Secured Service Oriented Communication in V2X Technology

OMAR HESHAM, ASHRAF SALEM AND BASSEM ABDULLAH
Computers and Systems Engineering,
Faculty of Engineering, Ain Shams University,
Cairo,
EGYPT

Abstract: - From the day communication started and messages being transmitted and received from one endpoint to another, security was always a concern. And nowadays this is not different from the old ages. In the era of IOT and V2X technology, communication protocols must be safe and secured and the traditional communication approaches like using CAN, LIN and Flexray will be hard to be integrated with IOT systems. Modern approaches are introduced for IOT integration through Ethernet, one of the most used approaches nowadays for vehicle communication is Service Oriented Communication which is represented by several protocols. One of protocols designed by AUTOSAR community is Service Oriented Middleware over Internet Protocol known as SOME/IP. After Surveying Service Oriented Communication, we propose a solution that adds security features to vanilla SOME/IP that prevents possible Denial of Service attacks. In this paper, we will survey the current solution used and tested and compare it with our proposal.

Key-Words: - SOME/IP, V2X, AUTOSAR, SOA, Adaptive/AR, DOS, IOT, in-vehicle communication security.

Received: August 13, 2022. Revised: September 25, 2023. Accepted: November 19, 2023. Published: December 31, 2023.

1 Introduction

In the modern era, digital communication has become the norm, and the risks of security breaches have only increased. confidential information. This can result in significant financial losses, identity theft, and damage to reputation.

In this paper, we will demonstrate the evaluation of security and its need nowadays in section 1.1 and will extend this to the Automotive industry in section 1.2. Also, will discuss service-oriented communication and its challenges in section 2. We will also propose a security solution for SOME/IP protocol in section 3 and discuss its results in section 4.

1.1 Background

Encryption is a vital component of modern communication. It involves the conversion of the message into a coded language that can only encrypt messages and use secure communication channels. It also means being aware of the risks and being vigilant about protecting sensitive information deciphered by the intended recipient. Encryption prevents unauthorized access to messages, ensuring that only the intended recipient can read them. That is the main components for security Integrity, Confidentiality, and Authenticity, and those components can be achieved by using the cryptographic primitives.

Integrity can be assured using hashing functions with it are different types like MD5 and SHA also by using Message Authentication Code also using Nonce Algorithm to prevent replay attacks. Confidently can be assured by using symmetric encryption like AES, DES, and Blowfish with blocking modes like ECB and CBC. Authentication be assured by using asymmetric encryption like RSA and Diffie Helman.

Digital certificates are critical components in the security of online communication, and they have become increasingly important in the era of information technology. Digital certificates provide a means of verifying the authenticity of websites and individuals, enabling secure communication through encryption.

Digital certificates are issued by Certificate Authorities (CAs), trusted third-party organizations that are responsible for verifying the identity of websites and individuals. CAs operate within a chain structure which is known as the Chain of Trust. Root CAs issue digital certificates to intermediate Cas which can generate certificates to another intermediate CA or the endpoint user.

Networking is an essential component of IOT and one of the modern technologies in IOT communication is low power wide area networks (LPWANs). The networks are designed to work on low power while connecting the IOT devices with acceptable efficiency and QoS and support different protocols used in different fields like LoRaWAN, Sigfox, and NB-IoT.

Another key development in modern IoT technology known as the third wave is the rise of edge computing. Edge computing involves processing data closer to the source, reducing latency, and enabling real-time decision-making. This is very important for applications that are safety-critical applications like autonomous vehicles for example. It also reduces the amount of data that needs to be transmitted to the cloud which improves network efficiency and reduces costs for the total computation. The connected topology of the devices makes them very vulnerable to several attacks that can affect the user himself and his data. To fix those problems a lot of IOT security technologies are now used to secure the users from those cyber-attacks.

Secured Elements (SE) is like a security shield for IoT devices that keeps sensitive information safe. There is a physical aspect of hardware that effectively stores sensitive data, e.g., as password and encryption keys so that they can't Let there be an interruption. SEs are often embedded in IoT devices, working behind the scenes to protect against theft attacks such as side-channel attacks. This attack tries etc. Surveillance and theft of sensitive information power or electromagnetic radiation output Through the machine. Hardware SE can be designed by the Trusted Platform Module (TPM) which is a standard for designing Hardware Security Modules and contains specifications for the hardware features and aspects like secure storage, secure booting, etc., [1]

To add an extra layer of security, network segmentation can be used. This means dividing the network into smaller subnetworks, each with its own set of rules and permissions. If an attack occurs, this can help limit the damage and prevent it from spreading to other parts of the network. If there is an attack, this can help prevent and prevent damage extending to other parts of the network. Also integrating the network with Intrusion Detection systems and Firewalls decreases the chances for the attacks to be successful.

Following the security standards while designing IOT systems should be considered as being up to date with new attacks and the attackers' methods. And for this sake, a community for security engineers has been made Open Web Application Security Project (OWASP). This community provides security engineers with the latest standards and also the known vulnerabilities which should be taken into consideration while designing and implementing the software of IOT devices.

The nature of IOT and their spread in many fields and places like homes, roads, etc. makes them a regular and obvious target to attackers and this nature can make the devices vulnerable to many attacks one of the most known attacks in IOT is DDOS.

Distributed denial-of-service (DDoS) attacks and man-in-the-middle attacks can affect Internet of Things devices. Several attacks were documented. For example, Mirai is a botnet that launches DDoS attacks on Internet of Things devices including routers, cameras, and DVRs. The Reaper botnet is a noteworthy example of an IoT assault. To take control of and add IoT devices to a botnet, the Reaper botnet takes use of security holes in such devices. The Triton malware, which attacked industrial control systems (ICS) utilized in critical infrastructure, was found in 2019. A Saudi Arabian petrochemical factory's safety systems were the target of malware, which resulted in an outage that forced the plant to close. The Ripple20 vulnerability, which impacted millions of Ethernet-enabled Internet of Things devices globally, was eventually found in 2020.

Vehicle-to-everything (V2X) communication refers to the technology that allows cars to talk to other cars, the infrastructure, and pedestrians. V2X has the potential to increase mobility, lessen traffic congestion, and improve road safety. It is an essential part of intelligent transportation systems (ITS). V2X and IoT are closely linked technologies because they both entail linking sensors and devices to the internet to provide real-time data interchange and decision-making. These capabilities can be further enhanced by technologies like artificial intelligence (AI) and machine learning.

IoT technology provides the necessary infrastructure for the V2X technology.V2X uses two main components, the first component is the onboard units known as OBUs which mainly exist on vehicles side. The second component is Roadside Units known as RSUs which exist on roads and traffic signals mainly.V2X also can utilize several communication protocols like Wi-Fi and dedicated short-range communication to transmit data between vehicles and infrastructure elements.

1.2 Securing Software in Automotive

Software is now one of the most important components in the manufacture of cars .it provides features such as infotainment systems, advanced driver assistance systems, and even autonomous driving capabilities. The software and its features now can be distinguished between the OEMs as there is not much progress in mechanical part in the industry unlike the software and electrical parts in the industry which has exponential growth. As software has become more common in automotive applications, it is important to ensure the software quality and reliability.

We classify Automotive Software into three categories: infotainment software, control software, and autonomous driving software. Infotainment software provides features such as audio and video entertainment, navigation, and communication systems as discussed in [2]. Control software includes systems such as engine control, braking systems, and transmission control. Autonomous driving software is used in self-driving cars to provide advanced driver-assistance systems and fully autonomous driving capabilities, [3], [4].

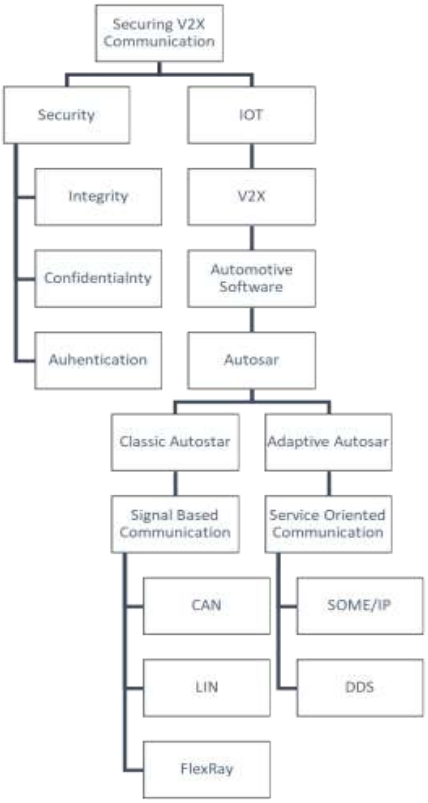

Fig. 1: Hiererchy of Software and Security in Automotive

Software in the Automotive industry is expensive and critical components. Problems and bugs in automotive software can be safety critical as they will threaten the vehicle passengers' lives and will cause OEMs financial losses due to the expensive recalls to maintenance centers. For those reasons, the automotive manufacturers apply quality and functional safety checks for the software used in automotives. Also, in 2003 the main manufacturers decided to put standards for the software to ensure its quality and reliability in the AUTOSAR standard.

AUTOSAR (AUTOmotive Open System Architecture) is an open-source standard for automotive software and its architecture by the main automotive manufacturers and suppliers. The standard is designed to ensure the quality and reliability of the software and it acts as a manifest between the software suppliers and automotive manufacturers. The first release was in 2003/2004 which is now called Classic AUTOSAR, and it is still maintained and updated and considered the main brick for automotive software. In 2017, another standard from the AUTOSAR community was released called Adaptive Autosar which is designed to work on High Performance ECUs but both standards Classic and Adaptive are still maintained as they have different purposes.

The communication stack in Classic AUTOSAR is considered one of its key components as it handles the communication between the ECUs. The standard includes several communication drivers like CAN, LIN, Flexray, Ethernet, etc. The driver standard for a certain protocol allows the car manufacturers to use any available driver implementations as long as it follows the AUTOSAR standard which increases the development stability and mobility.

In classic AUTOSAR, signal-based communication refers to the exchange of data between different software components within a vehicle using signals. These signals are defined in the AUTOSAR software architecture and are used to represent data values such as temperature, speed, and fuel level.

The signal-based communication is dedicated to run-time applications and safety-critical applications used in automotive, [5], that s' why it must be cost-efficient and the communication relationship between the ECUs must be defined and determined before the runtime (i.e., in the configuration) and can't be changed later, [6].

The communication between software components is achieved through the exchange of signals between the sending and receiving

components. The sending component publishes a signal with a specific value, and the receiving component subscribes to that signal. Whenever the value of the signal changes, the sending component sends an updated value to the receiving component.

When an Electronic Control Unit (ECU) wants to send a signal, it packages the data into a message and sends it over the communication bus. The receiving ECU then unpacks the message and updates its internal data structures with the new value of the signal.

The CAN protocol is nearly the most used protocol among classic AUTOSAR ECUs for signal-based communication and is used by a lot of applications and has several proposed gateways as discussed in [7], [8] and although there are modules in classic AUTOSAR for securing the on-board communication, some vulnerabilities were found, and some attacks were recorded on CAN bus.

Here is a brief list of successfully exploited CAN bus experiments.

- According to [9], a research group demonstrated their ability to manipulate the speedometer, disable the engine, and influence brakes by injecting messages into the CAN bus of a vehicle.
- The same research group effectively accomplished attacks remotely by exploiting communication interfaces. for example, the MP3 interpreter of the radio, the Bluetooth connectivity stack, and the telematics module.
- The researchers, [7], successfully proved the feasibility of remotely hacking an unmodified automobile.
- In 2016, the researchers, [8], successfully showcased the capability to remotely manipulate a Model S from Tesla in both Park as well as Drive mode using a laptop, thereby demonstrating the potential to do a range of physical actions.

Some approaches were introduced by [9], to improve signal-based communication, the security that can be applied on the CAN bus. We classify the methods and approaches to:

- Basic security symmetric encryption algorithms
  - Key distribution approaches
    * Using asymmetric encryption, i.e., using public and private keys. It may be not efficient on classic ECUs because the encryption and decryption need high computational resources, but still efficient during handshaking,[10], [11].
    * Using Short Term Keys, i.e., change the key every defined period. It will cause an overhead on the bus and decrease its actual throughput because of the cost of sending the keys as a separate message, [12].
- Physical Properties
  - By detecting clock drifts in transmission and applying mathematical functions to the electrical signal characteristics, researchers demonstrated that it was able to identify a fingerprint of the sender node, [13], [14].

At the current time we are heading to IOT and smart city technologies, V2X is becoming more important, and companies are becoming more interested in working on this topic. The AUTOSAR Working Group also established a standard in 2017 called Adaptive AUTOSAR (AR) which utilizes the High-Performance ECUs which are now more available and cheaper and can be used in automotive applications, so the vehicles can now fit in IOT systems.

Adaptive AUTOSAR -also known as “Adaptive Platform” consists of a group of functional clusters (can be called modules) that have two different types and functionalities, called foundation clusters and service clusters. The foundation function clusters like Persistency, Execution Management, and Communication Management function clusters provide basic functionality that is required by the AUTOSAR system, such as memory management, scheduling, and communication with hardware peripherals. Service clusters like Diagnostics, Update and Configuration Management, and Network Management provide higher-level services to the application software such as diagnostic services, update services, and signal to service mapping.

The Adaptive Platform must be based on a Linux kernel with a subset of POSIX 51 (PSE51) interface profiles, and this allows a variety of operating systems that have a Linux kernel to be a host to the Adaptive Platform. For example, there are Runtime and security optimized OSs that can host the Adaptive Platform like QNX by Blackberry which is known to have security utilities for automotive also an optimized Linux kernel to act like and RTOS, also another known OS that can host the platform is VxWorks which has low jitter and latency and can run on most of the hardware architectures, [15].

Adaptive AUTOSAR is based on Service Oriented Architecture (SOA) unlike the classic AUTOSAR which uses signal-based communication and uses relatively low-speed and data-limited buses like CAN, Lin, and Flexray. Adaptive AUTOSAR only uses Ethernet, which makes it much faster and more suitable for modern applications and a possible hierarchy for secured software communication is shown Figure 1, [16].

2 Service-Oriented Communication

The Service Oriented Architecture is more suitable for Remote Procedure Calls (RPCs) and has easier and more understandable user interface and is more suitable for high-performance ECUs especially that it uses Ethernet, [17], in communication which allows transmission of complex data like image data and easier to integrate with other end-user as Ethernet is nearly used everywhere and more spread than other protocols like CAN, LIN, and FLEXRAY, [6].

The AUTOSAR Working Group also published a new Service Oriented Protocol to fit with Adaptive AR called SOME/IP which stands for (Scalable Oriented MiddlEware over IP) but also Adaptive AUTOSAR can use other Service oriented protocols like DDS which stands for (Data Distributed Service), [16].

Data Distributed Service Protocol, known as DDS, is a standard released by Object Management Group (OMG) in 2004 at the beginning of the evolution of IOT technology to facilitate and standardize the communication between IOT applications, [18]. In 2015, DDS was modified to fit run-time applications. The DDS is a plug-and-play protocol that does not need any modifications or framework to operate and has built-in QOS and security features but does not support RPC, [19]. There are several available implementations for DDS like OpenDDS, FastDDS, and Connext DDS.

DDS implements the Service Oriented Architecture (SOA) using the Publish/Subscribe Model. There are three main components for the mode: Data Readers, Data Writers, and Topic. The applications can instantiate Data Writer to offer its service i.e., Publishing in the Topic and Data readers can receive data or request service i.e., Subscribing in the Topic. An Example of that is a motor, and we want to display its RPM on the digital cockpit. We will have a topic named RPM, a Publisher/DataWriter ECU that has a sensor to publish RPM value to the RPM Topic, and a Subscriber/DataReader in the digital cockpit ECU that subscribes to the RPM Topic and shows results to the driver. The configuration is represented in several XML files and an IDL file that contains the definition of the used data types within a topic, those files are passed to a code generator that writes the code of the data type in the required language C++ or C# or other languages supported by the DDS vendor, [20].

One of the key features of DDS is its support for Quality of Service (QoS) policies. QoS policies allow developers to specify the data characteristics being transmitted, such as reliability, durability, and priority. This allows developers to tailor the communication protocol to the specific needs of their application, ensuring that the right data is delivered to the right place at the right time, [18].

DDS has built-in security features that are represented as DDS plugins. Those plugins include Authentication, Encryption, Access Control, Key Management and Secure Communication, [18].

SOME/IP stands for Scalable service-Oriented MiddlewarE over IP which is a protocol that is used to communicate and exchange data between different Electronic Control Units (ECUs) in a vehicle. It is part of the AUTOSAR (Automotive Open System Architecture) standard and is designed to enable service-oriented communication between ECUs, [21].

The protocol is a lightweight protocol optimized specially to be used in the vehicles among its ECUs and standardizes the communication among them. The protocol design contains a lot of essential features like error detection, handling, and recovery mechanisms from network failures which can make the protocol more reliable, [21].

SOME/IP implements the Service Oriented Architecture (SOA) using the Server/Client Model. The server offers a Service Instance that implements a Service Interface. The client uses/requests a Service Instance. A Service is defined by its Service Interface, which may include Methods (with or without response), Events, and Fields. SOME/IP also has a Service Discovery feature that explicitly shows the status of service instances; how they can be reached is also used in publishing and subscription of services, so it is acting as a service registry, or a routing manager, [21]. SOME/IP has several implementations provided, like VSOME/IP and WRSOME/IP. In general, SOME/IP works on Ethernet supports TCP and UDP messages, and works on IPv4 and IPv6 as shown in Figure 2. Also,

some Adaptive AUTOSAR wrappers can be used SOME/IP for signal-based communication to be able to communicate with Classic AUTOSAR ECUs.

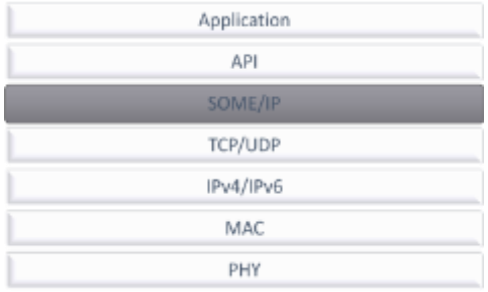

Fig. 2: SOME/IP in OSI

The basic components of SOME/IP according to SOME/IP standard, [21], are the Service registry, Service offerors, and Service requesters. A Service is defined by its Service Interface. This is comparable to a MOST Functional Block (FBlock) and may include:

- Request/Response Methods
- Events
- Fields

Service discovery is a crucial component of Some/IP, facilitating seamless communication and interaction between entities in a network. Through service registration, entities can make their services known to the network by providing essential information such as service ID, service type, and network address. This information is then stored in a service directory or registry, allowing other entities to easily locate and connect with the registered services. With dynamic discovery capabilities, services can be registered or deregistered at runtime, ensuring flexibility in the network. Entities can perform service lookups based on service IDs or types, retrieving the necessary information to establish communication with the desired service. The use of multicast or unicast communication further enhances flexibility, enabling efficient message delivery to multiple entities or directly to a specific entity. Service discovery in Some/IP not only enables entities to discover and connect with services but also supports additional management functionalities like service versioning and metadata exchange. Overall, service discovery plays a vital role in facilitating efficient and flexible communication within Some/IP-based networks.

2.1 Security Concerns in SOME/IP

Although SOME/IP is the most used service-oriented protocol in automotive, it does not contain any security concepts applied which makes it an overhead for its users to prepare a security layer for it especially since it does not fit with TLS / DTLS which can be suitable in other protocols as discussed in [9], and IPsec because one of its features is using multicast communications. The vanilla SOME/IP is vulnerable to attacks like intrusion most MMA attacks and DDoS attacks. Adaptive AUTOSAR introduces some security modules and features to protect the platform like CRYPTO, IDSM, and IAM, but the SOME/IP protocol itself is still vulnerable, [16].

2.2 Security Approaches in SOME/IP

The researchers, [9], introduced a framework for securing in-vehicle communication via SOME/IP. It was shown why traditional already implemented security add-ons like TLS/DTLS and IPsec are not suitable for SOME/IP.

The Threat model used was that the attacker gained access to the ECU by any means so it is not a part of the framework, but the framework can detect that the messages sent from the ECU are not legitimate anymore.

The securing framework consists of two phases with three layers of security. The first phase is the Session Establishment phase. The initial session establishment step occurs at startup between each application that wants to contact a SOME/IP service and the relevant offeror.

Asymmetric cryptography is leveraged to ensure that only authorized parties can start the communication, as well as exchange the data necessary for subsequent protection. The phase offers protection against replay attacks, as the nonce technique is used to ensure messages are not replayed. At the end of the phase, a secure channel is created to be used in the Run-time protection phase.

The second phase is the Run-time Protection phase. Using symmetric cryptography by the key that was exchanged during session establishment, this phase ensures run-time protection of message sending and receiving, as well as protection against Man in The Middle attacks.

There are three levels of security offered by the framework:

- NoSec Level
  - Vanilla SOME/IP (no security features).
- Authentication Level
  - Only Accept Messages / Offer / Request service from a legitimate source.
  - Protection against Replay-attacks.
- Confidentiality Level

  - Authentication Level PLUS symmetric encryption.
  - Ensures Authentication, Authorization, Integrity, and Confidentiality.
  - Prevent attackers from accessing message content.

For the authorization, The framework uses a White List Policy Technique similar to the approach used in [22], which is distributed among service requesters and offerors and the policy should follow this format: SOME/IP: app ID, service ID, service instance ID, role, min security level Each policy specifies that, provided the security level of the service instance is at least minSL, an application app is permitted to access it with a specific role. An infinite number of rules can be merged, ultimately specifying the complete range of communications that can take place between the apps housed in a vehicle. The framework additionally manages the secure dissemination of policies among the hosts, so there are two techniques used:

- Centralized
  - All the policies are stored in a centralized server, whenever a new connection is requested, the service provider connects to the server to check and validate the policy.
- Distributed
  - Only Related policies are stored in each ECU in the form of URI.
  - Installing or updating an application requires the creation of a new digital certificate containing only the revised rules relevant to that application, rather than regenerating the full rule set.

In the end, the framework can offer these security features to the SOME/IP protocol:

- Confidentiality of service key: An attacker cannot obtain or derive the symmetric key utilized to provide run-time protection.
- Server authentication: It won't be possible for an attacker to act to be the provider without being spotted by the requester.
- Client authentication: prevents unauthorized individuals from faking the requester and obtaining a service key.
- Observational equivalence: refers to the inability of an attacker to differentiate between a genuine message and a randomly generated message once it has been encrypted.

3 Built-in SOME/IP security for DOS Attacks

Service Oriented Architecture and Service Oriented Communication play a key role in IOT systems and V2X as most vehicles will have Adaptive AUTOSAR or any Linux-based platform in their Infotainment systems, so it is expected that SOA will be widely spread.

Also, the SOME/IP protocol by default as specified by AUTOSAR does not support any security features or any identity and access management features and this can be dangerous in the era of the IOT and V2X technology. Although SOME/IP can be integrated with TLS, DTLS as presented in [23], and IPSEC which will provide security features to the protocol, it will add constraints on the protocol to not use multicast which will not be suitable in most of the cases.

3.1 Threat Model

A possible threat model for a DOS attack is that the attacker could hijack one of the vehicle ECUs as shown in Figure 3 and start to flood the Ethernet network with dummy SOME/IP requests to one of the ECUs that has a critical functionality like controlling the digital cockpit which will cause the ECU to shut down or stall to respond the SOME/IP requests instead of dealing with real ECU requests as shown in Figure 4.

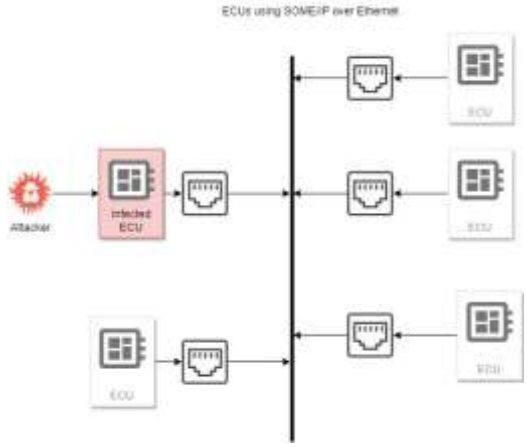

Fig. 3: One of the Vehicle ECUs Hijacked

3.2 Proposed Solution

The proposed solution follows the proxy server method for DOS attacks but with some modifications so it can be applicable and suitable for SOME/IP protocol. As shown in Figure 5, a proxy/guarding application monitors and organize the SOME/IP requests with a predefined policy

called DOS policy for approving or restricting the number of requests per service and per client to prevent service or network flooding as shown in Figure.

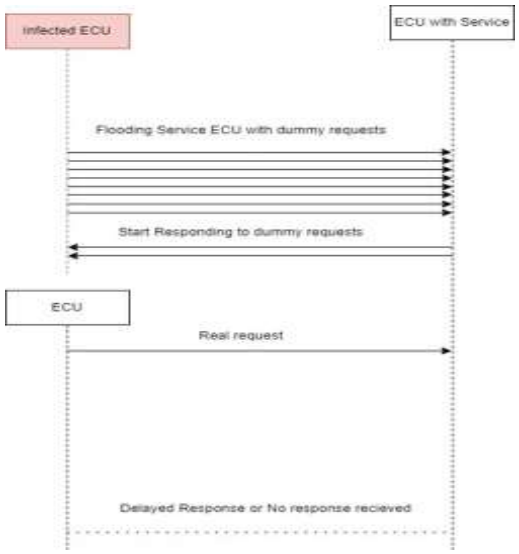

Fig. 4: Flooding network with SOME/IP requests

The algorithm is implemented as a plug-in in the SOME/IP instead of being a separate application and can be part of a plugin to the routing daemon in the SOME/IP as shown in Figure 6 which decreases the cost of integration for the end-user.

The DOS policy is implemented as a part of the SOME/IP configuration file, which is parsed at the initialization of the SOME/IP routing daemon process. The Policy is designed to maintain a certain number of sessions establishments and service requests for each client. The policy is represented as an object in the JSON configuration file, and it will contain the service ID, client ID, and number of requests allowed.

Algorithm 1 Guarding Application algorithm

function GuardAppInit
  $GuardECU \leftarrow ConfigurationFile$
  $GuardECU \leftarrow LoadsDosDefensePlugin$
  $GuardAPP \leftarrow ParsePolicy$
end function
function HandleAttackerRequest
  Reject *Request*
  Log the attack event
  Drop the suspicious message
  Disconnect *Client* Completely
end function
function HandleHealthyRequest
  Approve *Request*
  Send *Client* Approval
  Send Acknowledge *Server*
end function

Require: GuardECU starts up
Ensure: SOME/IP is up and running

while SOME/IP is UP **do**
  $GuardAPP \leftarrow ClientRequests$
  if $Client \notin ListOfClientsInPolicy$ **then**
    $HandleAttackerRequest$
  else
    if $ClientRequest \not> PolicyRequests$ **then**
      $HandleHealthyRequest$
    end if
  end if
end while

The algorithm is represented as pseudocode in 1.

In Figure 5 Policy Method is used and can be implemented as a plug-in in the SOME/IP instead of being a separate application and can be part or a plugin to the routing daemon as SOME/IP provides the ability to add customized plugins to the protocol. The DOS policy can be part of the SOME/IP configuration file, which is parsed at the initialization of the SOME/IP routing daemon process.

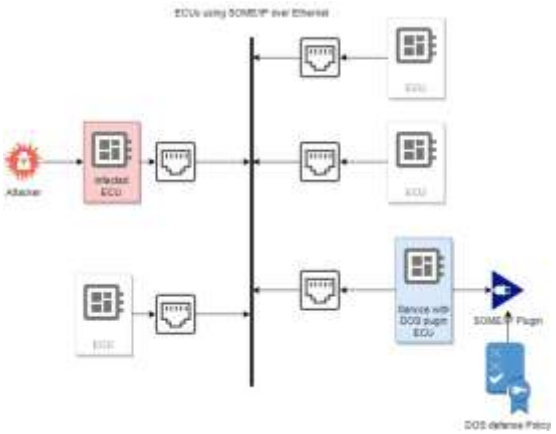

Fig. 5: Adding SOME/IP DOS plugin

In Figure 6 an example of an attack is shown were the attacker gained access to one of the vehicle

ECUs and started to flood the network with SOME/IP requests but in this case, the analytic or the Policy service ECU realizes the anomaly and the requests flooding and starts to reject the attacker requests and starts to respond to other ECU real requests.

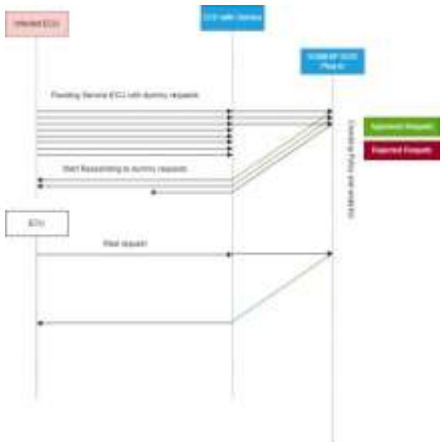

Fig. 6: One of the Vehicle ECUs Hijacked

3.3 Experimental Evaluation

For Experimental Evaluation, Docker Containerization technology is used for representing five ECUs each of them has an application that uses SOME/IP and has SOME/IP daemon running. A Guarding ECU is also represented in a Docker container with a configuration file and DOS policy file for DOS Guard use. This also can be shown in 5.

The Experiments and tests showed that there is no payload size increase as we did not add any headers to SOME/IP packet, unlike TLS, DTLS, and IPsec solutions.

The Experiments showed that the solution used is effective in securing the protocol against DOS attacks, but the performance is affected than using the vanilla SOME/IP comparing the processing request time to the traditional methods mentioned above showed that performance is nearly similar to using TLS and faster than using IPsec when taking reference results from, [9].

The algorithm is tested with several policies and shows efficiency against several attacks from several ECUs. However there is a limitation in the configuration due to the internal implementation of protocol itself. The obvious limitation is not to set the possible number of requests to be greater than 1024 as the request buffer in the protocol itself is set to this number, so setting a policy greater than this number will not protect against attacks and will cause the buffer to overflow.

4 Discussion

One of the withdrawals that used protocols used in the Service Oriented Communication is that they are not secured by nature and a framework will always be needed to fill the gaps. In DDS, we can find multiple commercial implementations that support QoS which adds some security to the protocol, but the protocol is still vulnerable as some attacks were recorded recently and the same for the SOME/IP protocol. Although researchers, [9], introduced a framework for securing the SOME/IP protocol and showed better performance than the integration with TLS, DTLS, and IPsec and nearly no constraints the protocol, the protocol is still vulnerable in the case of Denial-of-Service Attacks which can cause the applications using the protocol to stall or crash, [9].

The plugin we proposed is supposed to pave the way for future improvement for the SOME/IP protocol specifications. The current specifications do not support security features, but the protocol is now turning in high demand. So, adding the framework by [9], and the plugin for DOS attacks will be a good starting point for improving the SOME/IP.

Also, other methodologies for securing the In-vehicle communication are using ML and Deep learning techniques e.g. Using Anomaly Detection Mechanism using a pre-trained model using the data recorded on buses. This will prevent multiple attacks like Intrusion and DoS attacks. One of the obvious problems for this solution is finding a sufficient amount of data for training the models.

For the simulation and testing environment, several options are suitable away from the hardware. Using Hardware for five ECUs is a high cost, which is why we headed towards software simulation and Emulations. One of the proposed solutions is using the YOCTO project, [24], with QEMU emulators or QNX. The YOCTO project takes time to build images and deploying them on QEMU emulators also several problems will be faced during setting up the network interface, and it will not be very practical to take long build times for single changes during testing. Also, QNX is a really good solution, but unfortunately, it is not available as an open-source and only available for commercial use. In the end, choosing dockers was a good solution as setting the container with the suitable run-time options and parameters will give the same results as using the YOCTO project but with fast build and deployment time.

5 Conclusion

In this paper, we presented the securing of automotive software methods and surveyed the approaches and security features as well as the possible security concerns for signal-based communication and the service-oriented communication protocols. We found an open area for research to secure SOME/IP as vanilla SOME/IP is vulnerable to DOS attacks and in Section 3 we proposed a methodology and architecture to protect SOME/IP against DOS attacks and the next research work will focus on the implementation and testing the proposed solution and its impact on the protocol performance. The impact of the plugin in real life is important as it protects the communication of ADAS systems which are used in most modern vehicles. For future work, it should focus on integrating the secured SOME/IP with the Adaptive Autosar Platform and using its security solution functional clusters like CRYPTO, IDSM, and IAM modules to make the Service-Oriented Communication on the platform safe and secured. Also, exploring the possibility of using anomaly detection mechanisms using ML for securing SOME/IP protocol when sufficient data is available for training.

References:

[1] E. W. W. X. S. S. S. Liu, K. Zeng and Apr, "Hardware module-based message authentication in intra-vehicle networks," in *Proc. ACM/IEEE 8th Int. Conf. Cyber-Phys. Syst*, 2017.

[2] G. Meixner and C., Múller. *Automotive User Interfaces: Creating Interactive Experiences in the Car,* 2017, Springer, https://doi.org/10.1007/978-3-319-49448-7.

[3] H. W. S. H. F. Lotz and C., *Singer Handbook of Driver Assistance Systems: Basic Information Components and Systems for Active Safety and Comfort,* 2015.

[4] O. B. J. D. G. Doll and C. Knochenhauer, "Rethinking car software and electronics architecture" McKinsey Co," *Feb,* 2018.

[5] J. H. K. S. S. N. H. B. M. C. Y. S. Lee and W. J., "Jeon "Gateway framework for in-vehicle networks based on CAN FlexRay and Ethernet," *IEEE Trans. Veh. Technol,* vol. 64, p. 4472–4486, October 2015.

[6] S. T. M. G. C. H. E. J. M. Trivedi, L. Kilmartin and Apr, "Intra-vehicle networks: A review," *IEEE Trans. Intell. Transp. Syst,* vol. 16, p. 534–545, 2015.

[7] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," in *Proc*, Black Hat USA Aug, 2015.

[8] S. N. L. Liu and Y. D. "Free-fall, "Hacking tesla from wireless to CAN bus," in *Proc*, Black Hat USA, 2017.

[9] M. I. A. B. M. R. F. R. R. Sisto and F. Valenza, "Protecting in-vehicle services: Security-enabled SOME/IP middleware," *IEEE Veh. Technol. Mag,* vol. 15, p. 77–85, September 2020.

[10] B. Groza and P., "Murvay "Security solutions for the controller area network: Bringing authentication to in-vehicle networks," *IEEE Veh. Technol. Mag,* vol. 13, p. 40–47, March 2018.

[11] S. Nürnberger and C. Rossow, "– vatiCAN – vetted authenticated CAN bus," in *Proc. Int. Conf. Cryptographic Hardware Embedded Syst*, 2016.

[12] D. P. N. A. A. T. Arul and S. Katzenbeisser, "Using implicit certification to efficiently establish authenticated group keys for in-vehicle networks," in *Proc. IEEE Veh. Netw. Conf*, 2019.

[13] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *Proc. Int. Conf. Cryptographic Hardware Embedded Syst*, 2016.

[14] K.-T. Cho and G. K., "Shin "Fingerprinting electronic control units for vehicle intrusion detection," in *Proc. 25th USENIX Conf. Secur. Symp*, 2016.

[15] M. Oertel and B. Zimmer, *E/E architectures with AUTOSAR adaptive,* 2019, [Online]. https://cdn.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Adaptive_Architecture_ATZ_201905_PressArticle_EN.pdf (Accessed Date: February 10, 2024).

[16] AUTOSAR group, "Autosar Explantory Document", [Online]. https://www.autosar.org/fileadmin/standards/R21-11/CP/AUTOSAR_EXP_ClassicPlatformARTI.pdf (Accessed Date: February 10, 2024).

[17] "IEEE standard for Ethernet amendment 1: Physical layer specifications and management parameters for 100 Mb/s operation over a single balanced twisted pair cable (100BASE-T1," in *IEEE Standard 802.3bw-2015 (Amendment to IEEE Standard 802.3-2015*, 2016, p. 1–88.

[18] OMG group, *DDS security (Version 1.1),* 2018,

[Online]. https://www.omg.org/spec/DDS-SECURITY/1.1/About-DDS-SECURITY (Accessed Date: February 10, 2024).

[19] OMG group, *Remote procedure call over DDS (DDS-RPC) (Version 1.0),* 2017, [Online]. https://www.omg.org/spec/DDS-RPC/1.0/About-DDS-RPC (Accessed Date: February 10, 2024).

[20] OMG group, *The real-time publish-subscribe protocol DDS interoperability wire protocol (DDSI-RTPS) specification (Version 2.3),* 2019, [Online]. https://www.omg.org/spec/DDSI-RTPS/2.3/Beta1/PDF (Accessed Date: February 10, 2024).

[21] AUTOSAR group, *SOME/IP protocol specification,* 2016, [Online]. https://www.autosar.org/fileadmin/standards/R22-11/FO/AUTOSAR_PRS_SOMEIPProtocol.pdf (Accessed Date: February 10, 2024).

[22] M. H. M. Nolte and V. Prevelakis, "A framework for policy based secure intra vehicle communication," in *Proc. IEEE Veh. Netw. Conf*, 2017.

[23] D. Z. C. K. H. Strauß and K. Schmidt, "On using TLS to secure in-vehicle networks," in *Proc. 12th Int. Conf. Availability Rel. Secur*, 2017.

[24] O. Salvador and D. Angolini, Embedded Linux Development Using Yocto Project: Leverage the power of the Yocto Project to build efficient Linux-based products, 2023.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.