TEEm: A Tangle-based Elastic Emulator for Storing Connected Vehicle Data in a Distributed Ledger Technology

David Werden[a], Matthew Muccioli[b] and Anyi Liu[c]
School of Engineering and Computer Science, Oakland University, Rochester, MI, U.S.A.

Keywords: Automotive Cybersecurity, Autonomous Vehicles, CAN Bus, Distributed Ledger Technology, Hornet, Privacy, Road-Side Units, Tangle.

Abstract: This paper describes TEEm, a Cyber-Physical test-bed that emulates the data exchange of in-vehicle network communication between multiple vehicles. In particular, TEEm leverages the Distributed Ledger Technology (DLT) as the fundamental technology for data storage and exchange. TEEm uses a private Tangle instance and is extensible, thus we refer to this testing environment as the Tangle-based Elastic Emulator, or TEEm. To mimic realistic in-vehicle network traffic, we use both hardware emulation as well as software containers to replicate vehicles with Electronic Control Units (ECUs). TEEm seamlessly pushes in-vehicle network traffic to an IOTA private Tangle Hornet. Our implementation and evaluation demonstrate the feasibility of applying the DLT in building the shared storage, authenticating vehicles, and effectively retrieving a wide range of data generated by ECUs and other in-vehicle sensors. TEEm holds a great potential to coordinate with other emerging technology, such as Deep Learning and Edge Computing.

1 INTRODUCTION

This paper presents the creation of a hybrid virtual and physical environment, known as the Tangle-based Elastic Emulator (TEEm). TEEm is used for the testing of storing Controller Area Network (CAN) bus message data and connected application data in a Distributed Ledger Technology (DLT), specifically a Private Tangle network, known as Hornet (IOTA, 2021a).

TEEm demonstrates how CAN bus message data, along with other types of data, is easily stored in the Hornet instance, as well as the relatively simple methods for querying that data (IOTA, 2021a). Also discussed in (IOTA, 2021a) is the extensible nature of TEEm that allows for the easy addition of other types of data interfaces and networks. Finally, we present a use case describing how Hornet can be used in a VANET to help detect Sybil attacks.

[a] https://orcid.org/0000-0003-0038-1694
[b] https://orcid.org/0000-0002-0822-5007
[c] https://orcid.org/0000-0002-6011-1998

2 TEEm OVERVIEW

Despite being developed in 1986 by Robert Bosch, GmbH (Bozdal et al., 2018), the CAN bus protocol was not actively researched and targeted for cybersecurity attacks until the early 2000's (Security, 2020). This research started to get more active after 2010, and was punctuated by the dramatic research conducted by Miller and Valasek when they demonstrated the ability to hack a vehicle in St. Louis, MO over the internet from Pennsylvania (MIller and Valasek, 2015). Upstream Security's report (Security, 2020) states that there has been a rise of 99% in reporting vehicle cybersecurity incidents from Year 2018 to Year 2019. It also states that 57% of incidents in Year 2019 were criminal in nature in which most common attack vectors were remote/keyless entry, database servers, mobile apps, and ODB-II port attacks. Recent increases in research have not been limited to cybersecurity attacks and mitigations. The CAN bus messaging protocol itself has also seen a increased rise in research efforts. The CAN Bus protocol uses many different proprietary types of messages to communicate between ECUs in a vehicle. The authors of (Huybrechts et al., 2017) demonstrate that the the CAN message identification can be different between even vehicles of the same manufacturer. This means that

Werden, D., Muccioli, M. and Liu, A.
TEEm: A Tangle-based Elastic Emulator for Storing Connected Vehicle Data in a Distributed Ledger Technology.
DOI: 10.5220/0010990100003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 230-241
ISBN: 978-989-758-553-1; ISSN: 2184-4356

knowing what CAN messages to monitor for detection purposes, or what CAN messages may be used to cause affects on the vehicle, are different between vehicles and must be identified for each instance. Presented in (Huybrechts et al., 2017) is the author's tool, which they have named CANHUNTER. Finally, because TEEm is not limited to the storage of only CAN bus message data, it is worth noting that the different IoT apps and devices, such as In-Vehicle Infotainment (IVI) systems have also in recent years become a research focal area. The authors of (Mazloom et al., 2016) discuss their analysis of IVI platforms and vehicle-connected IoT applications.

The technological advancements and varying connectivity options do more than provide the driver and passengers with comfort, custom playlists, or hands-free driving. These increases in technology and connectivity options also increase the attack surface that vehicles present to today's attacker.

2.1 Key Features of Proposed Technology

TEEm is a framework that systematically addresses aforementioned security challenges and technical deficiencies.

First, to removing the risk of a single point of failure (SPOF), TEEm stores in-vehicle network communication traffic in a distributed ledger, namely the Hornet (IOTA, 2021a). Hornet's storage is highly distributed, such that it allows snapshots of the storage to be kept validly across the distributed ledger. Thus, it does not suffer from exploits that try to compromise the central storage of the in-vehicle network communication traffic, unlike recent database hacks of Marriott, Facebook, and other large companies. (Chua et al., 2021).

Second, to ensure data integrity and authenticity, TEEm leverages Hornet transactions to first store and then hash the in-vehicle data in its database. Thus, any attempt to modify or delete data in Hornet is detectable and not accepted by Hornet

Third, TEEm's extensible nature and usage of Docker Containers (Docker, 2021a) to emulate vehicle and ECU actions allow for the safe testing of prototype security functions, such as a CAN-based Intrusion Detection System (IDS) as well as the ability to model or baseline specific ECU behavior.

Fourth, the cyber-physical design of TEEm allows for testing of potential threat vectors such as pivoting from a Cellular interface to the CAN bus, as Miller and Valasek demonstrated in (MIller and Valasek, 2015).

Fifth, TEEm is configured to provide the user with easily accessible metrics such as system performance, Hornet performance, and CAN bus performance and load.

Sixth and finally, TEEm allows for the ability to create other Hornet instances and to move Hornet Nodes dynamically between instances.

2.2 Contributions of Proposed Technology

TEEm is expected to make the following contributions:

- TEEm constructs a framework that emulates the in-vehicle network traffic of ECUs. The behaviors of each ECU are emulated via Docker containers. We also include Arduino-based physical components, leveraging specific target processor memory requirements, such that TEEm emulates physical and virtual ECUs to reproduce realistic traffic patterns.
- TEEm's easily captured metrics allow for a cybersecurity engineer to quickly test firewall rules, such as rules designed to identify and potentially stop actions such as Denial of Service attempts.
- TEEm's virtual portion and physical connections make it an ideal environment for the prototyping of IDS technology. The virtual portion allows for the quick configuration of virtual vehicles, to include the limiting of access to specific ports and/or protocols. Thus, a software based CAN bus IDS can be connected to TEEm's active CAN bus interface, allowing the IDS and its ruleset to be tested in a safe, contained, and measurable environment. One example of this would the testing of rules for rate limiting on messages. To verify that an IDS ruleset of this nature is working, traffic can be specifically and easily generated on the virtual CAN interface. Additionally, the built-in OS tools as well as the Python language and libraries provide mechanisms to verify the rule (e.g., the cansniffer tool in conjunction with a small python script can be used to determine how often a given ArbID is seen). The physical side allows for the same testing as the virtual portion with the addition of potentially injecting latency and noise.

2.3 Structure of This Paper

The remainder of this paper uses the following structure. Section 3 presents details of technologies used for the Tangle-based Emulator described in this paper, Section 4 describes the design and setup of the Tangle-based Emulator system, and Section 6 presents the evaluation of the Tangle-based Emulator, Section 7 discusses other works related to this effort, and in Section 8 we present a basic use-case for autonomous vehicles storing and accessing data from the Tangle network.

3 BACKGROUND

This section presents some of the background regarding previous research of DLTs in vehicular security.

3.1 Previous Research of DLT in Vehicular Security

Some of the earliest research into the usage of DLTs in the Vehicle and Vehicle Security domains was related to the automotive insurance industry and to localization privacy.

As the authors of (Huybrechts et al., 2017) discuss, there are a large variety of OBD-II dongles provided by the vehicle manufacturers as well as 3rd-party suppliers. These dongles provide services centered around automotive insurance services and vehicle diagnostic information. As (Wen et al., 2020) shows, these dongles are normalized to connect to the OBD-II port and provide access to the CAN bus via Wi-Fi, BlueTooth, or cellular networks. In (Miller and Valasek, 2013) and their follow-up work, (Miller and Valasek, 2014), the authors explore how to use both the ODB-II port directly as well as through a wireless dongle interface, to issue vehicle control commands on the CAN bus. In 2009, the authors of (T. W. Chim and Li, 2009) presented a scheme for addressing the security and privacy issues inherent in VANETs. The authors of (M. and Lee, 2011) present PPAS, a Privacy Preservation Authentication Scheme for V2I communications, building in part upon the 2004 work found in (Hubaux et al., 2004). The idea of using localization nodes within a VANET while still preserving privacy was presented as the L-P2DSA scheme by the authors of (Mekliche and Moussaoui, 2013) in 2013. In 2015, we again find research related to location privacy protection presented by the authors of (Tyagi and Sreenath, 2015) and in 2018 a new type of attack, using trajectory and route planning of autonomous vehicles, was presented in (Banihani et al., 2018), where they make the claim that, at the time of their writing, over a decade of research into localization and privacy issues had already been conducted. However, research into the use of DLT in regards to vehicle security and privacy concerns started in 2017 and specific examples of this research are presented in Section 7.5.

4 SYSTEM DESIGN

In this section, we first present an overview of the architectural design of TEEm and the justification for each of three primary components. Next, we discuss specifics regarding each item of hardware and software items used. We then present how we have set up and configured the TEEm networks, Hornet, CAN bus functionality, and the necessary Docker images. Finally, we present how the ECU containers are run, the creation of the Arduino-based emulators, and storing CAN bus message data in Hornet.

4.1 System Architecture

In Sections 1 and 2 we presented the purposes of TEEm: 1) to test the storage and retrieval of CAN bus messages and other vehicle network data in a DLT, 2) the ability to query and use the data stored within the DLT, and, 3) the ability of the DLT to protect stored data, such as PII or billing information. In order to accomplish those purposes, TEEm was designed, from a high-level architecture perspective, using four primary components. These components are depicted in Figure 1 and discussed below.

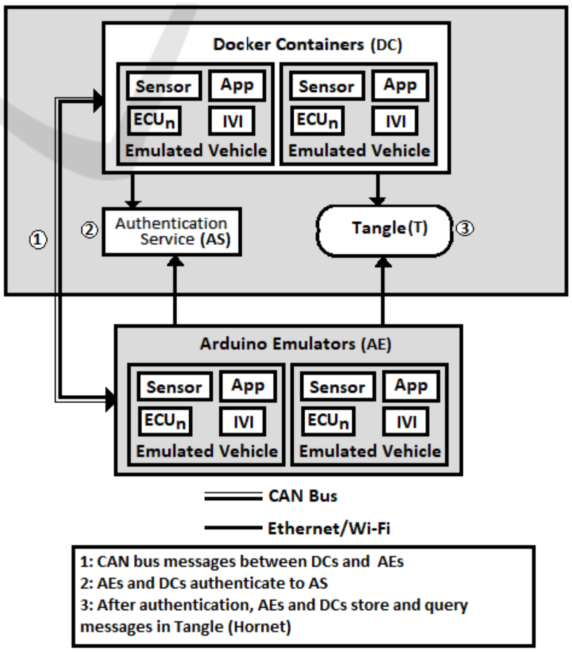

Figure 1: TEEm System Design Architecture.

4.1.1 Emulated Vehicles

TEEm's emulated vehicles are identified in Figure 1 as the Docker Containers (DCs) and Arduino Emulators (AEs). These emulators are configured with four ECU Arbitration IDs (ArbIDs) each as well as other functionality such as emulated battery state of charge (SOC) data.

The DCs are executed from a Raspberry Pi 4B, which also contains the Authentication Service (AS) and the Tangle (T) instance (Hornet). Excluding peripherals and the USB2CAN converter, no modifications were made to the standard hardware configuration. Using the DCs, it is trivial to introduce other types of emulated data producers and consumers and to test their interaction with the Hornet. One example of this was the testing of sending battery SOC data to the Hornet and querying that data.

The physical autonomous vehicle clients, the AEs, are comprised of Arduinos and Arduino CAN shields. The physical bus, external to the Tangle Server component provides extensibility of the number of devices on the CAN bus. By using low-cost hardware like Arduinos, a range of vehicle sensor functions and data can be more quickly and accurately emulated.

By using an unmodified Raspberry Pi 4B, we are able to identify performance issues by examining the performance metrics of the Tangle instance itself as it runs within the same physical environment as the primary Operating System (Ubuntu 20.04) and the different TEEm-required networks.

4.1.2 Tangle

Different distributed storage technologies were examined while researching options for TEEm's design. Primarily examined were Blockchain instances, Distributed Hash Tables (DHTs), and the private Tangle instance Hornet. Based upon ease of implementation the decision was made to use Hornet.

4.1.3 CAN bus

The CAN bus section of TEEm is comprised of a single physical and virtual CAN bus. The system component that contains the Tangle instance makes use of an internal Ethernet as well as an internal, virtual, CAN bus. This virtual CAN bus is extended, via a CAN to USB converter (USB2CAN) in order to interact with the physical CAN bus portion of TEEm.

The networks used by TEEm, including the CAN bus, are depicted and discussed in Figure 2 and discussed in Section 4.3.

4.1.4 Authentication Service

The Authentication Service (AS) of TEEm is envisioned to be the manager of authentication an emulated vehicle to the RSU. This service would provide a newly authenticated vehicle with the required security token and connection information to the Tangle instance.

4.2 Environment Setup

This section presents a discussion of the hardware and software that was used for each primary TEEm component. We will first articulate the physical components and then the software components.

4.2.1 Physical Components

In addition to hosting the Tangle instance, the Raspberry Pi 4B (RPi4B) was chosen for usage because it provides a programmable interface to interact with the Arduino-based ECU Emulators, allows for capturing of CAN bus and Ethernet traffic, and supports the ability to generate metrics specific to both the CAN bus and Hornet traffic.

As discussed above, TEEm makes use of InnoMaker's USB2CAN (InnoMaker, 2021). This converter is used to connect the physical CAN bus used by the Arduino-based ECU emulators to the virtual CAN bus of the RPi4B.

4.2.2 Software Components

Ubuntu 20.04 Server was chosen as the operating system (OS) for the RPi4B. This choice was initially arbitrary but proved to be fortuitous due to some of the complexity of the Hornet setup as well as the mature support for Docker and CAN bus within Ubuntu.

As previously mentioned, TEEm makes use of Docker containers to emulate vehicles with ECUs. In order to use Docker containers, both Docker Engine and Docker Compose are required (Docker, 2021b).

The default *can-utils* package from the OS repository is used by TEEm for all CAN bus activities and metrics within the OS. The *can-utils* package's primary function within TEEm is to provide for and support the creation and configuration of the virtual CAN bus on the RPi4B. The *can-utils* enables the use of the USB2CAN adaptor for connecting the virtual CAN bus on the RPi4B to the physical CAN bus that the Arduino-based ECU Emulators are connected to, allowing for all the Docker Container-based vehicles to communicate with the Arduino-based ECUs.

The *can-utils* package also provides tools for generating (*cangen*), sniffing (*cansniffer* and *candump*), and measuring CAN bus message traffic (*canbus-load*). It should be noted that the txqueuelen parameter for the can interface must be set large enough to handle the amount of traffic. For instance, if using the default *cangen* setting of one message every 200 milliseconds, a *txqueuelen* of 1000 is sufficient for at least 251 instances of *cangen* as well as the CAN bus message traffic from the Arduino-based ECU emulators. However, if *cangen* is configured to run with a delay of 0 between generated messages, the TEEm system as currently configured was not able to handle more than 28 instances of the *cangen* process unless the *txqueuelen* parameter is configured to be at least 10000.

The Tangle Web Interface, through five different menu options, provides the mechanism for capturing snapshot metrics as well as some searching capabilities. These five menu options are:

- Dashboard: This tab allows for a quick view of: how many neighbors are connected, if the Hornet is synced, Milestone snapshots, Transactions Per Second (TPS), and server memory cache usage.
- Neighbors: This tab shows each neighbor, if that neighbor is connected, and general network metrics per connected neighbor.
- Tangle Explorer: The Tangle Explorer tab allows a user to search Bundle, Transaction, or Address Hash, as well as Milestone Index.
- Visualizer: The visualizer tab will create and display a visual depiction of the Hornet. Hornet databases are Directed Acyclical Graphs (DAGs) and this visualization is constantly changing to reflect the confirmed milestones, solid and unsolid transactions, and tips.
- Misc: This tab provides visual metrics for: Tip-Selection Performance, Request Queue, Server Metrics, Cache Size, Requests, and Database.

4.3 Lab Networks

The TEEm lab setup described in this paper and shown in Figure 2 makes use of two primary network configurations: the private-tangle Docker Bridge network and the virtual CAN bus network (configured as can0). External Ethernet access is used only for maintenance of the other devices.

As Figure 2 depicts, the virtual and physical Vehicle emulators are connected to the same CAN bus. As described in Section 4.1, this CAN bus consists of both physical and virtual segments. Terminating resisters are provided by the Arduinos. The Docker based vehicle emulators are additionally part of a private (RFC1918-based) network. This private network

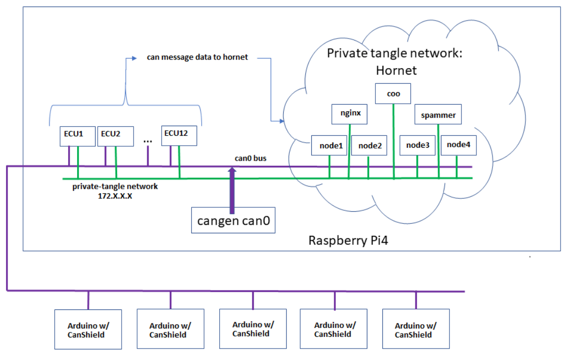

Figure 2: TEEm Network Setup.

is inherent to Docker and allows for the individual Docker Containers (individual virtual vehicle emulators) to communicate via Ethernet with the Tangle instance.

4.4 Setting up the IOTA Private Tangle

There are different ways in which an IOTA (IOTA, 2021b) Hornet network can be established. For the purposes of creating TEEm, we chose to utilize the private-tangle bridge created by IOTA's one-click setup script found at (iotaledger, 2021).

The private-tangle bridge is created by the one-click private-tangle script. This network will generally be a Class C network within the 172.0.0.0/16 block. The default private-tangle setup files have been modified to restrict the private-tangle bridge to networking only within the 172.20.0.0/24 subnet. Whenever a new instance of the Hornet network is started, specific IP Addresses (IPAs) for each container can change; typically, once an address has been assigned to a Docker Container, that container will have the same IPAs every time the Container is started.

The script creates a local Hornet network with a minimal number of components as Docker containers. These components are: a Coordinator, a Spammer, a NGINX, and a Node. For purposes of the TEEm setup, the Spammer and NGINX containers can be stopped and removed but the choice was made to not modify either of these. The primary changes made to the Hornet network from the script was the addition of more Nodes.

4.5 CAN bus Functionality on Raspberry Pi 4B

Configuring the CAN bus functionality on the Raspberry Pi 4B requires two steps: 1) ensuring that the standard Linux *can-utils* package is installed and functional and, 2) the installation and configuration of the USB2CAN hardware interface.

The *can-utils* package from the default Ubuntu repository was installed and then tested by establishing a CAN bus network interface and then generating and sniffing arbitrary CAN traffic. In addition to the *can-utils* package, some Kernel Modules are needed for the USB2CAN device to run properly. These Kernel Modules can be loaded by using the *modprobe* command with the can, can-raw, can-dev, can-bcm, and slcan modules.

Once the proper Kernel Modules are loaded, the chosen USB2CAN hardware interface (InnoMaker, 2021) used for the TEEm is supported by the *can-utils* package and was easily configured following the vendor instructions, using both the *slcand* and *ifconfig* commands. The *slcand* command will assign the USB device to a CAN interface and the *ifconfig* command is used to bring up the CAN interface and to set an appropriate *txqueuelen* value (we chose a large value of 50000). This device can be configured as the can0 interface on the OS and allows for the CAN bus traffic to be consumed by both the virtual ECUs and the Arduino-based ECU emulators.

4.6 ECU Docker Container Image

TEEm was originally based entirely within a single virtual machine (VM) and did not interface with any other physical components. In order to test functionality of the TEEm system using CAN bus traffic, virtual ECUs were created using Docker containers based upon the default Alpine (iotaledger, 2021) image from the Docker container repository. (Docker, 2021b)

The Alpine image is a minimally-sized Linux distribution. However, after the required software was installed for CAN bus network support as well as interaction with the Hornet network, the total image size was 878MB. As twelve virtual ECUs were to be created and used, the image was further customized to remove all unnecessary packages. This further customization resulted in a total size-on-disk of the image being approximately 146MB.

4.7 Run ECU Containers

Once the Docker Image has been created, there are two primary methods of running one or more ECU Containers based on this newly created image. The first method is to run an instance of an image. The other method is to create and use a docker-compose.yml file with the Docker Compose Engine. When using this method, the specific Docker Container instance must exist by name within the docker-compose.yml file.

4.8 Arduino-based ECU Emulators

The AEs can be created using any Arduino that supports a CAN bus Shield and from any version of the Arduino IDE as well and a working CAN bus library. In general terms, each AE emulator is configured to:

- Run a continuous loop checking for any received CAN messages
- Respond to CAN messages with specific Arbitration IDs (ArbIDs) by sending a CAN message to a different ArbID

Thus, it is the preference of the software author as to which libraries and/or packages are used to create these emulators. The current configuration has the Arduino-based ECU emulators in a listening loop-back until packets matching one of a specifically defined group of ArbIDs is received. Once any of the Arduino-based ECU emulators receives a specific ArbID, this will trigger the sending of a CAN bus message to another Arduino-based ECU emulator, which will then trigger the second Arduino-based ECU emulator to send to the third and, eventually, all five physical ECU emulators are sending and receiving to each other uninterrupted.

4.9 CAN bus Message Data in Hornet

The Hornet network can be interacted with through five (5) different programming languages. Python was chosen for its ease of use.

The PYOTA API library from (IOTA-Community, 2021) provides methods for querying the Hornet network by Address or Tag as well as getting Node API information, Neighbor information, and retrieving and decoding Tryte information.

The python code used to access a Hornet node's API interface and send a CAN bus message to the Hornet can be written in 20 lines of code, or fewer. TEEm uses a block of this python code that is executed within a while loop. This loop is sniffing the CAN bus and testing each message's Arb ID. If a message's Arb ID matches the ones assigned to the given vehicle, the CAN bus message string, to include the timestamp, is sent to the Hornet. This process is visualized in Algorithm 1.

The message string does require modification which involves appending the CAN bus message with "From Car" and the assigned vehicle number. Because a custom *address* and *tag* value is used per vehicle, both of these variables can be used to search the Hornet.

Algorithm 1: CAN message processing.

```
1: arbids =Assigned ArbIDs
2: while CAN message exists do
3:     if arbid is in arbids then
4:         CAN message sent to Hornet
5:     else
6:         CAN message is discarded
7:     end if
8: end while
```

As currently configured, when the TEEm startup script is executed, each of the twelve (12) Docker Container-based Vehicles is assigned a specific Hornet Node for executing Hornet API calls. Additionally, at TEEm startup, each Docker Container-based Car is randomly assigned four ArbIDs to represent a vehicle with four unique ECUs. When a Docker Container-based Vehicle receives a CAN bus message meant for one of its assigned ECU ArbIDs, the full message is sent to the Hornet instance, making use of the API access on a previously chosen Hornet Node. In addition to the actual CAN bus message, the stored string will also contain the ArbID that actually received the message along with the current number of messages that the ArbID has received since the starting of the TEEm iteration.

5 CYBERSECURITY AND PRIVACY APPLICATIONS

This section provides discussion on how the real-world use of a Tangle, or other Distributed Ledger Technology (DLT) can be used within connected vehicle environments to provide for cybersecurity and privacy protection applications.

As discussed in Section 2.1, the private Tangle instance, Hornet, provides key features that facilitate the cybersecurity and privacy goals: distributed copies of the database, individual transaction hashes, and the availability to query data from the Hornet database.

The distribution feature, in conjunction with the transaction hashing feature provides for protection against privacy and potential theft. If an adversary attempted to modify a transaction that has already occurred, the modified transaction will not be accepted as its hash already exists in the other database copies.

The transaction hashing feature provides for both immutability of stored data as well as a chaining of transactions that support forensic analysis. This type of analysis supports current research (Florea and Taralunga, 2020) for the detection of battery tampering. If an Electronic Vehicle (EV) is reporting health statistics to the Hornet, this data can be used in a few different types of applications:

- Trending of battery performance data
- Clustering algorithm support real time detection of battery data tampering or attacks on the battery directly

6 EVALUATION

6.1 CAN bus Traffic Creation and Evaluation

Every effort was made to design and configure TEEm to be as realistic as possible while attempting to establish upper bounds on storing messages in the Hornet instance. To accomplish this, each emulated ECU (Docker Container-based and Arduino-based) has a minimal delay of 100 milliseconds built-in via the processing loop code. This value was arbitrarily chosen as the starting point to represent message timing and queuing constraints injected by actions such as processor scheduling, dropped frames, and actual ECU processing time. This delay allows for additional future efforts focused on timing, blocking, queuing, and dropped metrics of CAN bus messages on the bus as well as the process of storing CAN bus messages in the Hornet.

There is no standard, fixed target regarding CAN bus loads. A given vehicle, on average, will have a CAN bus load between 40-60%. Errors, normally dropped frames, start to occur in CAN bus loads around 20%. Thus, it was decided that the maximum possible load should be applied to the TEEm CAN bus. The highest load percentage achieved, and used for all TEEm testing, is 96%. This was achieved by the CAN bus messaging produced by the Arduino-based ECU emulators as well as 228 instances of the *cangen* process. In order to achieve max load, and thus max errors and queue usage, the *cangen* process was executed using a delay of 0 seconds between messages. The actual data sent with each *cangen* message is of arbitrary value and length.

6.2 CAN bus Message Storage Process Evaluation

Because each ECU maintains a running count of the number of CAN bus messages that a given ArbID has received, and that count is stored along with its respective CAN bus message in the Hornet, it is trivial to determine if every message sent on the bus has been both received and processed by its assigned ECU/vehicle.

6.3 Hornet Performance Evaluation

The one-click private-tangle setup script for Hornet includes a web interface. This interface has five separate menu areas that are described in Section 4.A.2. Through these different options, the TEEm user can identify networking, cache, and database actions. The Hornet database is configured to auto-prune in order to keep the database to a minimal size. This auto-pruning also fully supports future planned efforts to emulate a vehicle entering and leaving a given Hornet ”zone” when that vehicle acts as a member node of the Hornet. Through the Misc tab, we can see that the Hornet database grows within the TEEm instance at an approximately constant rate of 32 MB/h until the maximum size is reached. It should be noted that the auto-pruning functionality is defined per Hornet Node and the database metrics found in the web interface are specific to the Node connected to by the web browser.

In addition to the database size metric, the web interface also allowed for our determination that the largest number of transactions processed by the connected Hornet Node in one second was 74 (52 Incoming and 22 Outgoing) and that this used approximately 43.1 MB of Heap memory with an additional 2 MB allocated on the Stack. The average amount of Heap memory used over a six (6) day iteration was 53.7 MB but this value will vary from machine to machine based upon resource availability and allocation.

6.4 Hornet Data Access Evaluation

Access to the Hornet system in TEEm is done through the PYOTA python library or through the linux built-in command *curl*. The PYOTA library allows for both a core and extended set of API calls. These API calls and their responses are JSON formatted and thus easily digested by common tools. However, the current TEEm configuration makes use of python scripts specifically created for TEEm. Most Hornet API calls through the PYOTA library will return a duration value in addition to the requested data. Throughout all TEEm experiments, the largest duration value seen was 2.3 seconds. This 2.3 seconds was atypical and was the result of a purposely crafted query that was intended to test the API call and does not have any real-world application. Standard queries using transaction ”tags” and the PYOTA library are shown in Table 1 and queries for transaction ”addresses” were performed using *curl* and are shown in Table 2. Ten different transaction ”tags” and ”addresses” were queried and for each queried tag and address, the number of returned records are shown in the ”Records” column. These queries were executed with the linux *time* command. The output of the *time* command is shown in the ”Real”, ”User”, and ”Sys” columns.

The results of the PYOTA based queries, summarized in Table 1, show that between 984 - 999 records were returned in times ranging between 472ms - 508ms. While these execution times are relatively short, we show that the queries executed via the *curl* command are significantly faster.

Table 1: Hornet Data Query Using Python.

Query	Records	Real	User	Sys
Tag-1	985	486ms	374ms	90ms
Tag-2	999	478ms	394ms	65ms
Tag-3	999	485ms	398ms	68ms
Tag-4	999	481ms	365ms	97ms
Tag-5	999	480ms	347ms	108ms
Tag-6	999	473ms	374ms	81ms
Tag-7	999	472ms	380ms	73ms
Tag-8	984	477ms	371ms	88ms
Tag-9	999	475ms	383ms	73ms
Tag-10	999	508ms	398ms	90ms

As previously mentioned, the queries of transaction ”addresses” using the *curl* command are below. The number of records returned range between 991 and 1001. The execution times are, as mentioned above, significantly smaller, ranging between 3 ms - 39 ms

Table 2: Hornet Data Query Using Curl.

Query	Records	Real	User	Sys
Address-1	1001	35ms	17ms	14ms
Address-2	1001	34ms	16ms	12ms
Address-3	1001	34ms	12ms	16ms
Address-4	1001	33ms	8ms	21ms
Address-5	1001	34ms	12ms	17ms
Address-6	1001	35ms	17ms	13ms
Address-7	1001	34ms	13ms	17ms
Address-8	993	39ms	5ms	26ms
Address-9	1001	35ms	24ms	5ms
Address-10	991	34ms	14ms	15ms

Shown in Table 3 below is the change in system memory usage over a given period. In this case, we measured memory usage at TEEm initialization and 24 hours later. These measurements were obtained using the linux *top* command. The system has 7113.3 Mb of RAM and as shown here, TEEm can become memory intensive. During this 24-hour period, the

Hornet database averaged in size of 142.0 Mb and the virtual CAN bus was running at an average load of 25% with 50 virtual docker vehicles having four assigned Arbitration IDs each.

Table 3: Hornet Memory Usage.

Measure	Free	Used	Buff
Mem at Start (MBs)	6170.9	732.1	908.3
Mem at Start+24hrs	98.2	3473.5	4239.6
Change	-6072.7	2741.4	3331.3

7 RELATED WORK

7.1 In-/Inter-Vehicle Attack Vectors and Countermeasures

As vehicles have become more connected and make use of multiple technologies, the number of attack vectors and the size of the attack surface has increased. Additionally, this attack surface consists of both In-Vehicle and Inter-Vehicle networks. Recent research has made efforts to categorize the types of attacks, the types of attackers, and potential detection and mitigation techniques. The authors of (Aliwa et al., 2021) present four general categories of attacks, with possible countermeasures, against in-vehicle vectors: direct interface-initiated attacks, infotainment-initiated attacks, telematics-initiated attacks, and sensor-initiated attacks.

7.2 Current Advancement Areas - In-Vehicle Security

Some of the recent In-Vehicle Cybersecurity areas of advancement include Message and ECU Authentication (ByteParadigm, 2021), Data Encryption, Intrusion Detection Systems and Gateways/Firewalls, and Cloud storage of encrypted data. These areas continue to be primary focal points of commercial and academic research. Advancements in Artificial Intelligence/Machine Learning have been investigated for IDS applications. Further areas of current advancement are those of encryption and hashing algorithms.

7.3 Current Advancement Areas - Inter-Vehicle Security

Current Inter-Vehicle cybersecurity research areas for advancement predominantly focus on vehicle localization and secure communications. Researchers in (Du et al., 2019) show a schema for using both Dedicated Short-range Communication (DSRC) and Extended Kalman Filtering for cooperative localization. Maintaining vehicle spacing using Adaptive Cruise Control (ACC), road grade, and motion preview from the lead vehicle is presented by researchers in (Firoozi et al., 2019). Researchers in (Alipour-Fanid et al., 2020) investigate the effects of jamming attacks on ACC. Both localization and distancing efforts are key components in efforts to detect Sybil attacks. Using a shared key for protecting intra-convoy communications is the focus of (Al-Shareeda, 2019).

7.4 New Challenges

Networking advancements, especially in the Cellular-Vehicle to Everything (C-V2X) domain, presents new challenges in protecting the communications that use these new advancements. The security of LTE-V2X is examined by researchers in (Zhu and Zhang, 2020). Connectionless communications using Bluetooth 5.0 technologies are examined in (García Ortiz et al., 2020). One of the problems of RF communications is that they do not always provide reliable and stable links and the authors of (Wang et al.,) show how unreliable links can still be used in vehicle platooning control. Recently, research into 6G technologies and implementations presents another area of focus regarding vehicle cybersecurity. Researchers in (Chen et al., 2020) examine the use of deep learning with 6G technologies as a means to provide safe distancing with autonomous vehicles. The challenges and research areas mentioned in this section are only a tiny fraction of the issues currently facing vehicle cybersecurity efforts.

7.5 DLT Technologies and Current Vehicle Data

The popularity of DLT technologies in recent years, especially Blockchain, has led to novel research efforts in the domain of using DLT technologies to store and/or process vehicle data. Researchers in (Dorri et al., 2017) focus directly on the use of Blockchain with Connected Smart Vehicles (CSV) to store and protect data related to security and privacy. The authors of (Falco and Siegel, 2020) present a framework that uses vehicle-based DHT Nodes combined with an external Blockchain storage structure. The primary focus of this work is the secure updating of vehicle/ECU software. The survey found in (Rahtore et al., 2020) discusses the use of Blockchain within Cyber-Physical Systems (CPS) to include different vehicle and transportation systems as well as Smart

Cities. Another study of Blockchain used as a data storage mechanism for vehicles and Smart Cities is presented in (Javed et al., 2020).

There has been some research into the usage of DLT and Blockchain with regard to vehicle cybersecurity. The authors of (Dorri et al., 2017) present a potential solution for using Blockchain to store both automotive security and user privacy data. Focused on a Smart Public Safety system, the authors of (Xu et al., 2019) present BlendMAS, a Blockchain architecture based on Service Oriented Architecture (SOA) and Internet of Things communications using the HTTP protocol. Integrating a DHT with a Peer to Peer (P2P) architecture for the purposes of storing IoT data is presented in (Tracey and Sreenan, 2019). Smart Contracts, a form of Blockchain, and their usage within Smart Cities is discussed in (Lindsay, 2018) and (Montes et al., 2019).

8 HORNET USE-CASE

In this section we present a real-life, high-level use-case for storing and using CAN bus message data in a Hornet instance. The use-case presented here is based upon the following assumptions:

Assumption 1: Each Roadside Unit (RSU) has a service range based upon the signal strength measured from each vehicle. Determining the effective range is out of scope for this effort.

Assumption 2: Each RSU implements an authentication system when a vehicle enters or leaves the RSU's service range. The ability of RSUs to implement authentication and message signing can be found in (Ashritha and Sridhar, 2015), (Hsin-Te et al., 2010), (Aman et al., 2021), and (Bendouma and Bensaber, 2017).

Assumption 3: The RSU has an active Hornet Node at all times and will use its own Hornet Nodes' API for performing Hornet functions.

Assumption 4: The RSU will prune from the Hornet the data belonging to a vehicle's VIN when said vehicle is deauthenticated from that RSU.

Assumption 5: The RSU makes use of a dynamically configurable firewall in order to control access to the Hornet from authenticated vehicles.

Assumption 6: A vehicle that has been inactive for a set period (we arbitrarily chose five minutes), with regards to Hornet operations, will be deauthenticated from the RSU and will, at first available instance, remove all data previously retrieved from the RSU's Hornet. If a vehicle is still within the RSU's service range, the authentication to the RSU will be preformed again. This accounts for use cases of:

- Vehicles that may have been parked and shutdown for a length of time
- Failures or obstacles to line-of-sight (LOS) communication links
- Vehicle is unable to make of LOS links and is not capable of establishing non-line-of-site (NLOS) communication lengths.

This inactivity period also removes the dependency on the vehicle to notify the RSU that the vehicle is stopping or ending a route. After deauthentication, the actions performed during the "Leaving the RSU Area" phase are performed.

Assumption 7: Data in the Hornet has two levels of access: 1) publicly available data that all RSU-authorized vehicles, as well as all Hornet nodes, can query, and 2) encrypted data that access is denied to all vehicles and Hornet nodes unless specially permitted.

8.1 Entering RSU Area

Entering the RSU area involves the following steps:

1) RSU-controlled authentication of vehicle

2) RSU dynamically adjusts its firewall rules

3) RSU submits vehicle VIN to Hornet

4) Vehicle queries Hornet

5) Vehicle is able to fully interact with Hornet

We start by assuming that a vehicle will be subjected to an (1) RSU-controlled authentication process when it is within an effective range of communications with an RSU. As part of the the authentication processes, when the RSU is satisfied that the vehicle is who it claims to be, (2) the RSU will dynamically adjust its firewall rules to allow the vehicle's IPAs to connect to, and interface with, the API from any of the RSU's Tangle Nodes. Once the RSU has allowed a vehicle to connect to the Hornet, (3) the RSU will create and submit a Hornet transaction that contains the VIN of the just-authorized vehicle. (4) The vehicle will query the Hornet for its own VIN and, upon a (5) successful query execution, will be able to communicate with the Hornet and will be able to actively query all publicly available data within the Hornet.

8.2 Storing CAN bus Message Data in Hornet

Once the vehicle is authenticated and is a node in the RSU's Tangle, the vehicle is able to store data in the Hornet, in the form of a null-value transaction. Using its Vehicle Identification Number (VIN) as the address for all of its transactions, the vehicle is able to send data-agnostic transactions to the Hornet. For our purposes here, the vehicle will submit a new transaction at 500 millisecond intervals to the Hornet. These transactions will contain the vehicle's VIN and velocity. Thus, the Hornet will have accurate an accurate location of the vehicle that is no older than 500ms.

8.3 Using CAN bus Message Data from Hornet

RSU-authenticated vehicles are able to query all publicly available data that is stored in the Hornet network. As such, the data stored by each vehicle is potentially usable by every other vehicle node in the Hornet network. This availability of data, along with the ability for each vehicle node to query Hornet provides a mechanism to support safety, privacy, and cybersecurity concerns.

Because the RSU stores all currently authenticated VINs within the Hornet, using the RSUs Hornet address and a transaction tag of ”VINs,” all vehicles joined to the Hornet are able to query by address and tag to identify all of the vehicles currently within range of the RSU. Once a vehicle has a list of the other connected VINs, a query of the Hornet, using a specific VIN, is executed to obtain all globally available information that the target VIN has stored in the Hornet. This function can be executed by the vehicle as a batch job and, obtaining location and velocity data of every RSU/Hornet connected VIN, the vehicle is able to determine if there are any imminent threats of collision with other vehicles.

8.4 Leaving RSU Area

When a vehicle leaves a specific RSU's area, the RSU will deauthenticate the vehicle from both its Hornet as well as itself. The vehicle, after no longer having authenticated communications with the Hornet and RSU will purge all Hornet data that the vehicle was storing and using.

When the RSU deauthenticates the vehicle, it will first dynamically reconfigure its firewall to block all non-authentication based traffic between the Hornet nodes and the vehicle. The RSU will then purge the Hornet database of the vehicle's data.

9 CONCLUSION

This paper presented the creation of TEEm, a hybrid virtual and physical environment, that is used for the testing of storing CAN bus message data in a DLT (i.e. Hornet). It presented details of the technologies used, described the design and setup, and presented the evaluation of TEEm. In addition, other works related to this effort were discussed, and a basic usecase for autonomous vehicles storing and accessing data from the Tangle network was presented. The versatile nature of TEEm allows for quick configuration of virtual vehicles and realistic injection of latency and noise in physical connections. In future work, TEEm should contribute to ECU behavior modeling, prototyping of CAN bus firewall rules, and prototyping of CAN bus IDS technology.

REFERENCES

Al-Shareeda, S. (2019). Veiled shared key generation for intra-convoy secret communication sessions. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7.

Alipour-Fanid, A., Dabaghchian, M., and Zeng, K. (2020). Impact of jamming attacks on vehicular cooperative adaptive cruise control systems. volume 69, pages 12679–12693.

Aliwa, E., Rana, O., Perera, C., and Burnap, P. (2021). Cyberattacks and countermeasures for in-vehicle networks. 51(1):1–37.

Aman, M. N., Javaid, U., and Sikdar, B. (2021). A privacy-preserving and scalable authentication protocol for the internet of vehicles. *IEEE Internet of Things Journal*, 8(2):1123–1139.

Ashritha, M. and Sridhar, C. S. (2015). Rsu based efficient vehicle authentication mechanism for vanets. In *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–5.

Banihani, A., Alzahrani, A., Alharthi, R., H., F., and Corser, G. P. (2018). T-paad: Trajectory privacy attack on autonomous driving. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–2.

Bendouma, A. and Bensaber, B. A. (2017). Rsu authentication by aggregation in vanet using an interaction zone. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6.

Bozdal, M., Samie, M., and Jennions, I. (2018). A survey on can bus protocol: Attacks, challenges, and potential solutions. In *2018 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, pages 201–205.

ByteParadigm (2021). Introduction to i2c and spi protocols.

Chen, X., Leng, S., He, J., and Zhou, L. (2020). Deep learning based intelligent inter-vehicle distance control for 6g-enabled cooperative autonomous driving.

Chua, H. N., Teh, J. S., and Herbland, A. (2021). Identifying the effect of data breach publicity on information security awareness using hierarchical regression. *IEEE Access*, 9:121759–121770.

Docker (2021a). Docker: Docker.

Docker (2021b). Docker repository.

Dorri, A., Steger, M., Kanhere, S., and Jurdak, R. (2017). Blockchain: A distributed solution to automotive security and privacy. volume 55, pages 119–125.

Du, L., Chen, L., Hou, X., and Chen, Y. (2019). Cooperative vehicle localization base on extended kalman filter in intelligent transportation system. In *2019 28th Wireless and Optical Communications Conference (WOCC)*, pages 1–5.

Falco, G. and Siegel, J. E. (2020). Assuring automotive data and software integrity employing distributed hash tables and blockchain.

Firoozi, R., Nazari, S., Guanetti, J., O'Gorman, R., and Borrelli, F. (2019). Safe adaptive cruise control with road grade preview and **V2V** communication. In *2019 American Control Conference (ACC)*, pages 4448–4453.

Florea, B. C. and Taralunga, D. D. (2020). Blockchain iot for smart electric vehicles battery management. 12(10):3984.

García Ortiz, J. C., Silvestre-Blanes, J., Sempere-Paya, V., and Tortajada, R. P. (2020). Feasability of bluetooth 5.0 connectionless communications for i2v applications. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1119–1122.

Hsin-Te, W., Li, W.-S., Tung-Shih, S., and Hsiehz, W.-S. (2010). A novel rsu-based message authentication scheme for vanet. In *2010 Fifth International Conference on Systems and Networks Communications*, pages 111–116.

Hubaux, J. P., Capkun, S., and Luo, J. (2004). The security and privacy of smart vehicles. volume 2, pages 49–55.

Huybrechts, T., Vanommeslaeghe, Y., Blontrock, D., Barel, G. V., and Hellinckx, P. (2017). Automatic reverse engineering of CAN bus data using machine learning techniques. pages 751–761. Springer International Publishing.

InnoMaker (2021). Usb2can.

IOTA (2021a). Hornet: Hornet.

IOTA (2021b). Iota homepage.

IOTA-Community (2021). Python iota workshop.

iotaledger (2021). One-click tangle.

Javed, M. U., Rehman, M., Javaid, N., Aldegheishem, A., Alrajeh, N., and Tahir, M. (2020). Blockchain-based secure data storage for distributed vehicular networks. 10(6):2011.

Lindsay, J. (2018). Smart contracts for incentivizing sensor based mobile smart city applications. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–4.

M., C. and Lee, J. (2011). Ppas: A privacy preservation authentication scheme for vehicle-to-infrastructure communication networks. In *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1509–1512.

Mazloom, S., Rezaeirad, M., Hunter, A., and McCoy, D. (2016). A security analysis of an in-vehicle infotainment and app platform. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*.

Mekliche, K. and Moussaoui, S. (2013). Location-based privacy-preserving detection of sybil attacks. In *2013 11th International Symposium on Programming and Systems (ISPS)*, pages 187–192.

Miller, C. and Valasek, C. (2013). Adventures in automotive networks and control units. 21(260-264):15–31.

Miller, C. and Valasek, C. (2014). A survey of remote automotive attack surfaces. 2014:94.

MIller, C. and Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle.

Montes, J. M., Ramirez, C. E., Gutierrez, M. C., and Larios, V. M. (2019). Smart contracts for supply chain applicable to smart cities daily operations. In *2019 IEEE International Smart Cities Conference (ISC2)*, pages 565–570.

Rahtore, H., Mohamed, A., and Guizani, M. (2020). A survery of blockchain enabled cyber-physical systems. volume 20.

Security, U. (2020). Upstream security's 2020 global automotive cybersecurity report.

T. W. Chim, S. M. Yiu, L. C. K. H. and Li, V. O. K. (2009). Security and privacy issues for inter-vehicle communications in vanets. In *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, pages 1–3.

Tracey, D. and Sreenan, C. (2019). Using a dht in a peer to peer architecture for the internet of things. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 560–565.

Tyagi, A. K. and Sreenath, N. (2015). Location privacy preserving techniques for location based services over road networks. In *2015 International Conference on Communications and Signal Processing (ICCSP)*, pages 1319–1326.

Wang, J., Ma, F., Yang, Y., Nie, J., ksun Guvenc, B., and Guvenc, L. Adaptive event-triggered platoon control under unreliable communication links.

Wen, Q., Chen, A., and Lin, Z. (2020). Plug-n-pwnded: Comprehensive vulnerability analysis of obd-ii dongles as a new over-the-air attack surface in automotive iot. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 949–965.

Xu, R., Nikouei, S. Y., Chen, Y., Blasch, E., and Aved, A. (2019). Blendmas: A blockchain-enabled decentralized microservices architecture for smart public safety. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 564–571.

Zhu, K. and Zhang, L. (2020). Security analysis of LTE-V2X and a platooning case study. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 532–537.