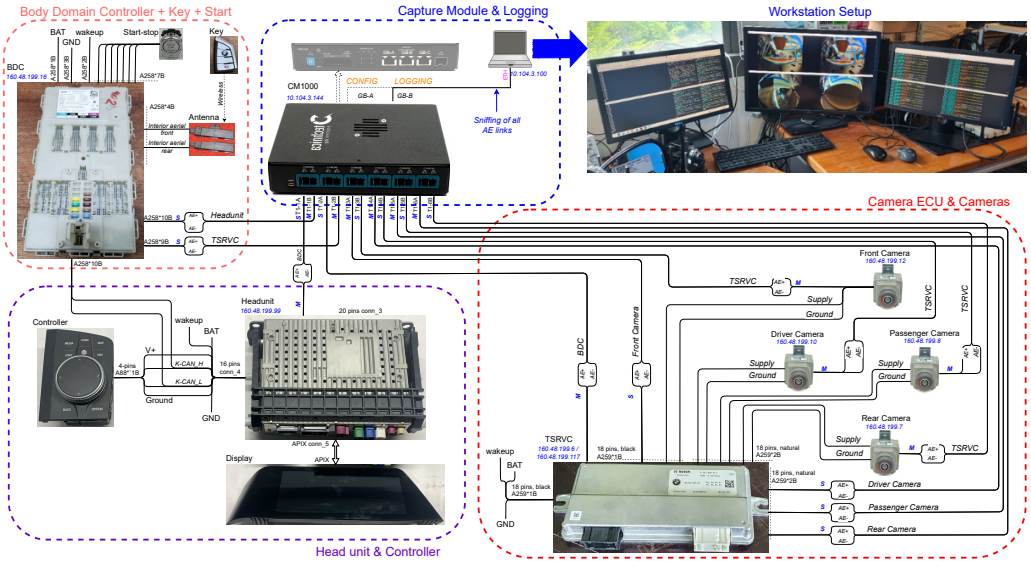


Fig. 4. Automotive AE Testbed Design

[4]https://en.wikipedia.org/wiki/BMW_3_Series_(G20)

Table 2. ECU Description

Item	Description
All-round cameras: Fcam, Rcam, Dcam, and Pcam	The cameras provide a surround view that includes a Rear View, Panorama View, Top View, and 3D View. The cameras in the exterior mirrors (Dcam and Pcam) as well as at the front (Fcam) and rear (Rcam) of the vehicle allow for a 360° view.
TSRVC	TSRVC stands for Top Side Rear View Controller. The controller is used to manage the different video feeds and display them.
BDC	The Body Domain Controller is responsible for managing network and identity requests. It acts as a gateway to control all modules in a centralized way, manage all loads, and allocate system resources.
HU	The headunit is the vehicle's audio-visual component providing screens, buttons, and system controls for numerous integrated information and entertainment functions.

4.2 Message Monitoring via Capture Module

The communication interface for capturing Ethernet messages is implemented by a capture module, CM1000 High MATEnet[5]. The captured messages are then logged by WireShark. In particular, the capture module is an active tap device capturing traffic supporting up to 6 point-to-point 100/1000BASE-T1 connections using the "Technically Enhanced Capture Module" (TECM) Protocol. The ports are configured via its web server. If further ports are required, the built-in synchronization using 802.1AS enables the cascading of multiple capture modules.

There are six AE links on this testbed and CM1000 is configured for 100Base-T1 sniffing from these six links as shown in Table 3. Each ECU is connected to an input port and assigned an interface ID. Each port is either Master or Slave depending on the design of the ECU.

Table 3. Capture Module Configuration

Link	Connect	ID	ECU	IP Address	100Base-T1
1	T1-1A	1	Headunit (HU)	160.48.199.99	Master
	T1-1B	2	Body Domain Controller (BDC)	160.48.199.16	Slave
2	T1-2A	3	Camera ECU (TSRVC)	160.48.199.117 / 160.48.199.6	Master
	T1-2B	4	Body Domain Controller (BDC)	160.48.199.16	Slave
3	T1-3A	5	Camera ECU (TSRVC)	160.48.199.6	Slave
	T1-3B	6	Front Camera (Fcam)	160.48.199.12	Master
4	T1-4A	7	Camera ECU (TSRVC)	160.48.199.6	Slave
	T1-4B	8	Driver Camera (Dcam)	160.48.199.10	Master
5	T1-5A	9	Camera ECU (TSRVC)	160.48.199.6	Slave
	T1-5B	10	Passenger Camera (Pcam)	160.48.199.8	Master
6	T1-6A	11	Camera ECU (TSRVC)	160.48.199.6	Slave
	T1-6B	12	Rear Camera (Rcam)	160.48.199.7	Master

All the links' messages are monitored from the output port GB-B and captured by the workstation via Wireshark. As shown in Figure 5, CM1000 encapsulates the message packet with a TECM header. The header shows the Interface ID which indicates the connected ECU. Wireshark also shows its decoded message's protocol.

4.3 Message Injection and Modification via Bridging

Bridging capability is implemented in the AE testbed to intercept messages from a targeted ECU. An illustration of this bridging process is depicted in Figure 6. In this example, messages from the Rear camera (Rcam) are isolated from

[5] https://www.technica-engineering.com/capture-module-1000-high/

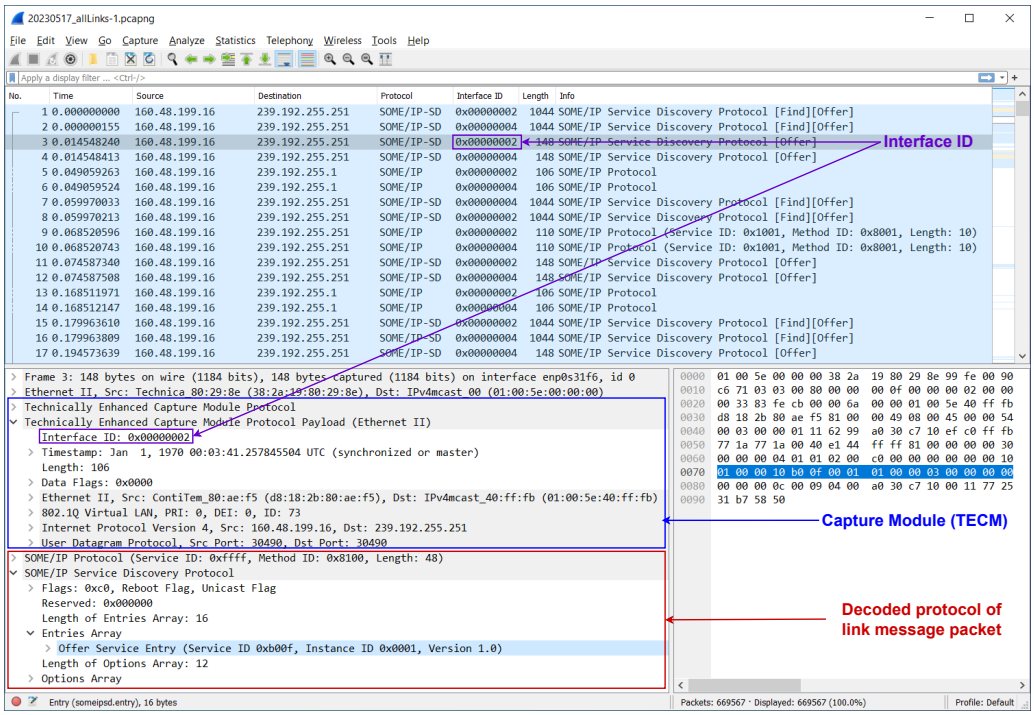


Fig. 5. Wireshark Sniffing

the Camera ECU (TSRVC) using two AE adapters. These adapters facilitate bidirectional data conversion between the vehicle's two-wire Ethernet 100BASE-T1 interface and the standard Ethernet RJ45 interface. Intercepted messages are then routed to the workstation and back to both the Rcam and TSRVC. These intercepted messages can either be passed as is (e.g., for inspection purposes) or modified by the workstation (e.g., for fuzzing and attacks). The link is also being actively tapped and logged by the Capture Module as part of the message monitoring.

4.4 Implementation of Software Component

The following software components are implemented and executed on the workstation.

Message Monitoring. For sniffing and logging of messages, we leverage on Wireshark - a free and open-source packet analyzer. We use Wireshark to monitor the captured messages from CM1000 and saving to a PCAP log file. Wireshark can also monitor and save the messages from the two bridging Ethernet interfaces on the workstation.

Message Injection and Modification. We implement injection and modification of messages via C programs. Nonetheless, any programming language can be chosen for this purpose. When a link is bridged, the targeted ECU is isolated and its messages pass through the workstation. We use `pfbridge` from the `PF_Ring library`[6] to implement the message's pass-through. This implements a one-directional pass-through from one Ethernet interface `eth1` to the other interface `eth3`. Another instance of pfbridge is required for the other direction i.e., from `eth3` to `eth1`.

[6] https://www.ntop.org/guides/pf_ring/index.html

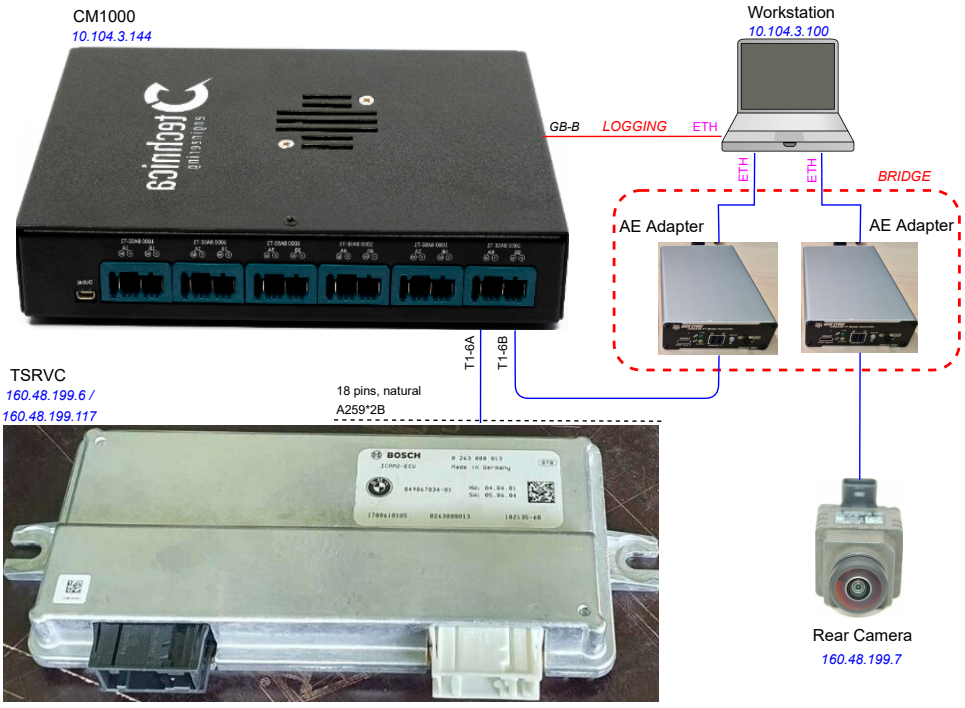


Fig. 6. Bridging the rear camera from TSRVC

Fuzzing or Attacks. At present, our testbed includes *random fuzzing* of the message packets. We randomly select a byte and replace its value randomly, before sending the fuzzed packet. Although our current implementation only supports random fuzzing, our test platform allows the community to research and implement sophisticated fuzzing algorithms on the test platform. The message buffer also facilitates our coding to modify the message to conduct *attacks* by injecting crafted messages to send to the ECU.

Video Monitoring. AEVisionLab includes a display to demonstrate the video from the four cameras. We develop a C program for this video monitoring in real-time. The camera sends each image frame with multiple video packets to TSRVC via its link. When the testbed is running, the video packets are collected by the workstation by sniffing from port GB-B of CM1000 and consolidated into an image frame. Continuous image frames are then displayed on the monitor to form a video stream (see Figure 7). For display purposes, we leverage `FFmpeg` - a free and open-source software for video processing. `FFmpeg` includes several different libraries and programs to manipulate and handle video files.

4.5 Video Simulation

We set up a video simulation framework incorporating various cameras of the all-round vision system. The video simulation enhances the research for navigation, safety, security, and other similar applications. It has two types of video simulation: *(i) Simulated road scenes* where video of road scenes are played back on monitor displays and captured by the cameras to simulate various driving scenarios. *(ii) Augmented video messages* where video messages are intercepted

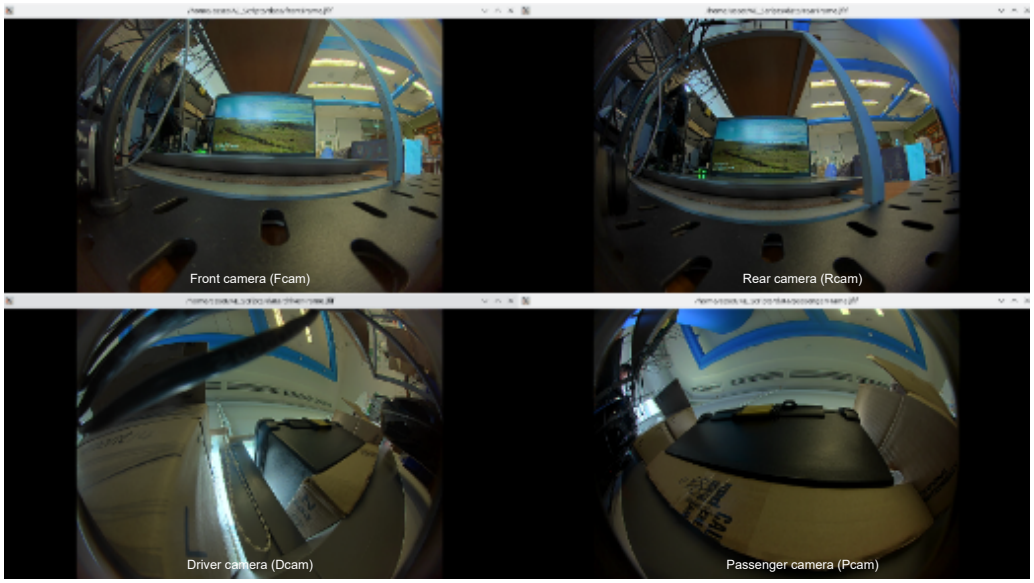


Fig. 7. Display of Four Cameras on Workstation

from the camera and augmented before returning to their AE link. These two capabilities are discussed in more detail below:

(i) Simulated Road Scenes. The car simulated by AEVisionLab has four cameras connected by 100BASE-T1 links to the camera ECU. The front camera looks forward, the rear camera looks behind the car, and the passenger and driver cameras look to the left and right sides of the car, respectively. We align a camera to focus on a monitor display playing a simulated road scene, and the camera converts the captured images into video messages to the camera ECU.
Video Messages: The video messages are viewed in real-time on Wireshark and also logged for subsequent offline analysis. While Wireshark only shows that it is an AVTP protocol with an *unknown format*, we employ *reverse engineering* (explained in Section 4.6) to reveal that the messages shown by Wireshark are in 12-bit compressed JFIF format. The messages are not interrupted but tapped from the original camera link by the capture module, and hence, it has no impact on the function or timing of the camera. Each camera is assigned to a CM1000 port and can be identified by the ID in the TECMP messages.
Video Live Feed and Playback: Different road scenes are played back on an external monitor display. The videos of these road scenes are collected from other sources, or converted from pre-logged video messages of the AEVisionLab testbed. These videos serve as live feed to the camera.

(ii) Augmented video messages. A camera link can be intercepted by bridging. The intercepted video messages are then routed to the workstation for modification, blocking, or injection of additional video messages, before returning to its original link.
Modification of Video Messages: A video stream consists of multiple video frames. Each video frame is transmitted as multiple packets of the AVTP messages. The information of each message packet can be modified via fuzzing or the transmission of the packet can be delayed arbitrarily.

Blocking and Injection of Additional Video Messages: The video messages can be completely blocked or injected with additional crafted AVTP messages. A fake video stream can also be injected by adding its own AVTP messages.

4.6 Reverse Engineering

AEVisionLab employs reverse engineering when the message IP address, format, or protocol is unknown. In the following, we describe the key techniques that we employ (see Figure 8) for such reverse engineering.

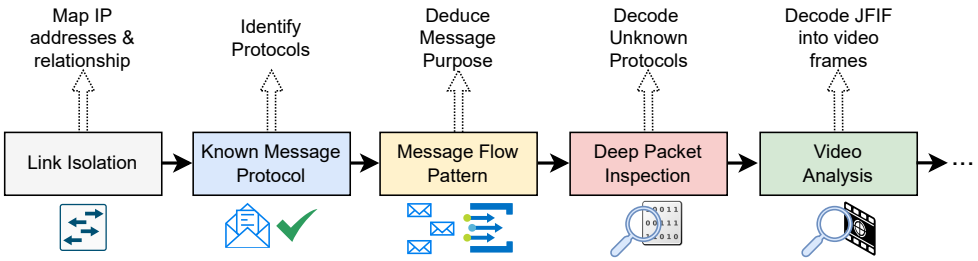


Fig. 8. Process of Reverse Engineering

Link Isolation. We identify the communication link between two ECUs by the CM100 interface IDs. In particular, by studying the messages with respect to each communication link, we determine the source ECU and its IP addresses. The intermediate ECU relaying messages are also identified, such as TSRVC sending a message to HU via the BDC relay (Figure 9). Overall, this process allows for mapping the ECUs' IP addresses by checking the source and destination addresses, and relay ECUs.

The procedure of link isolation for reverse engineering is as follows. The capture module is wired according to Table 3 and each wire link is assigned an interface ID. Figure 9 illustrates the communication between the links of TSRVC to BDC (ID=3), BDC to HU (ID=2), HU to BDC (ID=1) and BDC to TSRVC (ID=4). Using Wireshark to monitor from the capture module, we can identify the different messages by their link IDs and identify their source and destination IP addresses. An ECU can have multiple AE links. A message can end in an ECU or relay by passing through this ECU. If the same message continues through a relay ECU to the other ECUs, then the destination IP address belongs to the final ECU. The destination IP address is confirmed when the ECU sends the message as the source IP address. For example, TSRVC sends a message as the source IP 160.48.199.117 and destination IP 160.48.199.99 to BDC (ID=3). The BDC relays the message to HU (ID=2). Similarly, HU sends a message as the source IP 160.48.199.99 and destination IP 160.48.199.117 to BDC (ID=1), which relays the message to TSRVC (ID=4). Hence, we deduce that the IP addresses of TSRVC and HU are 160.48.199.117 and 160.48.199.99, respectively, and BDC is the relay ECU.

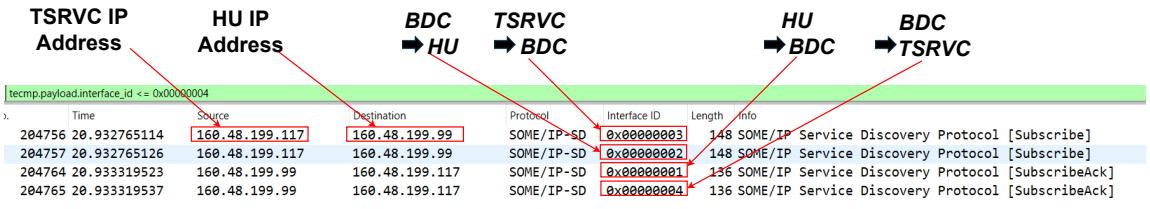


Fig. 9. BDC Relay Messages

Known Message Protocol. Wireshark can decode commonly used message protocols. In addition, some generally known port numbers also help to identify the protocol. By trial and error, we find additional port numbers or fields for Some/IP and IEEE1722 (AVTP) as shown in Figure 10. Using these fields, initially unknown protocols can be decoded.

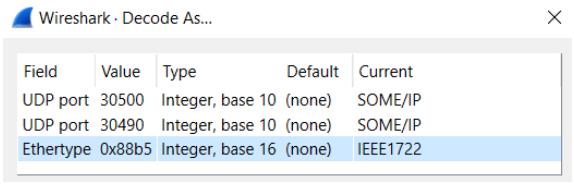


Fig. 10. Additional WireShark Decoded Protocols

Before adding the port numbers, we found that many messages using port numbers 30490 and 30500 are not decoded by Wireshark. In particular, port numbers 30401 - 30831 are unassigned by the Internet Assigned Numbers Authority (IANA) [7]. Using literature search, we discover that port number 30490 is usually used for SomeIP Discovery Service and noted that most automotive systems use SomeIP protocol. Specific search for SomeIP and port number 30500 indicated that it can be used as a default or temporary client port for SomeIP. We added these ports to Wireshark decoding and find that the decoding does not exhibit errors.

We set up the test platform incrementally with ECUs. After we connect a camera to the test platform, we notice a significant number of unknown messages flowing through the link from the camera to the TSRVC. We posit these messages to be video messages. The common video protocol is IEEE 722 i.e., Audio Video Transport Protocol (AVTP), which defines transmission of video data over Ethernet networks. Since IEEE1722 uses Ethertype `0x88b5`, we added it for Wireshark decoding.

Message Flow Pattern. Boot-up and regular message flow reveal that certain information is communicated between the two ECUs. In certain cases, the purpose of the information contained in the messages can be deduced from the message flow pattern. First, we capture messages from boot-up until the messages reach a steady state. In this state, message-flow pattern repeats itself. We repeat this process for multiple captures from boot-up. Then we compare the captured messages to check for repeated pattern at boot-up. The boot-up similarities are noted, which we used to detect the assignment of camera IP address (see the example in the subsequent paragraph). For regular message flow, we noted the SomeIP services, i.e. “Offer", “Request" and “Response". We found repeated SomeIP “Response" services for each ECU’s link as shown in Figures 19(b). From this information, we devise the SomeIP attacks in Section 5.3.4.

As an example, our purchased front camera does not have an IP address initially. The link messages between the camera and the camera ECU can be identified by the TECM message interface IDs (5 and 6). The video stream messages show that the camera MAC address is (`fc:d6:bd:37:63:4a`) and the IP address is 160.48.199.12 (`a0.30.c7.0c`). From the flow of messages in this link, we find a proprietary UDP message that assigns the IP address to the camera.

- When the system boots up, there are UDP messages from the front camera and camera ECU. The camera starts by broadcasting a UDP message source ID of `0.0.0.0`.
- The camera ECU responds with another UDP message with a payload. We find bytes within the payload (Figure 11) that tally with camera MAC address (`fc:d6:bd:37:63:4a`) and IP address (hex `a0.30.c7.0c`).
- Subsequently, the front camera starts the video stream to the camera ECU.

[7] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

- We can deduce from the two handshaking UDP messages that the camera (MAC address = `fc:d6:bd:37:63:4a`) requests an IP address and IP address=160.48.199.12 is subsequently assigned by the camera ECU.

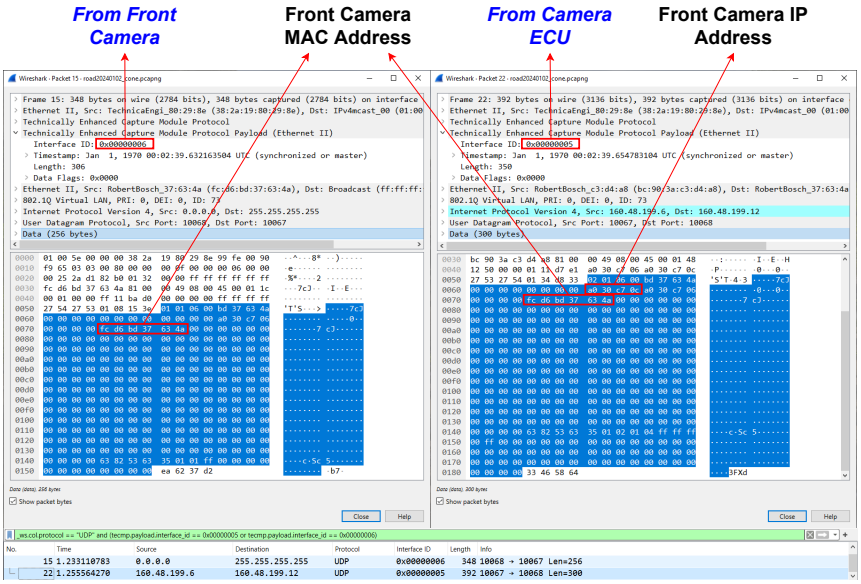


Fig. 11. Assign Camera IP Address

Deep Packet Inspection. Most automated reverse engineering tools are for CAN network and from the OBD connection [10, 18, 30], but does not support AE network. Wireshark allows us to inspect individual packet to the message's data byte level. We use visual inspection in Wireshark to find clues to understand unknown protocol in the AE network. Clues like similarities among the logged AE message packets or familiarities that are related to the intended ECU's function helps the reverse engineering process. For example, we investigate clues for image frame standard, as it is related to the video stream.

The logged AE message packets needs to be studied in detail to investigate unknown protocols. For instance, we do not initially know the video stream protocol and format. Moreover, Wireshark decodes the video stream as an audio stream. After analyzing the message flow pattern (Figure 12), we spot the text "`JFIF`" by inspecting the raw data bytes whenever there is a video stream. Table 4 shows the "`JFIF`" format of the video packets. To extract the raw image, we map up the different headers of the image from the data bytes of multiple AVTP message packets such as "Start of Image", "End of Image", "Quantization Table", "Huffman Table", etc. We form each image frame by consolidating from the first "Start of Image" packet to the last "End of Image" packet.

Video Analysis. The video stream is from a series of image frames. After forming a complete image frame from multiple AVTP message packets, we download the frame as a file to view it as a JFIF image. It is unsuccessful using standard application software. After troubleshooting the problem by analyzing the data of the image file using Table 4, we find that the JFIF image is using non standard format, i.e. it is neither 8 bits or 16 bits precision. Dedicated software has to be programmed to view the image.

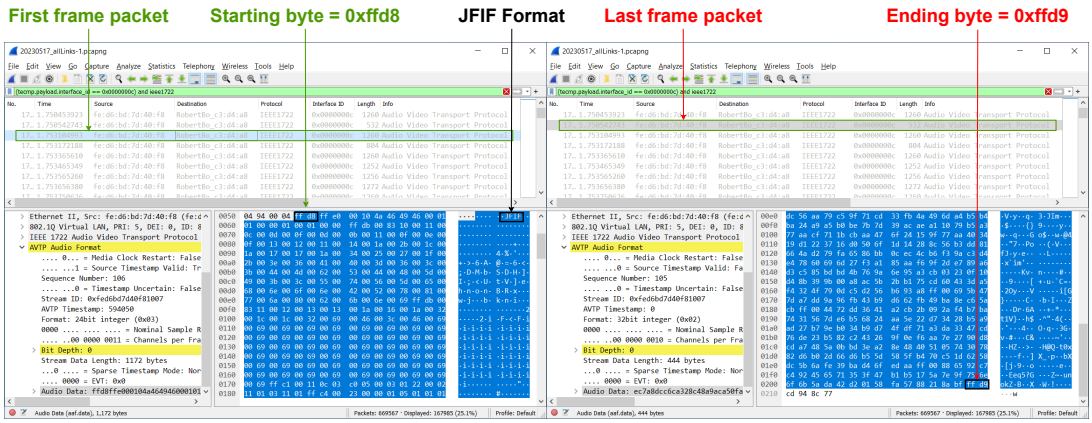


Fig. 12. Decoding the Video Packets

Table 4. JFIF Structure

Header	ID	Name	Comments
SOI	0xffd8	Start of Image	
APP0	0xffe0	Application Segment 0	Application meta-data
DQT	0xffdb	Quantization Table	Defines one or more quantization tables
SOF1	0xffc1	Start of DCT Frame 1	Data precision, width, height and number of components
DHT	0xffc4	Huffman Table	Defines one or more Huffman tables
DRI	0xffdd	Restart Interval	Interval between RSTn markers
SOS	0xffda	Start of Scan	Starts a top-to-bottom scan of the image
RSTn	0xffd$n$	Restart Marker n	Every r restart interval - where r is set by DRI marker
EOI	0xffd9	End of Image	

We know from the decoded information image headers that the image resolution is 1280×960 and the data precision is 12 bits per sample. Further packet inspection shows that the video uses three Huffman tables and two quantization tables. The image is `YCbCr` encoded, i.e., luminance (`Y`) uses the first quantization table, and chrominance (`CbCr`) uses the second quantization table. We note that the information obtained from reverse engineering the video was crucial for AEVisionLab to illustrate video simulation and video manipulation, as discussed in Section 4.5 and Section 5.3.3.

5 Evaluation

In this section, we discuss potential usage scenarios for the AEVisionLab test platform. It briefly overviews the experimental setup, followed by a discussion of the platform's key capabilities. Finally, we detail studies conducted using the platform for security evaluation and testing of attacks.

5.1 Testing Setup

There are three possible testing scenarios i.e., message monitoring (Figure 2a), message injection (Figure 2b), and message modification (Figure 2c). The messages are monitored by tapping the links via the capture module. The setup for message injection and modification is the same, i.e., the targeted ECU is isolated by bridging the respective link.

After connecting the links according to the test scenario, the workstation starts the testing as follows:

- Launches Wireshark to monitor and log messages from all channels.
- Launches the programs for bridging, fuzzing and attacks.
- Powers up the testbed.
- When the testbed is powered up, messages flow in different links. These messages are sniffed and displayed on Wireshark. Data collection is also activated by saving the links' messages in a PCAP file.

5.2 AE Testbed Capabilities

AEVisionLab offers comprehensive message monitoring and is capable of real-time monitoring of various functionalities such as camera video. Other monitoring means, for example, audio and sensor information can be added if required. AEVisionLab also enables real-time injection and modification of messages.

5.2.1 Evaluation of Message monitoring. The testbed is powered up and 2,052,514 messages are logged for two minutes. We evaluate the message monitoring in a few aspects:

- *ECUs links:* We observed all the messages at each of the six links and there were no missing messages. For the cameras' links, video AVTP packets occupy most of the communication, i.e., 99% of each camera's link packets.
- *Types of messages:* Using Wireshark and reverse engineering (see Section 4.6), the decoded protocols are shown in Figure 13. Excluding AVTP messages, the amount of unknown messages is below 0.1%.
- *Size of logged file:* The file size is significantly large for two minutes of logging, i.e., 2.5GB. This is primarily due to the AVTP packets. Consequently, during monitoring, Wireshark starts to occupy an extensive hard disk space even before saving the logged file.

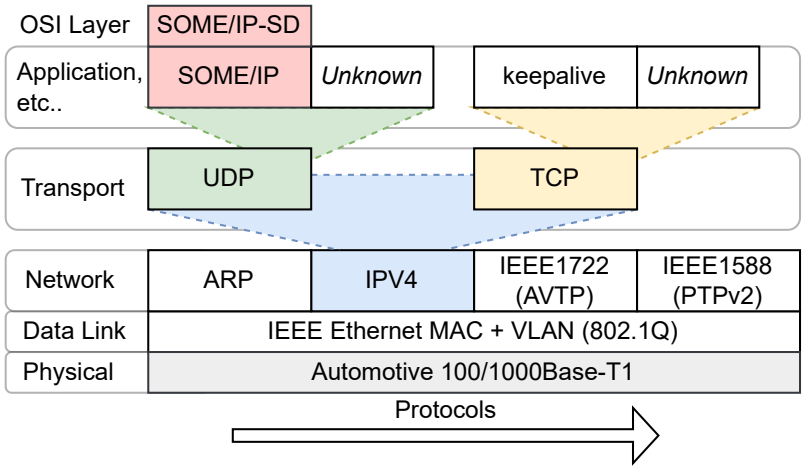


Fig. 13. Decoded Protocols

5.2.2 Video monitoring. The inter-packet timing from the collected video packets is within 0.09ms to 0.11ms (detailed in Figure 20). We evaluate the feasibility of displaying real-time video from the four cameras using the following three methods:

- A *Python program* to frame packets into a JFIF image, convert to jpeg and display on screen.
- *Wireshark LUA programming* to download and frame packets to JFIF image and use FFmpeg library to display on screen.
- *C programming* to download and frame packets to JFIF image and use FFmpeg library to display on screen.

The first two methods are too slow and cause a backlog of image frames. The third method, employing a single C program for framing images and displaying them with FFmpeg, is fast enough for real-time monitoring. Further tests confirm the feasibility of streaming video from all four cameras. Importantly, this video processing is performed offline and does not impact the real-time video streaming or accuracy within the testbed.

5.2.3 Message injection and modification. To test the bridging scenario for intercepted messages, the Front camera (Fcam) is isolated by bridging it from the TSRVC. Intercepted messages between Fcam and TSRVC pass through the workstation, to bridge the interfaces in both directions. We measure the latency of both non-AVTP and AVTP packets directly from the capture module, which provides precise latency measurements via hardware timestamping and end-to-end system measurement.

For the latency test, we re-wire the test platform and connect the front camera to interface 6 and the TSRVC to interface 7 of the capture module, as shown in Table 5. AVTP messages flow from the camera into interface 6 to the workstation and from the workstation into interface 8 to TSRVC. The dataset "`latency_test_ifc5-8.pcapng`" is provided with AEVisionLab [8]. The non-AVTP and AVTP messages can be filtered in Wireshark using filters "!aaf" and "aaf", respectively. The timestamp of each bridged message is obtained by comparing the corresponding camera's message and the TSRVC's message.

Table 5. Bridging Configuration

Link	Connect	ID	ECU	IP Address	100Base-T1
1	T1-3A	5	Workstation		Slave
	T1-3B	6	Front Camera (Fcam)	160.48.199.12	Master
2	T1-4A	7	Camera ECU (TSRVC)	160.48.199.6	Slave
	T1-4B	8	Workstation		Master

Concretely, we measure the latency of 523,489 messages (non-video: 9,144; video: 514,345) passing through the workstation, monitored by the capture module at the input and output of the bridge. Figure 14 and Figure 15 show the results. The median AVTP latency of 0.465 ms between interfaces 6 and 8. This is within the 33 ms frame interval (30 fps), with no observable impact on the camera's video display. The data also does not show any missing image frame. For time-sensitive Audio Video Bridge (AVB) network, the latency targets from IEEE Std 802.1BA-2021 [1] for SR class A is 2 ms and SR class B is 50 ms. The camera is connected to the camera ECU via a single workstation bridge, and the measured AVTP latency of 0.465 ms is within the IEEE standard..

The non-AVTP messages consist of mainly SomeIP messages. They appear in regular intervals from interfaces 5, 7 and 8. The shortest regular interval in the dataset is 6.3 ms. Moreover, from fingerprinting in Section 5.3.1, Figures 20(c)-(d)) show that the shortest non-AVTP regular interval from non-camera links is 2.67 ms. The median non-AVTP latency test result of 0.16 ms is therefore acceptable. This is because such a latency is within the shortest regular non-AVTP interval of 2.67 ms.

[8] https://github.com/yeoant/AEVisionLab/tree/main/Dataset

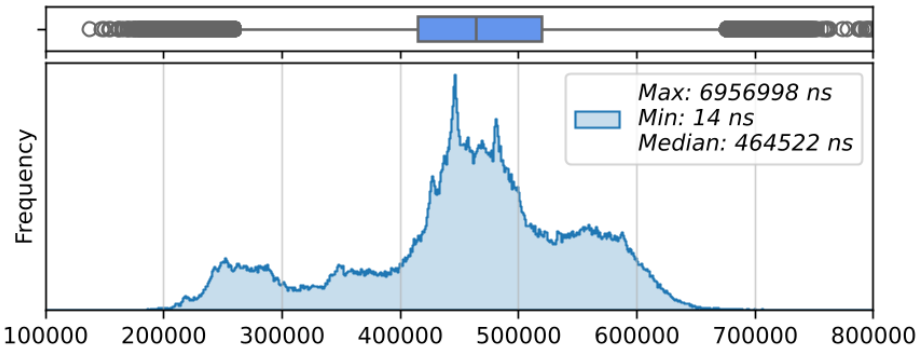


Fig. 14. Video Pass-through Messages Latency (ns)

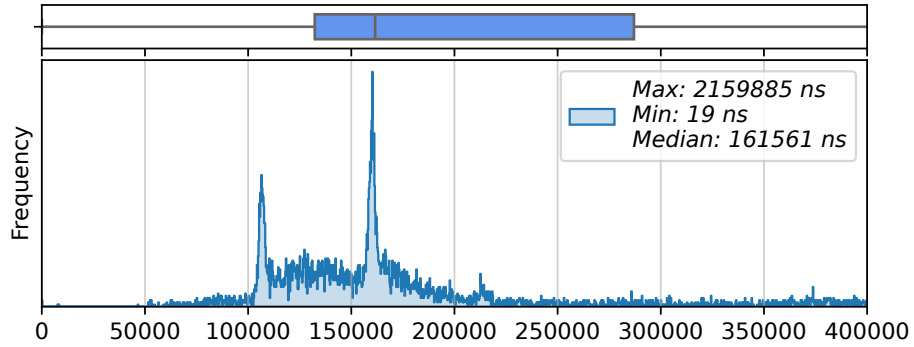


Fig. 15. Non-Video Pass-through Messages Latency (ns)

We modify arbitrary pass-through messages or craft fake ones, simulating their injection from an isolated ECU to another connecting ECU during bridging. For example, we isolate the Headunit (HU) from the Body Domain Controller (BDC) by bridging and crafting a subscription message for a SomeiP service (0x3544) via BDC to TSRVC, the service provider, as shown in Figure 16. This crafted message is then used to conduct eavesdropping on the same service provided to BDC, as described in the subsequent case study on attacks.

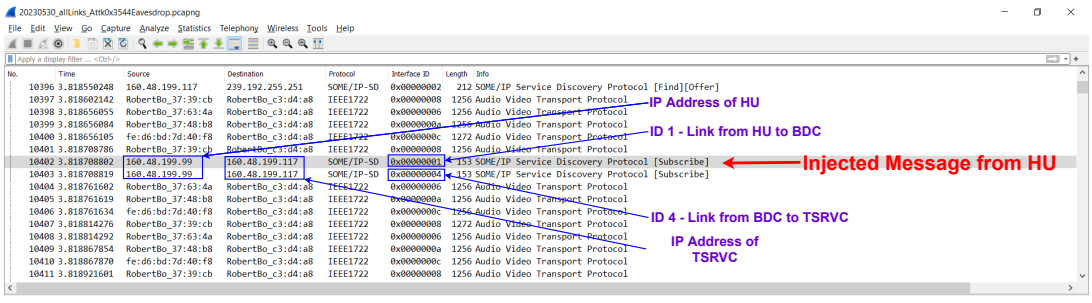


Fig. 16. Injected Message from HU to TSRVC

5.2.4 Video manipulation. We evaluate AEVisionLab's capabilities to modify video streams via manipulating AVTP message packets or individual image frames in both real-time and offline scenarios. The corresponding testing setup is illustrated in Figure 17. We note that real-time video manipulation is challenging due to the tight 0.09 ms packet interval and 33 ms frame interval. Additionally, offline image processing needs to be performed to obtain a modified video stream, which can then be injected back into the test platform in real-time. In the following, we discuss a few salient features of our video manipulation.

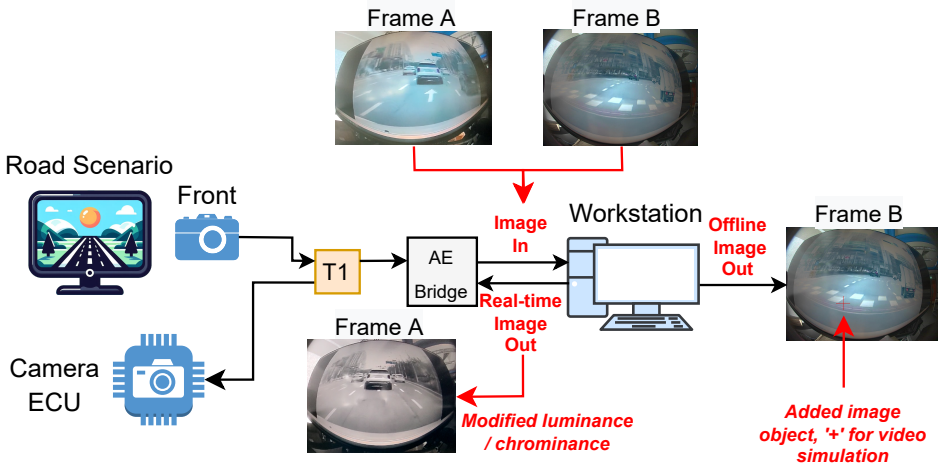


Fig. 17. Video Manipulation in AEVisionLab

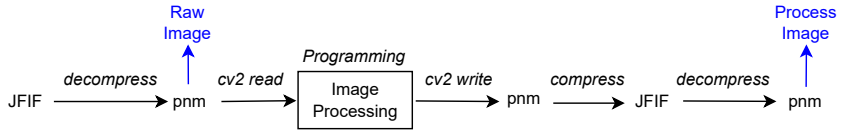
Real Time Manipulation. JFIF protocol has two quantization tables for luminance and chrominance used for decompressing images. We modified the video stream's luminance and chrominance by altering these tables within the AVTP message packets. Changing the chrominance table adjusted the image color. As illustrated in Figure 17, Frame A is converted from color to grayscale. We observe that luminance changes affect the brightness and pixelization of the image, impacting the visibility of road lane markings.

Offline Manipulation. We evaluate offline image processing on the collected JFIF video stream using the `libjpeg-turbo` library[9]. This library is used to decompress and compress JFIF files to PNM format. We develop a Python program to add a "+" object at the center of the image, as shown in Figure 18(a) and illustrated in Frame B of Figure 17. The JFIF image is the raw image from the actual car, without any conversion to other formats like MP4, and is the same image used in the car's operation. We describe a more complex video simulation (aligned with real-world attack scenarios) in Section 5.3.3.

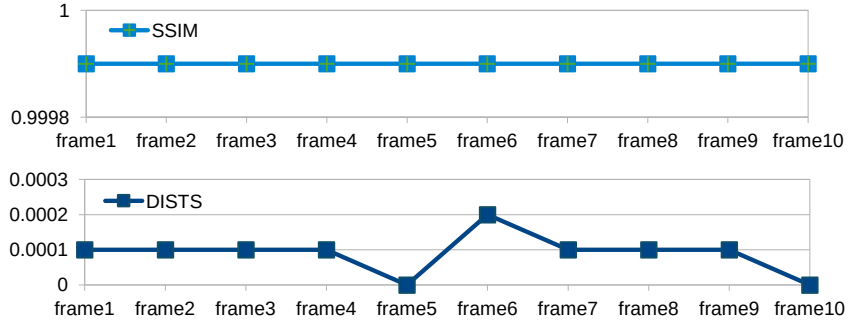
To check for the degradation of the quality of raw images during processing (see Figure 18(a)), we compared the processed image with corresponding unmodified image (i.e., without any image processing). In particular, we computed the Structural Similarity Index Measure (SSIM) and Deep Image Structure and Texture Similarity (DISTS) [10] for ten image frames (see Figure 18(b)). The SSIM index of 0.9999 indicates a high similarity between the processed and

[9] https://github.com/libjpeg-turbo/libjpeg-turbo
[10] https://github.com/chaofengc/Awesome-Image-Quality-Assessment

initial uncompressed raw images. The DISTS index of less than 0.0002 shows no degradation in the processed images, confirming that the conversion process preserves image quality.



(a) Process Image Flow



(b) Image Conversion Similarity Evaluation

Fig. 18. Image Processing

5.3 Case Studies

In this section, we discuss four different case studies using the monitoring, injection, and modification capabilities described in the preceding section.

5.3.1 Case Study I (Fingerprinting). AEVisionLab test platform captures the sequence of messages from each AE link and analyzes their protocols in a process called fingerprinting. This involves the following actions to identify and characterize the messages from ECUs:

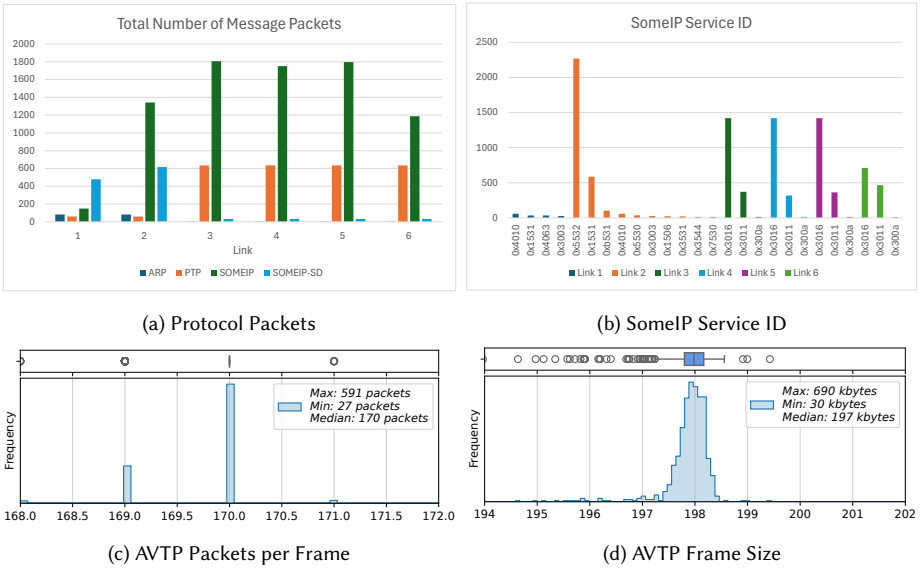
- Active tapping each pair of ECUs' link by CM1000,
- Collect the AE messages by Wireshark,
- Analyze the collected AE messages.

We identify the ECUs' source messages, IP addresses, and protocols from the collected AE messages. We also gather statistics, message frequency, and inter-packet intervals for periodic messages. The IP addresses, protocols, and message statistics form the ECUs' fingerprints for the car. The built-on functionality embodied in Wireshark can identify some messages as ARP, PTP, or UDP protocols. However, reverse engineering is required to identify the ports for SomeIP and SomeIP-Service_Discovery (SomeIP-SD) protocols. After successfully finding the ports, Wireshark can decode the protocols for SomeIP and SomeIP-SD.

Wireshark incorrectly decodes video packets (AVTP protocol) as audio packets. To correctly decode the AVTP packets, hence, significant reverse engineering effort was involved, as detailed in Section 4.6. In particular, the reverse engineering process discovered that the video packets use the JFIF format, starting with `0xffd8` and ending with

`0xffd9`, and utilize an older 12-bit data format unsupported by Wireshark. To address this, a program was written to assemble the AVTP packets and display the image frames correctly (see Figure 7).

We performed further analysis over 30 seconds of data to study the different messages by filtering them according to their ECU and protocols, such as SomeIP, AVTP, ARP, and PTP messages. Figures 19(a)-(b) show the number of different protocol messages and the major SomeIP services for each link. As discussed before, many AVTP packets form a single video frame. Links 3-6 are for the four cameras to the TSRVC and their AVTP behaviors are the same (refer to Table 3 for the links definition). Figures 19(c)-(d) show the distribution of the total packets for each frame and its frame size from link 3 (Fcam). By studying the protocols and employing the fingerprinting process, we understand the flow of different messages between ECUs. This, in turn, facilitates a fingerprint of the activity within the test platform and such a fingerprint can be used for various purposes including but not limited to the detection of abnormal message flow.



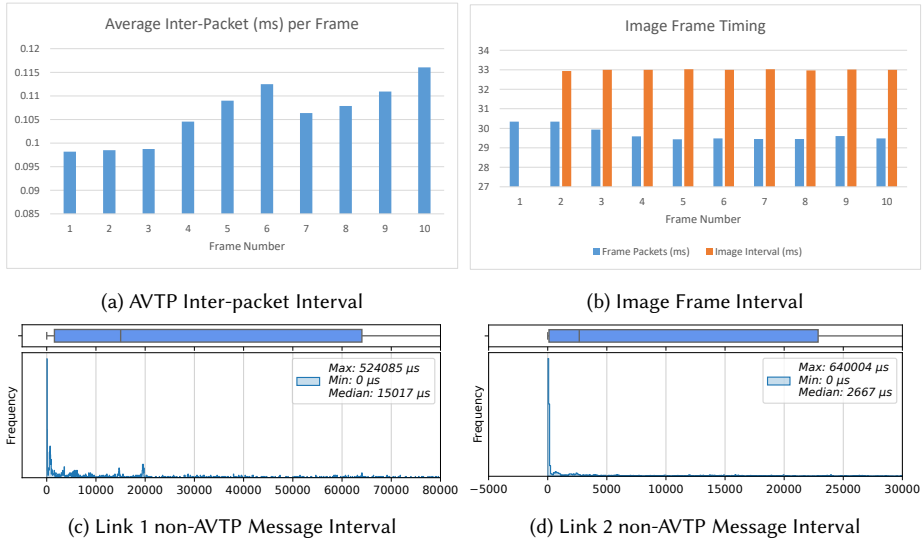
(a) Protocol Packets (b) SomeIP Service ID

(c) AVTP Packets per Frame (d) AVTP Frame Size

Fig. 19. Fingerprinting the Messages' Protocols

We also collect one minute of data from all links to analyze the frequency of AE messages and the timing of regular AVTP packets from link 3 over ten consecutive image frames. Figure 20(a)-(b) show that the inter-packet interval ranges from 0.098 ms to 0.116 ms, and each image frame interval is approximately 33 ms, corresponding to 30 fps video. Figures 20(c)-(d) show that the frequency of AE non-AVTP messages from links 1-2 (HU, BDC and TSRVC) is significantly slower, with medians of 15.0 ms and 2.6 ms, respectively. Such time-interval-based fingerprints can also provide insights on the presence of abnormal activities e.g., during an attack. This set of experiments shows that AEVisionLab test platform allows for the reliable collection of various information for fingerprinting the message flow and timing within the in-vehicle network.

5.3.2 Case Study II (Random Fuzzing). We enable *fuzzing* actions on arbitrary messages within the AEVisionLab test platform to facilitate comprehensive security testing. Once the targeted message for fuzzing is intercepted, there are several ways to fuzz the message from one ECU to the other ECU. As a proof-of-concept fuzzing implementation, we perform the following actions:



(a) AVTP Inter-packet Interval (b) Image Frame Interval

(c) Link 1 non-AVTP Message Interval (d) Link 2 non-AVTP Message Interval

Fig. 20. Fingerprinting the Messages' Timing

- Select random or targeted message protocol,
- Fuzz selected random or targeted data byte, or
- Fuzz selected random or targeted bytes corresponding to a decoded field.

Additionally, to arbitrarily modify the message flow through the network, the following mechanisms can be performed:

- Block the targeted message.
- Send the targeted message back to the destined ECU with modifications via fuzzing.
- Replay multiple copies of the targeted message.

For instance, random fuzzing of one byte in the "precisionOriginTimestamp (seconds)" field of the PTP message from TSRVC to Rcam is shown in Figure 21. The modified value is selected randomly, with only the targeted field changed while the other bytes remain unchanged. Figure 21 illustrates the capture of two messages from the input interface and the output interface, and the fuzzing of the targeted field.

5.3.3 Case Study III (Video Simulation). We evaluate the feasibility of modifying the video stream captured by the testbed. In order to do so, we extend the video manipulation capabilities detailed in 5.2.4 to alter the road scene seen by the user. We use the testbed to capture a real video stream (1537 frames) from the camera as shown in Figure 17 and attempt to add a persistent traffic cone to a short section (250 frames) of stream. In particular, we first convert the JFIF image to a PNM image with the aid of libjpeg. We then leverage several heuristics to determine the size, and the exact location of the cone and insert it into the PNM image. We then convert it back to the JFIF format and inject it back into the testbed.

In our experiments, we used an Intel i7-11800H @ 2.30GHz machine with 16GB of RAM. We found that extracting the JFIF from the PCAP file takes at least 10ms. Similarly, we also found that the conversion between JFIF and PNM minimally takes 22 ms in each direction. Since the extraction and conversion process together take more than the 33ms inter frame interval, real-time modification of the image stream is currently infeasible using our current test platform.

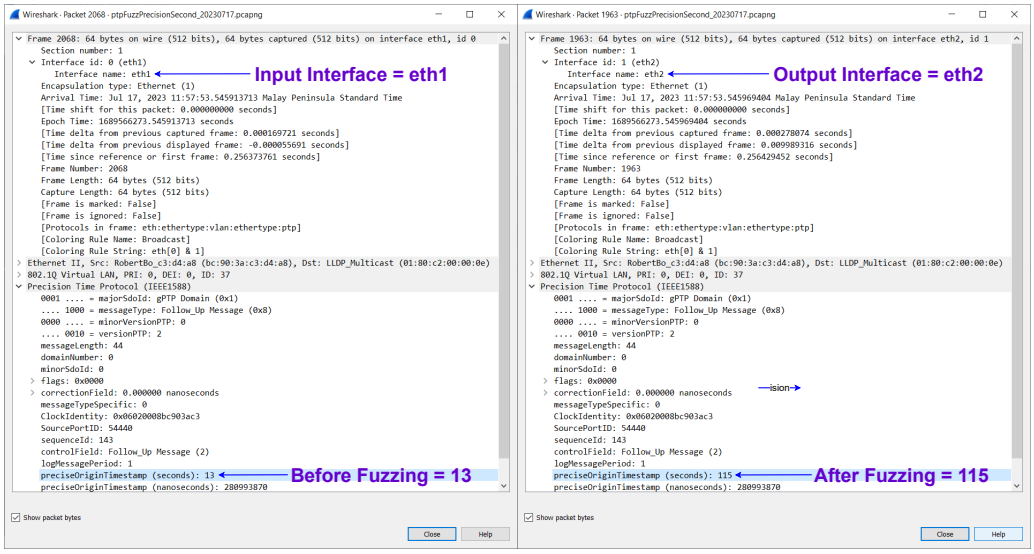
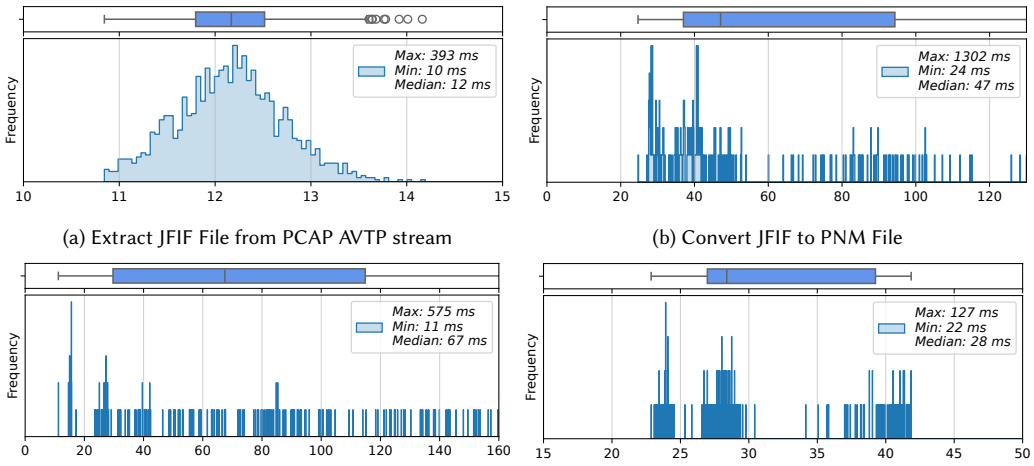


Fig. 21. Fuzzing of precisionOriginTimestamp (seconds)

We also evaluated the time to analyze and modify the image as observed in Figure 22. We further detail the heuristics involved in the insertion of the cone below.



(a) Extract JFIF File from PCAP AVTP stream

(b) Convert JFIF to PNM File

(c) Add Cone to PNM File

(d) Convert PNM to JFIF as per original AVTP format

Fig. 22. Timing - Add Cone to One JFIF File

Consider a single image frame. We first determine the relative size of objects in the frame. The size of an object in the captured image is dependent on two factors, namely the actual physical dimensions of the object and its distance from the camera. However, the rate at which different objects change in size, as a function of distance, is constant. Consequently, the relative size of objects must be preserved when objects are at a similar distance from the camera. We

leverage this intuition to find the height (number of pixels) of a reference object already in the image frame and use this to determine the height of the cone if it were to be placed in the same location as the reference object. In addition, we also determine how the size would change as the distance to the camera increases or decreases. This is then used to compute a scaling factor. These, in turn, help us find the size of the cone to be inserted.

We now consider a sequence of image frames. Suppose the direction and speed of travel are roughly consistent for a short period of time. In such a case, we find that the rate at which an object approaches is likely to be consistent. We leverage this knowledge to determine the position of the cone in consecutive image frames. Consider a sequence of frames with a static reference object. If the object approaches the camera at some speed, then we can safely assume that the cone should also approach the camera at a similar speed. Thus, we vary the location of the cone throughout the sequence of targeted image frames. We also continue to use the scaling factor previously computed to find the size of the cone at the new location. We illustrate the qualitative results from this experiment in Figure 23 and demonstrate that the location of the added cone appears to be fairly static and maintains the appropriate size throughout the video sequence.

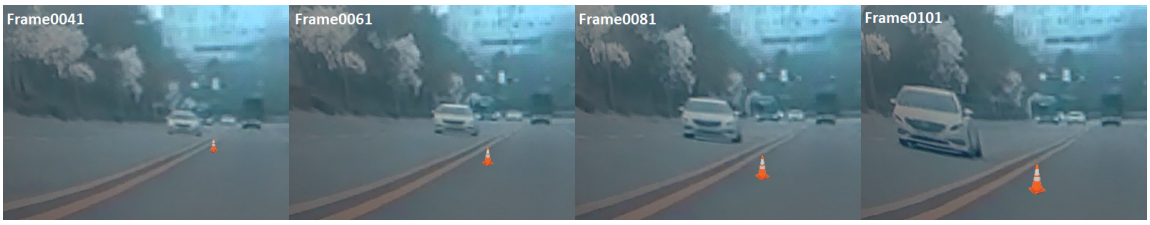


Fig. 23. A sequence of zoom-in image frames depicting the change in size and position of the cone over time. The shown frames are 20 frames (two-thirds of a second) apart.

5.3.4 Case Study IV (Attacks). In this section, we demonstrate that AEVisionLab test platform can be used to launch attacks without endangering an actual car and driver. In assisted driving or autonomous driving, one of the main operational components is the video streaming of the cameras. SomeIP messages are used extensively in automotive AE networks to communicate system functions and maintenance of network reliability among ECUs. Adversaries either target a car's functionality and reliability as a SomeIP MITM attack[11], or a safety-critical operational attack on the video stream. Hence, we design the attacks on SomeIP services and the camera video stream in a laboratory. Table 6 describes potential impacts of the attacks on the camera and SomeIP. Figure 24 illustrates an overview of our attacks.

(i) Camera Delay Attack. We establish a bridge between TSRVC and Rcam, creating a three-second video buffer at the workstation. This buffer continuously transmits a delayed video stream from Rcam to TSRVC. For illustration (Figure 25)), Fcam and Rcam are positioned to view a person's thumb from slightly different angles, as shown in the video display. Initially, at zero seconds (left of Figure 25)), Fcam shows a thumb-up, but Rcam does not. However, three seconds later (right of Figure 25)), Rcam displays the thumb-up, similar to Fcam at zero seconds, indicating a delayed version of the event. A real-time clock is displayed on the central screen to illustrate the delay (see Figure 25). This delay could cause a driver using the rear camera to reverse to collide with an object behind earlier than expected.

[11] https://plaxidityx.com/blog/engineering/some-ip-protocol-man-in-the-middle-attack/

Attack	Description	Possible Impact: Safety	Possible Impact: Funct
Camera delay	Autonomous or assisted driving rely on the camera. Any video delay will affect the driving. An example is using the rear camera for reversing. The delay will cause the reversing car to hit an obstacle earlier than expected.	●	
Stop camera	If an autonomous vehicle suddenly stops or diverts due to the camera stop, for example, at a dangerous place when crossing a traffic junction or a busy highway, it will pose a danger to the driver or other cars.	●	
SomeIP Eaves-drop	When a compromised ECU eavesdrops another ECU service, it can use the information for adversary purpose, for example devise a coordinated attack or divulge the driver's private information to a 3rd party.		●
SomeIP Hijack	When an attacker hijacks a SomeIP message from an ECU for an identified function, it can change the service to the ECU to cause a safety attack or affect the car function and reliability,	●	●

Table 6. Possible impact of attack. Safety: Safety-critical operation, Funct: Functional and reliable operation.

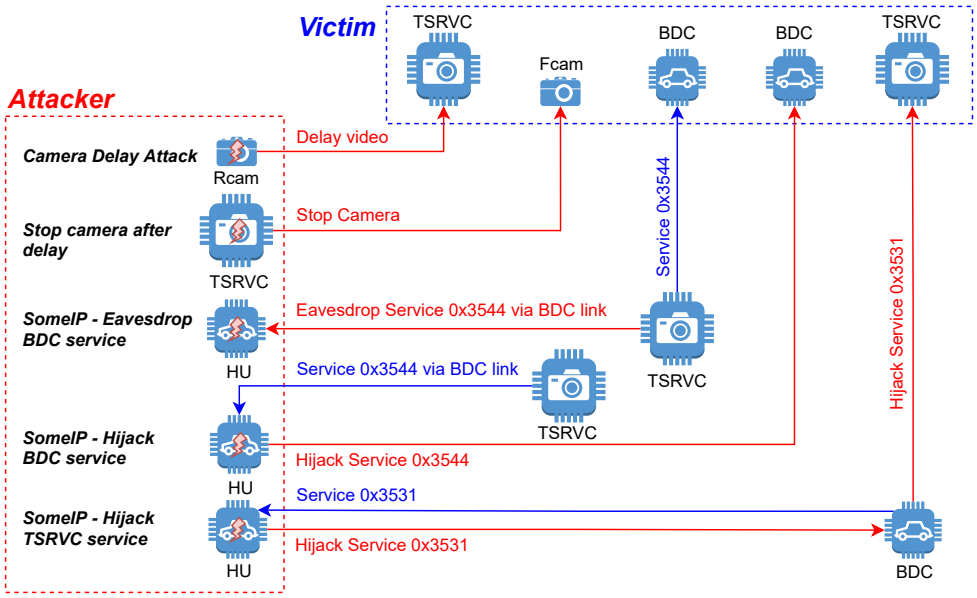


Fig. 24. Attacks conducted on AEVisionLab test platform

(ii) Stop camera after delay. During fingerprinting, we discovered that SomeIP service `0x300a` is related to the camera, with TSRVC requesting this service from the camera. Thus, we bridge the link between TSRVC and Fcam, and different method IDs for this service were fuzzed to confirm that method ID `0x3f` would request the camera to stop. To test this (see Figure 26), an attack was devised using a delay to trigger a fake crafted message requesting SomeIP service `0x300a` method `0x3f` to be sent by the TSRVC. As a result, Fcam responded to this message and stopped, while the other three cameras remained operational. Some cars use a front camera for lane assist, and the sudden stop of the front camera, as illustrated by this attack, will impact driving.

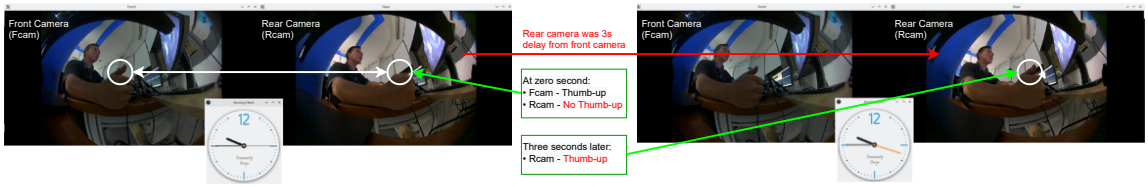


Fig. 25. Camera Delay Attack

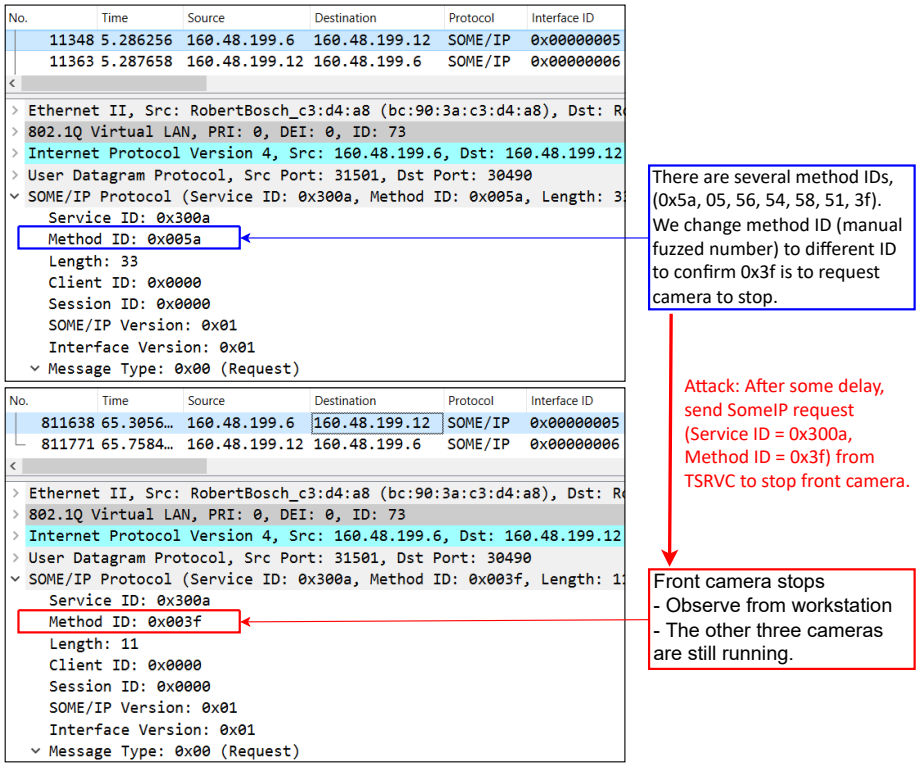


Fig. 26. Stop Camera After Delay

(iii) SomeIP - Eavesdrop/Hijack BDC service. For this attack, we intend to eavesdrop or hijack a SomeIP service. From the collected AE messages, we filter the messages for regular SomeIP services and find that BDC subscribes to a regular SomeIP service ID `0x3544` from TSRVC. Thus, we set up a bridge between HU and BDC. In normal circumstances, HU will not subscribe to this service. For the attack, whenever HU observes a service offer of `0x3544` from TSRVC via BDC (see Figure 27), HU will subscribe to TSRVC via BDC for the same service. Subsequently, TSRVC will send the service to both BDC and HU, resulting in HU to eavesdrop on BDC.

Instead of eavesdropping only, the attacker proceeds to another step to hijack the service from TSRVC to BDC. Concretely, whenever HU observes a service offer of `0x3544` from TSRVC via BDC (see Figure 28), HU sends a fake `StopOffer` as though it is from TSRVC to BDC. Then, HU sends its own offer of service `0x3544` for BDC to subscribe

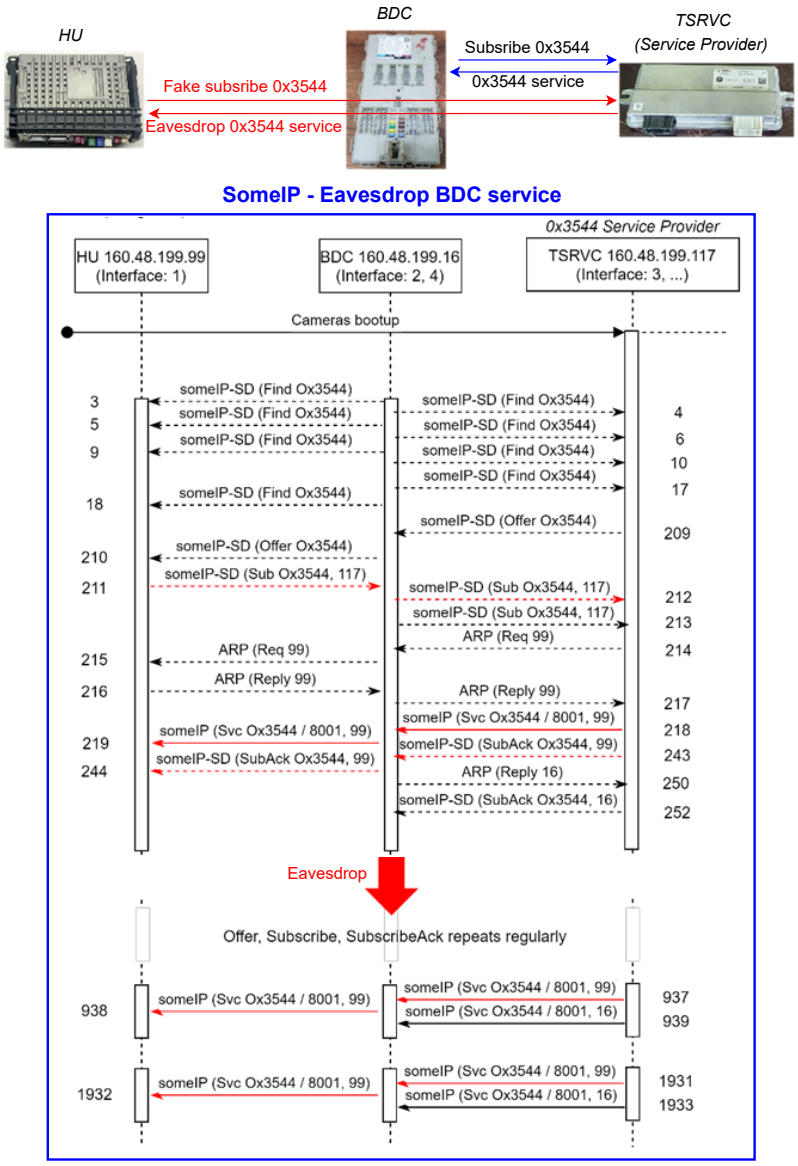


Fig. 27. Attack SomeIP Service 0x3544 Eavesdrop

and also concurrently, HU subscribes to the service from TSRVC. Subsequently, when HU receives the service from TSRVC, it can arbitrarily change the payload and relay the modified service to BDC. The result of the flow of service 0x3544 is illustrated on the top part of Figure 28, showing that HU has hijacked the service from TSRVC to BDC.

(iv) SomeIP - Hijack TSRVC service. In AEVisionLab, TSRVC regularly subscribes to SomeIP service 0x3531 from BDC. The difference from 0x3544 is that TSRVC subscribes to different method IDs (i.e., different functions) in service

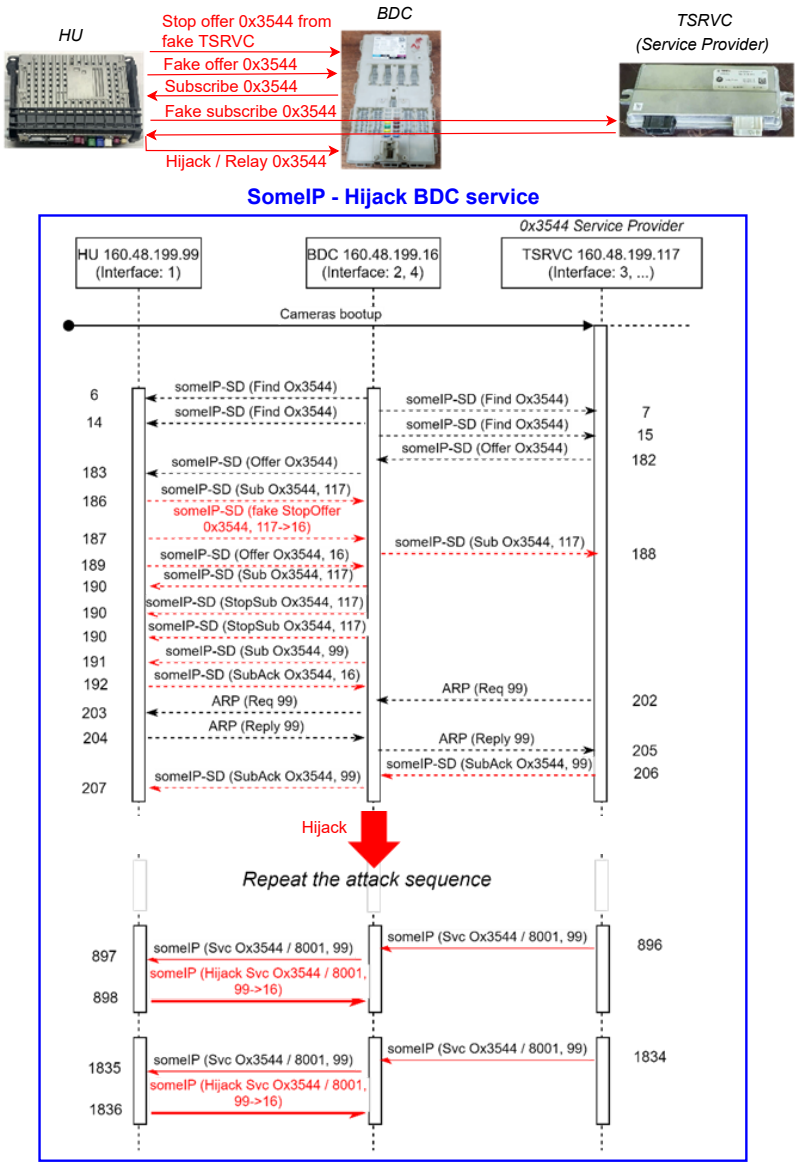


Fig. 28. Attack SomeIP Service 0x3544 Hijack

0x3531. As a result, HU cannot fake a fixed-service subscription from BDC. After hijacking the service from BDC to TSRVC using the same technique as 0x3544, HU then relays the changing subscription method ID received from TSRVC to BDC. Figure 29 illustrates that the subscription message received by HU from TSRVC via BDC is relayed to BDC. Subsequently when the service from BDC is received, HU relays the modified service to TSRVC via BDC.

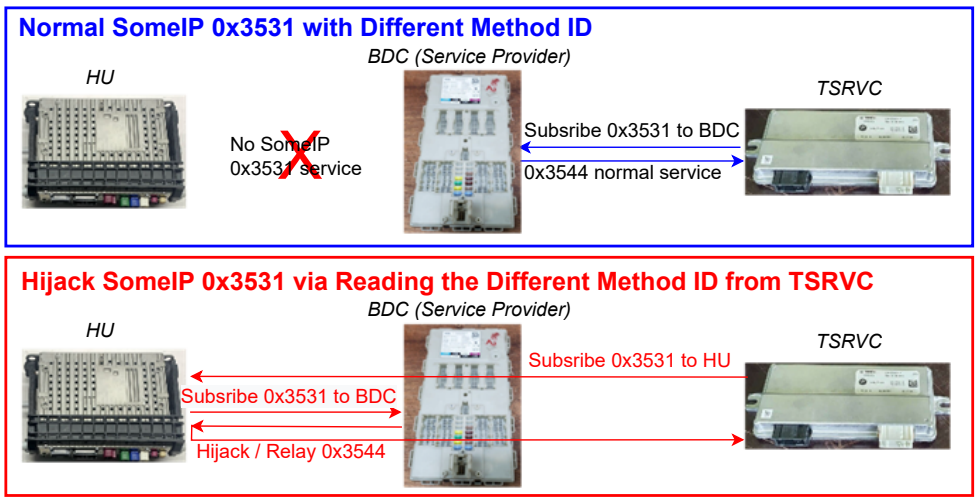


Fig. 29. Attack SomeIP Service 0x3531

The preceding two attacks target SOME/IP service (ID 0x3531 and 0x3544), which is a middleware protocol for Automotive Ethernet. In contrast to certain earlier works [2, 31] that conceptually discuss attacks on SOME/IP protocol model, we investigate and learn the realistic SOME/IP service flow within the AE network of AEVisionLab platform. This, in turn, allows us to design the preceding two attacks targeting specific IDs. Due to the proprietary nature of BMW protocol, we note that the concrete objective of the targeted service IDs (i.e., 0x3531 and 0x3544) is unknown. Nonetheless, the previous two attacks show that the real-world implementation of SOME/IP service in the IVN is vulnerable to both eavesdropping and hijacking. Hence, there is a need for more research in intrusion detection targeting these services, as also discussed in earlier work [4].

(v) UDP - Fake identity of TSRVC to Pcam. An initial attempt was made to fake the identity of HU as TSRVC to make the camera send video to HU. A bridge was set up between HU and BDC, and a fake UDP identity message was sent to Pcam via BDC and TSRVC. However, the attack failed because TSRVC did not forward the message to Pcam. Although the message passed from HU through BDC to TSRVC, it did not continue from TSRVC to Pcam. The exact reason for this behavior is unclear due to the black-box nature of TSRVC. We suspect that the TSRVC filters the messages intended for the Pcam or has two internal processing elements, separately for BDC and Pcam.

(vi) Attack success rate. To reliably evaluate the attack success, we launch the attacks multiple times and record the success rate. The results are recorded as follows.

(1) **Camera Delay Attack:** We perform several tests and observe on the workstation monitor that the Rcam video is always delayed by three seconds for each test.
(2) **Stop camera after delay:** This attack is performed several times with 100% success. At each test, Fcam will stop its video stream as frozen on the workstation monitor, while the other three cameras continue streaming.
(3) **SomeIP - Eavesdrop / Hijack BDC service:** We conduct several tests for each attack as shown in Table 7. Attack messages are crafted from the attacker (compromised HU) to the service provider (TSRVC) and the

Attack	Dataset File	Sets of Messages	Successful Attack	Result
SomeIP - Eavesdrop BDC service	20230524_HU_SvcEavesdrop0x3544.pcapng	17	17	TSRVC (160.48.199.117) sends service 0x3544 to both BDC (160.48.199.16) and attacker (160.48.199.99)
	20230524_HU_SvcEavesdrop0x3544-1.pcapng	15	15	
	20230530_allLinks_Attk0x3544Eavesdrop.pcapng	16	16	
SomeIP - Hijack BDC service	20230606_allLinks_Attk0x3544Hijack.pcapng	21	21	TSRVC (160.48.199.117) sends service 0x3544 to attacker (160.48.199.99) and then attacker sends to BDC (160.48.199.16)
	20230607_allLinks_Attk0x3544Hijack.pcapng	15	15	
	20230608_allLinks_Attk0x3544Hijack.pcapng	8	8	
	20230608_allLinks_Attk0x3544Hijack-1.pcapng	6	6	
SomeIP - Hijack TSRVC service	20230619_allLinks_Attk0x3531Hijack117.pcapng	5	5	BDC (160.48.199.16) sends service 0x3531 to attacker (160.48.199.99) and then attacker sends to TSRVC (160.48.199.117)

Table 7. SomeIP attack result

service receiver (BDC). For each test, the attacks are successful 100% in either eavesdropping or hijacking the service to BDC.

(4) **SomeIP - Hijack TSRVC service:** Similarly, the test result is shown in Table 7. Attack messages are crafted from the attacker (compromised HU) to the service provider (BDC) and the service receiver (TSRVC). The attacks are successful 100% in hijacking the service to TSRVC.

(5) **UDP - Fake identity of TSRVC to Pcam:** This attack intends to fool the camera into sending a video stream to the attacker (compromised HU). We notice that TSRVC sends UDP messages to inform Pcam on its IP address. Similar UDP messages are crafted from HU, but the IP address is changed to HU, instead of TSRVC. The message from HU has to pass via BDC, TSRVC and finally to Pcam. But the attack is not successful because TSRVC does not forward the message to Pcam..

6 Related Work

Effective cybersecurity testing of vehicles is essential for identifying and addressing security flaws, but testing real vehicles poses safety and economic risks. Consequently, researchers often use testbeds to uncover cybersecurity vulnerabilities. Shahid Mahmoo et al. [17] survey several automotive cybersecurity testbeds used by researchers for automotive networking technologies including Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST), and FlexRay. They categorized the testbeds into three types: simulation-based, hardware-based, and hybrid. Simulation-based testbeds use software to simulate ECUs and networks whereas hardware-based testbeds use real or emulated hardware, allowing for physical interaction studies. Finally, hybrid testbeds combine software and hardware components. VitroBench [29] is a hardware-based test platform using a COTS ECUs and it programs for controlling CAN network message packets. AEVisionLab employs similar concept of test platform as VitroBench by monitoring and manipulating the Automotive Ethernet messages of a testbed consisting of a car's COTS ECUs. However, in contrast to VitroBench, AEVisionLab focuses on the Automotive Ethernet Protocols

and their security. Additionally, AEVisionLab includes capabilities of video simulation and manipulation for security testing and evaluation.

There have been few Automotive Ethernet (AE) cybersecurity testbeds. Kai Muller et al. [19] demonstrate an Ethernet prototype platform using an ARM-centered system-on-chip for future automotive applications. Shane Tuohy et al. [27] developed a simulation platform integrating real video streams to test automotive video systems and ADAS algorithms. Hexuan Li et al. [12] use a Vehicle-in-the-Loop testbed and open-source Apollo [12] architecture to validate automated driving functions. These testbeds mainly focus on driving functionality rather than cybersecurity testing and evaluation.

Mostaq Hossain et al. [6] predict the increasing prevalence of Digital Twins in autonomous vehicles. A Digital Twin testbed is a virtual representation of a physical product containing information about the car, significantly aiding car design. Shetty [25] developed a digital twin model using a Toyota Prius equipped with various sensors to record driving scenarios. Popa et al. [22] developed CarTwin models in the laboratory to mimic real vehicle networks, using Infineon TC275 lite kits for CAN network communication. Their testings are using CAN protocol as the medium. Our work differs by focusing on the security testing of Automotive Ethernet protocols within in-vehicle networks (IVNs). Moreover, AEVisionLab uses COTS components for such testing, instead of focusing on the design and modeling of a digital twin.

In recent years, researchers have focused on the cybersecurity aspects of Automotive Ethernet (AE) networks. Seonghoon Jeong et al. [7] and ML Han et al. [5] look into intrusion detection systems using convolutional neural networks and wavelets, respectively. Rishikesh Kakade et al. [9] and Mahdi Fotouhi et al. [3] analyze vulnerabilities in time synchronization within AE networks. Zachariah Threet et al. [26] propose a Named Data Networking (NDN) security infrastructure to integrate protocols like CAN, LIN, and AE into a unified system. Wooyeon Jo et al. [8] developed an automatic whitelist generation system to filter abnormal packets, enhancing cybersecurity for in-vehicle networks (IVNs). Their test platforms typically involve simulations and embedded computers such as Raspberry Pi. In contrast, our AEVisionLab test platform allows researchers to test their solutions using real car ECUs in a laboratory setting. Additionally, AEVisionLab is also complementary to the research in intrusion detection systems (IDS), as the capabilities embodied in AEVisionLab can facilitate evaluation of IDS e.g., in terms of data collection (for training) and validation.

Fuzzing is widely used to identify vulnerabilities in automotive systems, including interfaces like Bluetooth, Wi-Fi, Ethernet, USB, and CAN. Pranav Patki et al. [20] use penetration testing and fuzzing on diagnostic services data to find and patch vulnerabilities, applicable to various interfaces. Yuekang Li et al. [13] develop a greybox fuzzer (Ori) for SOME/IP applications, demonstrating its efficiency in testing server programs and generating valid SOME/IP packets to explore deep paths. Feilong Zuo et al. [32] introduce PAVFuzz, a state-sensitive fuzz testing framework for protocols used in autonomous vehicles, which identified unknown vulnerabilities in protocols like RTPS and SOME/IP. Our work is orthogonal to the aforementioned approaches on fuzzing, as AEVisionLab provides a comprehensive platform for the development and real-world testing of fuzzing tools targeted for Automotive Ethernet protocols. This is also illustrated by the development of a proof-of-concept fuzzing tool on top of AEVisionLab.

7 Discussion and Future Outlook

AE will play a crucial role in the new generation automotive networks [14, 24]. Modern vehicles are increasingly complex and interconnected, which has expanded their attack surfaces. To this end, building on the previous standards

[12] https://github.com/ApolloAuto/apollo

of SAE J3061 and ISO 26262, the committee has created a new cybersecurity standard ISO/SAE 21434 [16][13] in 2021 to provide a framework for both cybersecurity process requirements and risk management. Cybersecurity standards and regulations, such as ISO/SAE 21434, highlight the necessity of cybersecurity verification and validation in the development lifecycle. Feng Luo et al. [15] survey cybersecurity testing for automotive domain. They address the importance of using testbeds for automotive cybersecurity testing to avoid economic and safety issues associated with testing on actual vehicles, though the results may differ from real systems. Security testing is now a mandatory phase in automotive development due to the ISO/SAE 21434, making it essential for type approval. In light of such development in automotive security, we present our outlook for using AEVisionLab in cybersecurity research.

7.1 Towards a flexible Automotive Ethernet test platform

Laboratory testing, unlike using a real car [11] which is subject to road elements, provides a controlled and safe environment. Test results can be analyzed offline, and further testing or changes can be promptly performed for validation. AEVisionLab offers more reliable test results compared to using simulated ECUs [3, 7–9, 26], reducing ambiguity regarding their validity on the actual car.

AEVisionLab offers flexibility for configuring an AE test platform in the laboratory to research car design, study vulnerabilities, and develop Intrusion Detection and Prevention Systems (IDPS). Due to the diversity of vehicle designs, a single testbed cannot cover all car types. To address this, we design AEVisionLab to allow easy replacement or addition/removal of ECUs without altering the rest of the platform, such as the communication capture module and software components (see Figure 2). This design ensures an extensible and flexible test using COTS ECUs, ensuring reproducibility of test results that closely mirror real car performance. This is ensured while maintaining the AEVisionLab capabilities, including bridging, sniffing, and fuzzing.

Without changing the test platform, a car designer has the flexibility to reprogram the same ECU with an improved version and test its functionality and vulnerabilities in an integrated test environment. The testing can be scaled up by performing incremental modification to the communication message using the workstation bridging before embarking on the entire ECU reprogramming. Any new IDPS module or algorithm can be injected to the network either physically or by software bridging via the workstation. Another example is that additional components such as internal and external antennas can be connected to BDC to study the wireless functionality of the car key.

With the advent of zonal architectures [14], the car backbone can utilize AE alongside multiple mixed controllers of AE, CAN, and LIN. Our prior work VitroBench [29] and AEVisionLab share similar bridging, sniffing, and fuzzing concepts. Thus, a hybrid car can be tested in the laboratory by setting up a single test platform that combines both AEVisionLab and VitroBench. Such a setup will enable integrated and synchronous monitoring of AE, CAN, and LIN messages, making the testing software more robust in checking the responses of all protocols and allowing for further investigation of cross-protocol attacks. We can integrate various types of IVNs into AEVisionLab and facilitate researchers to launch and evaluate cross-network attacks. The test platform can be expanded to multiple workstations, each connected to a communication module for accessing IVNs in different studies. These studies may include cybersecurity research, support for automotive software development, ECU testing and logging, and V2V/V2X wireless communication.

The application of AEVisionLab can be scaled up to study assisted or self driving function according to different road scenes. The simulation of a driving scenario using captured video from external display or injected video stream can be

[13] https://www.iso.org/standard/70918.html

extended to study different driving algorithms. The driving algorithm can be further developed either by reprogramming the ECU or by software bridging through the workstation.

7.2 Towards an extensible research platform for sensors-oriented driving

We demonstrate that AEVisionLab can monitor and manipulate video messages from cameras. With the increasing use of diverse sensors [14] to facilitate driving, more research into their design and cybersecurity is necessary. These sensors require a high-bandwidth network, and AE is suitable for supporting them. Similar to video messages, any sensor messages can be monitored or manipulated. Therefore, AEVisionLab is easily extendable for researching the security of such sensors. The test platform can study sensor responses via external live feed-in or internal modification of sensor messages.

For our evaluation, we present a road scene to a car camera and studied the resulting messages and the car's responses. We also investigated the manipulation of road scene and injecting such manipulated video into the AE network. To further research sensors-oriented driving, external live feeds can be sent to the car's sensors to simulate driving scenarios. This simulation does not affect the current monitoring software, and customized software can be programmed to manipulate the sensor messages.

Several add-on systems can scale up the test platform such as Active Cruise Control, Parking Manoeuvring Assistant, Driver Attention Camera and Driving Assistance. More ECUs (Table 8) can be added to AEVisionLab for different research purposes. Each capture module, CM1000 High MATEnet has 6 point-to-point 100/1000BASE-T1 connections. More connections can be added by cascading multiple capture modules.

Table 8. Sensors-Oriented Driving

System	Description	Main ECUs
Active Cruise Control (ACC)	Performs the task of acceleration and braking to regulate the distance and speed	ACC radar sensor and ACC control unit are one component, KAFAS camera, Steering Angle System (SAS), BDC
Parking Manoeuvring Assistant (PMA)	Park by controlling steering, acceleration, and braking after selecting a suitable space	PMA control unit, Electronic Power Steering (EPS), Dynamic Stability Control (DSC), Engine (DDE), instrument cluster (KOMBI), HU, BDC, 4 x radar detection sensors (i.e. front left, front right, rear left, rear right), TSRVC
Driver Attention Camera	Analyse the driver's attentiveness by evaluating the position of the head and eyes' opening from the instrument cluster	KOMBI, HU, BDC
Driving Assistance	Provide safety and comfort features like Lane Departure Warning, Forward Collision Warning, and Pedestrian Warning	KAFAS camera, BDC, 4 x radar detection sensors (i.e. front left, front right, rear left, rear right)

7.3 Towards directed fuzzing for AE network

In AEVisionLab, we created a random fuzzing algorithm to arbitrarily manipulate messages in AE IVNs, demonstrating its effectiveness in providing useful information to designers. Previous research [13, 20, 32] has also emphasized the importance of fuzzing for testing automotive systems. However, there is still significant work needed to develop

effective, targeted fuzzing algorithms specifically for automotive systems. Unlike application software, where test subjects are readily available, creating realistic test platforms for automotive systems remains challenging, hindering the advancement of security testing technologies in this field.

AEVisionLab facilitates the development and realistic evaluation of fuzzing research on AE IVNs by separating the automotive testbed design from the software component. This separation allows researchers to focus on software aspects, such as developing directed fuzzing algorithms aimed at uncovering vulnerabilities or attacks in IVNs. Moreover, AEVisionLab facilitates the development of complex fuzzing lgorithms that not only target IVN messages, but also the manipulation of videos from the road scene. While there have been several works in fuzzing image/video processing applications [21] as well as fuzzing IVN messages [13, 20, 32], the interplay between the IVN messages and video may be explored via advanced hybrid fuzzing techniques. In this context, AEVisionLab provides a foundation to advance the state-of-the-art in fuzzing algorithms for automotive systems.

7.4 Towards creating a research dataset for AE network

Since AEVisionLab is formed from the actual ECUs, the collected AE messages constitute the car network fingerprint (Section 5.3.1). Datasets of different normal driving scenarios can be collected. Attacks and video simulations as shown in the preceding sections, can be devised for dataset collection. This formed an important resource for further research and development of detection and prevention techniques, especially in the area of machine learning (ML) or deep learning (DL).

Moreover, we use Wireshark as a tool to monitor the network messages and collect them into a PCAPNG file when required. Dedicated software can also be programmed to extract specific information or features from the messages. We have demonstrated that using C programming for bridging and changing the message's data is possible in real time (Section 5.3.2). If real-time modification is not possible for more complex processing, the dataset can be collected for post-processing. The post-processed dataset can also be re-injected into the network if required, as shown in Section 5.3.3.

To illustrate the application of the datasets, we have collected a normal scenario, `20230517_allLinks.pcapng`, and an attack scenario, `20230608_allLinks_Attk0x3544Hijack.pcapng`. The attack is to hijack a SomeIP service (0x3544) to TSRVC, as described in Section 5.3.4. Further work can be performed to detect the attack using the two datasets. The datasets in this research can be obtained from AEVisionLab repository [14].

Along with this dataset, we have also included two software tools: *(i)* `pcap_videolinkV1.py`: this tool is used to view the video stream extracted from an AE link on the screen and extract the individual video frames to a local folder. and *(ii)* `pcap_ProInfoV1.py`: this tool extracts and decodes protocol information used in an AE link. Such information is stored in a text file attributed to each protocol. We hope the accompanying python tools can facilitate smooth usage of our datasets and benchmark existing and new attack performance using the data extracted for each protocols in an AE link. Additionally, such tools can also help in developing novel intrusion detection systems.

8 Conclusion

In this paper, we implement an Automotive Ethernet test platform, named AEVisionLab. AEVisionLab is a realistic embedded system that can be built using low-cost COTS ECUs. It can be used for security analysis of AE IVNs and protocols. The design can extend the test platform with different ECUs without altering the rest of the setup and the

[14] https://github.com/yeoant/AEVisionLab/tree/main/Dataset

software. It supports the development of advanced attacks, monitoring, and defense techniques without hardware changes. We have also curated numerous techniques to efficiently collect, manipulate, and inject videos of real road scenes.

We propose methodologies to reverse engineer and fingerprint proprietary AE protocols and demonstrate this capability by extracting and manipulating video feeds from the IVN of a BMW G20 car. In a nutshell, the embodiment of various capabilities in AEVisionLab allows us to curate attacks targeting both video and non-video messages within IVN, as demonstrated by manipulation/injection of video stream and hijacking/eavesdropping on SOME/IP services, respectively. We hope that AEVisionLab test platform will provide a foundation for future cybersecurity research of Automotive Ethernet. To promote further research in this area and for reproduction of results, we make all our code and experimental data available in the following URL:

https://www.aevisionlab.com

References

[1] 2023. ISO/IEC/IEEE International Standard–Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 1BA: Audio video bridging (AVB) systems. *IEEE Std 802.1BA-2021 (Revision of IEEE Std 802.1BA-2011)* (2023), 1–48. doi:10.1109/IEEESTD.2023.10108543

[2] Jinze Du, Rui Tang, and Tao Feng. 2022. Security analysis and improvement of vehicle Ethernet SOME/IP protocol. *Sensors* 22, 18 (2022), 6792.

[3] Mahdi Fotouhi, Alessio Buscemi, Abdelwahab Boualouache, Florian Jomrich, Christian Koebel, and Thomas Engel. 2023. Assessing the Impact of Attacks on an Automotive Ethernet Time Synchronization Testbed. In *2023 IEEE Vehicular Networking Conference (VNC)*. IEEE, 223–230.

[4] Tobias Gehrmann and Paul Duplys. 2020. Intrusion Detection for SOME/IP: Challenges and Opportunities. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 583–587. doi:10.1109/DSD51259.2020.00096

[5] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. 2023. TOW-IDS: Intrusion Detection System Based on Three Overlapped Wavelets for Automotive Ethernet. *IEEE Transactions on Information Forensics and Security* 18 (2023), 411–422. doi:10.1109/TIFS.2022.3221893

[6] SM Mostaq Hossain, Sohag Kumar Saha, Shampa Banik, and Trapa Banik. 2023. A new era of mobility: Exploring digital twin applications in autonomous vehicular systems. In *2023 IEEE World AI IoT Congress (AIIoT)*. IEEE, 0493–0499.

[7] Seonghoon Jeong, Boosun Jeon, Boheung Chung, and Huy Kang Kim. 2021. Convolutional neural network-based intrusion detection system for AVTP streams in automotive Ethernet-based networks. *Vehicular Communications* 29 (2021), 100338.

[8] Wooyeon Jo, SungJin Kim, Hyunjin Kim, Yeonghun Shin, and Taeshik Shon. 2022. Automatic whitelist generation system for ethernet based in-vehicle network. *Computers in Industry* 142 (2022), 103735.

[9] Rishikesh Kakade, Joey Chou, and Shannon Torcato. 2022. Vulnerability Analysis of Time Synchronization in Automotive Ethernet. *arXiv preprint arXiv:2208.11878* (2022).

[10] Tae Un Kang, Hyun Min Song, Seonghoon Jeong, and Huy Kang Kim. 2018. Automated reverse engineering and attack for CAN using OBD-II. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 1–7.

[11] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. 2010. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*. IEEE, 447–462.

[12] Hexuan Li, Vamsi Prakash Makkapati, Li Wan, Ernst Tomasch, Heinz Hoschopf, and Arno Eichberger. 2023. Validation of Automated Driving Function Based on the Apollo Platform: A Milestone for Simulation with Vehicle-in-the-Loop Testbed. *Vehicles* 5, 2 (2023), 718–731.

[13] Yuekang Li, Hongxu Chen, Cen Zhang, Siyang Xiong, Chaoyi Liu, and Yi Wang. 2020. Ori: A greybox fuzzer for SOME/IP protocols in automotive Ethernet. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 495–499.

[14] Lucia Lo Bello, Gaetano Patti, and Luca Leonardi. 2023. A perspective on ethernet in automotive communications—current status and future trends. *Applied Sciences* 13, 3 (2023), 1278.

[15] Feng Luo, Xuan Zhang, Zhenyu Yang, Yifan Jiang, Jiajia Wang, Mingzhi Wu, and Wanqiang Feng. 2022. Cybersecurity testing for automotive domain: A survey. *Sensors* 22, 23 (2022), 9211.

[16] Georg Macher, Christoph Schmittner, Omar Veledar, and Eugen Brenner. 2020. ISO/SAE DIS 21434 automotive cybersecurity standard-in a nutshell. In *Computer Safety, Reliability, and Security. Proceedings 39*. Springer, 123–135.

[17] Shahid Mahmood, Hoang Nga Nguyen, and Siraj A Shaikh. 2021. Automotive cybersecurity testing: Survey of testbeds and methods. *Digital Transformation, Cyber Security and Resilience of Modern Societies* (2021), 219–243.

[18] Mohamad Ali Mokhadder, Samar Bayan, and Utayba Mohammad. 2021. An intelligent approach to reverse engineer CAN messages in automotive systems. In *2021 IEEE International Conference on Electro Information Technology (EIT)*. IEEE, 1–7.

[19] Kai Müller, Till Steinbach, Franz Korf, and Thomas C. Schmidt. 2011. A real-time Ethernet prototype platform for automotive applications. In *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*. 221–225. doi:10.1109/ICCE-Berlin.2011.6031866

[20] Pranav Patki, Ajey Gotkhindikar, and Sunil Mane. 2018. Intelligent fuzz testing framework for finding hidden vulnerabilities in automotive environment. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. IEEE, 1–4.

[21] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *SOSP*. ACM, 1–18.

[22] Lucian Popa, Adriana Berdich, and Bogdan Groza. 2022. CarTwin—Development of a digital twin for a real-world in-vehicle CAN network. *Applied Sciences* 13, 1 (2022), 445.

[23] Donovan Porter. 2018. 100BASE-T1 Ethernet: the evolution of automotive networking. *Texas Instruments, Techn. Ber* 2 (2018).

[24] Michele Scalas and Giorgio Giacinto. 2019. Automotive cybersecurity: Foundations for next-generation vehicles. In *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*. IEEE, 1–6.

[25] SS Shetty. 2022. *Development of a Digital Twin of a Toyota Prius Mk4.* Ph. D. Dissertation. Eindhoven University of Technology Eindhoven, The Netherlands.

[26] Zachariah Threet, Christos Papadopoulos, William Lambert, Proyash Podder, Spiros Thanasoulas, Alex Afanasyev, Sheikh Ghafoor, and Susmit Shannigrahi. 2022. Securing automotive architectures with named data networking. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2663–2668.

[27] Shane Tuohy, Martin Glavin, Edward Jones, Ciaran Hughes, and Liam Kilmartin. 2016. Hybrid testbed for simulating in-vehicle automotive networks. *Simulation Modelling Practice and Theory* 66 (2016), 193–211.

[28] Lennert Wouters, Benedikt Gierlichs, and Bart Preneel. 2021. My other car is your car: compromising the Tesla Model X keyless entry system. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 149–172.

[29] Anthony Kee Teck Yeo, Matheus E Garbelini, Sudipta Chattopadhyay, and Jianying Zhou. 2023. VitroBench: Manipulating in-vehicle networks and COTS ECUs on your bench: A comprehensive test platform for automotive cybersecurity research. *Vehicular Communications* 43 (2023), 100649.

[30] Le Yu, Yangyang Liu, Pengfei Jing, Xiapu Luo, Lei Xue, Kaifa Zhao, Yajin Zhou, Ting Wang, Guofei Gu, Sen Nie, et al. 2022. Towards automatically reverse engineering vehicle diagnostic protocols. In *31st USENIX Security Symposium (USENIX Security 22)*. 1939–1956.

[31] Daniel Zelle, Timm Lauser, Dustin Kern, and Christoph Krauß. 2021. Analyzing and securing SOME/IP automotive services with formal and practical methods. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*. 1–20.

[32] Feilong Zuo, Zhengxiong Luo, Junze Yu, Zhe Liu, and Yu Jiang. 2021. PAVFuzz: State-Sensitive Fuzz Testing of Protocols in Autonomous Vehicles. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 823–828. doi:10.1109/DAC18074.2021.9586321

A AE Testbed Layout

Figure 30 shows the layout of the testbed components. Three ECUs are mounted on a test board:

- Headunit (HU)
- Body Domain Controller (BDC)
- Camera ECU (TSRVC)

The other four ECUs are connected externally:

- Front camera (Fcam)
- Rear camera (Rcam)
- Left side mirror, including driver camera (Dcam)
- Right side mirror, including passenger camera (Pcam)

All six ECUs' AE links are connected to the capture module (CM1000). The workstation sniffs the links from the capture module's monitoring port.

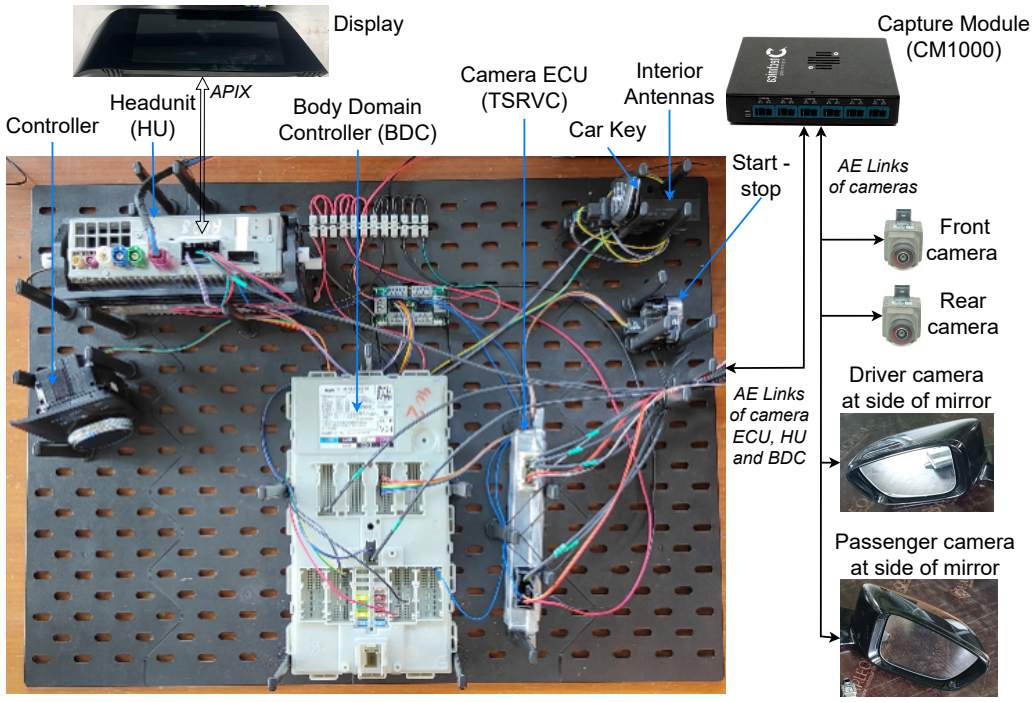


Fig. 30. AE Testbed Layout