TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technology

Cyber Security Engineering

Ahmed Ruhul Quddos Joyon 184059IVSB

Intrusion Detection in Self-Driving Cars Using Honeypots

Bachelor Thesis

Supervisor

Kaido Kikkas

PhD

Hayretdin Bahsi

PhD

Tallinn 2022

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author:	Ahmed Ruhul Quddos Joyon	....
		(signature)
Date:	January 06, 2022	

Annotatsioon

Viimasel ajal on autonoomsete või isejuhtivate autode valdkonnas tehtud palju arendusi. Uuringud näitavad, et seda tüüpi nutikad autod on haavatavad erinevat tüüpi turvaohtude ja nullpäeva-rünnakute suhtes. Erinevate "asjade Interneti" ja isejuhtivate autode uurimuste põhjal selgus, et enamik neist on kas Telneti või SSH rünnakud, mille abil ründaja üritab süsteemi üle kontrolli saavutada. Selle lõputöö eesmärk on näidata, et seda tüüpi sõidukite jaoks on võimalik välja töötada kõrge interaktsioonitasemega meepott. Selleks sõnastatakse töös nõuded meepoti väljatöötamiseks, mis simuleerib isesõitvat autot. Pärast seda luuakse Duckieboti abil meepoti prototüüp, mida turvatestitakse sõltumatu kasutaja poolt.

Lõputöö on inglise keeles ja sisaldab 33 lehekülge teksti, 5 peatükki ja 27 joonist.

Abstract

In recent times a lot of development has been done in the field of autonomous or self-driving cars. Research shows that this type of car is vulnerable to different types of security threats and zero-day attacks. From different research honeypots for IoT and self-driving cars showed that most of this are either telnet or SSH attacks by which attacker tries to take control of the system. This thesis aims to show that it is possible to develop a high interaction production honeypot as a deception technique for self-driving cars. For this author will set up requirements to develop a honeypot that will simulate as self driving car. After that author designed a prototype honeypot using Duckiebot.This honeypot was then evaluated by common pen testing technique. Feedback from this evaluation and some recommendation had been provided.Future research direction on this topic was discussed.

The thesis is in English and contains 33 pages of text, 5 chapters and 27 figures.

List of abbreviations and terms

HIH	High Interaction Honeypot
IOT	Internet Of Things
MQTT	MQ Telemetry Transport
ROS	Robot Operating System
SDC	Self Driving Car
UAV	Unmanned Aerial Vehicle
TLS	Transport Layer Security

Table of Contents

List of Figures

1. Introduction

Vehicles are one of the important aspects of our society. We use these to carry people and goods from one place to another. With digitalization, old mechanical cars became faster, reliable and user friendly. Although it has a lot of necessity, it is also can create hazards to our society in terms of traffic accidents. Usually, these accidents are caused by human errors or faulty mechanical parts in the car. For that,the development of self-driving cars has been the main focus in this decade. A lot of big name companies such as Google, Tesla, GM are continuously working to improve their cars to hand over more control to machines . This autonomy of car is becoming a reality due to the advancement of positioning and sensors technology. All self-driving cars use different sensors that feed data to a central processing unit that by using artificial intelligence move the car move safely on the road. These cars also has a remote monitoring system that can control or monitor its operation from a remote location.

Also, most of these modern cars are connected to the internet to receive and send different information.These cars are not only getting connected to the internet they also communicate with other vehicles to gather different information such as traffic lights, pedestrian movement,etc. Globally 30 million connected cars are sold in 2020[1]. This technology also solves the problem of long-haul trucking as Tesla demonstrated in their autonomous truck. Before we make this vehicle used by the mass public all different communities has to make sure that these types of vehicles are safe to use. One of them is the security research community, who will have to identify possible security threats in self-driving cars and propose possible remediation to these threats.

1.1 Problem Statement

Self-driving cars rely on different sensors, software and hardware to run their operation. As these new additions of different components made driving easier , they also increase the attack surface and threat scenario for the car. A vulnerable self-driving car is not only harmful to itself but can also be used as a launchpad for attacking other self-driving car connected to it using vehicle to vehicle communication. Because of this different architecture of self-driving and its communication protocol inside and outside the network, it is always a possibility that an intruder can create a backdoor to the main network of the car and take control of its component to create harm full and dangerous circumstances. Rather than being on the defense because of these attacks it is smarter to act in offense and detect the intruder. One of the main offensive security measure is intrusion detection. Detecting a attack early without compromising the car can save not only our property but also a lot of lives. Chris Sanders, in his book "Intrusion Detection Honeypot", describes a detection process through deception[2]. Using deception techniques we can detect, slow down, respond, analyze and alert intruder attacks. Honeypot is one of the deception techniques that's being used as a part of deception mechanism. All though there are lot research honeypots are created to understand different attack scenario on IoT devices and autonomous vehicles, there is currently no approach to actually develop a production honeypot for self-driving car as an intrusion detection system. For these reasons following research questions are formulated:

1. What kind of design requirements are needed to develop a honeypot for a self-driving car?
2. Will this honeypot can be used as detection mechanism for a self-driving car?

1.2 Objective

The main objectives of this thesis are given below:

1. Develop requirements for high interaction honeypot for self-driving cars.
2. Design a prototype honeypot based on those requirement.
3. Evaluate and test the honeypot as a detection tool for self-driving cars.

1.3 Methodology

The author will design the thesis based on design science research framework[3]. Following diagram shows us processes author will follow in this thesis:

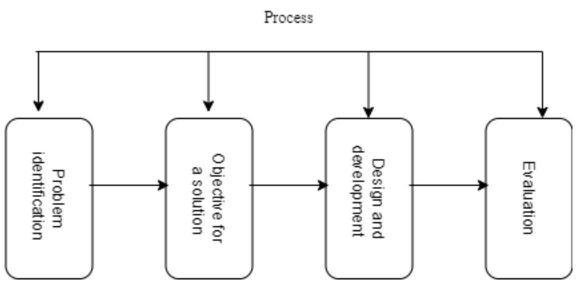


Figure 1. DSR methodology[3]

To create requirements and design an actual honeypot author will keep four things in mind:

1. Deceptive
2. Discoverable
3. Interactive
4. Monitored

In chapter two author discusses how to achieve these steps based on frame work provided by Attivo networks. To evaluate and test the proposed honeypot author will use the framework provided by MITRE ATT&CK.The following steps will be used to check validation of honeypot.

1. Initial Access
2. Execution
3. Persistence
4. Privilege Escalation
5. Defense Evasion
6. Credential Access
7. Discovery

All this section has different subsections or techniques. Author will discuss which technique will be used in detail in Chapter 2.

1.4 Thesis Outline

This thesis contains the following chapters.

1. Background: Provides background information for this study and discusses works already conducted for security in self-driving cars, different honeypots that are developed as part of security measures in self-driving cars and IoT devices.
2. Proposed Solution: This chapter analysis different component required for a successful high interaction honeypot, propose a feasible design.
3. Evaluation: This chapter discusses the evaluation experiment conducted , its result and feedback.
4. Summary: This chapter summarises our thesis, its outcome and future research direction.

2. Background

In this chapter author will discuss self-driving car and there network architecture, honeypot and, different attacks that can happen. The author will also discusses related works done in this field.

2.1 Self-Driving Car

A self-driving car is a vehicle that by using sensors to understand the surrounding environment operates without human involvement. At any point in time, this car does not require a human driver or passenger. It acts like any traditional car and can go to any road a normal car can go. Currently, there is six levels of autonomy in self-driving cars .

1. Level 0: No automation. All driving related works are done by the driver.
2. Level 1: Driver Assistance. The driver completes all driving related tasks with help of some safety features.
3. Level 2: Partial Automation. The driver remains alert and monitors the environment. One or two tasks are done by the driver simultaneously.
4. Level 3: Conditional Automation. All environments are monitored by car but the driver must be ready to take over at any point in time.
5. Level 4: High automation. The car performs all tasks but the driver needs to present in the car.
6. Level 5: Full automation. Driver not required. The car performs all tasks.

2.1.1 Working Principle and Components

To build and understand how SDC car works we have to understand five components[4].

1. Vision
2. Sensor
3. Location
4. Route
5. Control

Vision is how SDC sees its surrounding. It is done by placing cameras around the car. These images are then processed by a different machine learning algorithm to identify objects.

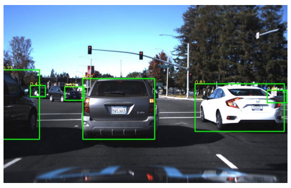


Figure 2. Computer Vision on SDC[5]

One of the main parts of driving on the road is to always be aware of the surrounding environment. SDC comes with a lot of sensors that feed data and using technologies like LiDAR creates a 3D representation of the surrounding environment.

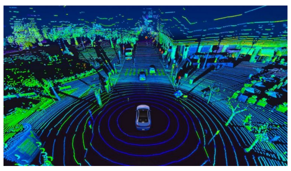


Figure 3. 3D modeling using LiDAR[6]

Next component is locator . This component is used to locate where the car is in the real world so that Route planning can be done. Using locator components such as GPS and some complex algorithm self-driving cars decides the most optimal path to go from point A to point B. After getting all necessary information using AI cars to travel from point A to B. Author showed a figure of hardware component self-driving car in appendix 1. This figure shows all the components described above. It also includes a network switch that connects all the component together and a logging system for monitoring purposes.

2.1.2 Software

Autoware

Autoware is an open source all in one software for self-driving vehicle. It has a rich set of modules for self-driving car such as:

- Sensing
- Computing
- Actuation capability

Details regarding these modules are described in below picture:

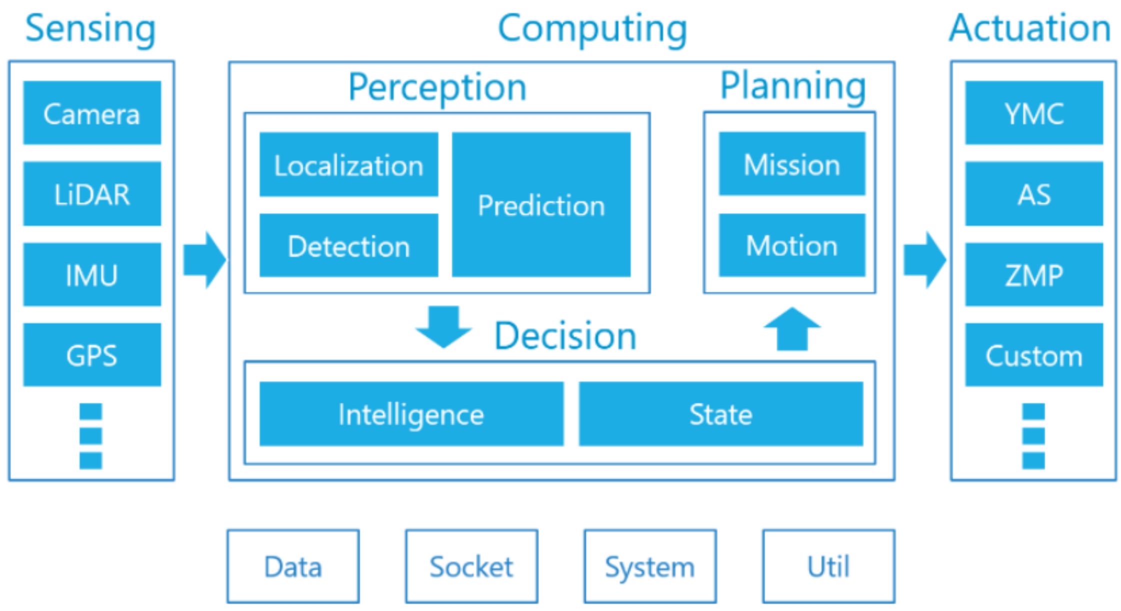


Figure 4. Different modules of self-driving car[4]

Robot Operating System

ROS is a unix based open-source light operating system to control robotic functions. It works like a regular operating system and also provide frameworks tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS file-system level contains following packages:

1. Packages: These are the main units of organizing software in ROS. It contains nodes, library , data sets, configuration files.
2. Metapackages: these are special packages that represent a group of related packages.
3. Package manifests: It contains metadata regarding different packages.

ROS uses different nodes to process data together. These nodes communicate with each other by passing messages. These messages are routed via topics using publish/subscribe model[7].

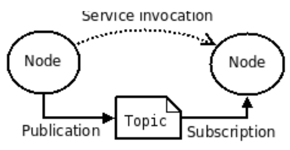


Figure 5. Communication between two nodes[7]

2.1.3 Security Vulnerability

As per the above discussion, it is seen that SDC uses a lot of IoT components. This components interact with each other and other cars over the internet. A lot of these components are vulnerable to attacks that are very easy to execute even for a novice attacker. Using some known method already out in the open one can remotely take control of a vehicle just using very simple equipment. Rather than that it is very clear based on some research that these cars are also vulnerable to zero-day attacks. The reason behind these security issues are mainly manufacturer does not design there product having security threats in mind. To prevent this kind of threats our paper mainly focuses on intrusion attacks.

Intrusion attacks

Intrusion attacks refer to illegal activity in a digital environment. This kind of attack usually overwhelms network infrastructure, steals confidential data and use it as a launchpad for cyber attack in connected networks. There are different techniques of intrusion attacks. Such as:

1. Multi-routing
2. Buffer Overwriting
3. Covert scripts
4. Brute-force attack
5. Malware
6. Worms

MQTT attack

One of the important security vulnerability present in current self-driving cars are in MQTT communication.there are several security breaches that can happen while using this protocol[8].

1. Brute Force Authentication
2. Denial Of Service Attack
3. Distributed Denial Of Service Attack

In DoS attack, an attacker tries to deliver malicious packets using MQTT protocol that results in interruption in receiving actual massages from legitimate broker. In this type of attack , it is possible to exhaust resources and provide false information. In DDoS service attack it is also possible to send these packets from a distributed source which means it will be much more harder to mitigate.

2.1.4 MITRE ATT&CK

The MITRE ATT&CK[9] is a global knowledge base consisting of tactics and techniques used by hackers. This knowledge base is created based on real-life observation. Anybody can use this matrix to develop threat models and methodology. Currently, this matrix is used in the private sector, government and security product development community. ATT&CK Matrix:

1. Reconnaissance: Active Scanning, Gather victim host information, identity information etc.
2. Resource development: Acquire infrastructure, Compromised accounts, develop capability, etc.
3. Initial Access: External remote service, valid accounts, exploit public facing applications etc.
4. Execution: User execution, system execution etc.
5. Persistence: Create accounts, create and modify system processes, etc.
6. Privilege Escalation: Abuse Elevation Control mechanism, valid accounts, scheduled tasks and jobs, etc.
7. Defense Evasion: Deploy container, hide artifact, etc.
8. Credential access: Brute force, password stores, etc.
9. Discovery: File and directory services, remote system discovery, etc.
10. Lateral movement: Exploitation of remote services, session hijacking, etc.
11. Collection: Archive data collection, data from local file system, etc.
12. Command and Control: Data encoding, protocol tunnelling, etc.
13. Exfiltration: Automated exfiltration, scheduled transfer, etc.
14. Impact: Data destruction, data manipulation, etc.

2.2 Cyber Deception

Cyber deception is a technique to deceive attacker into a false and pre-planned trap inside a network to control, monitor his every movement and then take action against them. Cyber deception creates uncertainty in attackers mind, wastes his time and advisory gets to understand attackers kill chain and prevent them in real life. Advantages of cyber deception is that it puts advisory in driving sit as they know exactly what attackers are doing inside there network. Using cyber deception advisories can prevent cyber kill chain by planning a deception strategy for every stage of kill chain.Figure

6 shows how cyber deception technique can be used to counteract kill chain[10]. Although cyber deception sounds like honeypot but actually it is just one important

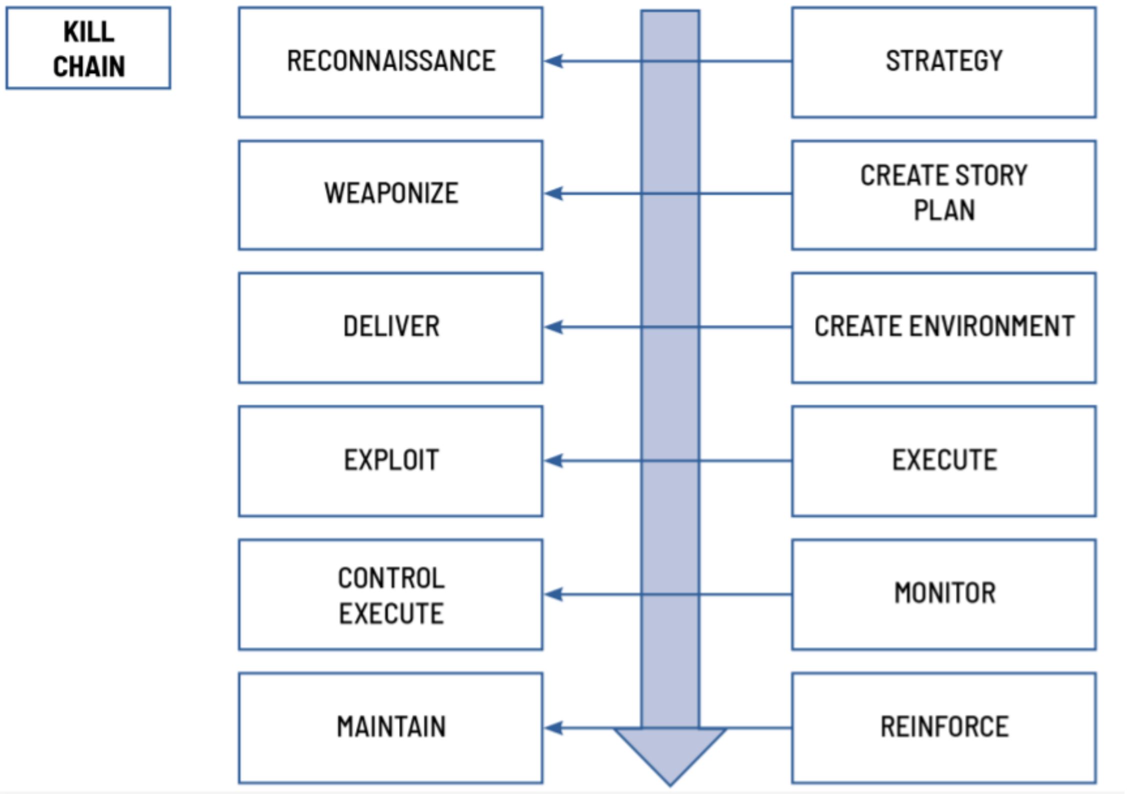


Figure 6. Kill Chain vs Cyber deception[10]

part of deception technique.

Honeypot

A honeypot is a trap to lure an attacker into seemingly working network infrastructure, which is created to attract and study attacker's behavior. Honeypot is the first step to have an operational intrusion detection and prevention system. It is always created very realistically and also contains tools to monitor and analyze intruder's behavior[11].

Characteristics

All honeypots usually has four characteristics.

1. Deceptive: Honeypots present some form of deception by representing false truth. They might appear as a real system or service but it does not represent a real system and does not have a business value. There are two ways one can achieve deception in our honeypot. Either by hiding it in the background of real services or by showing which means that it is showing something especially to lure them in.
2. Discoverable: Honeypots has to be discoverable to accessible within the proper

context. The placement of honeypot defines its purposes.
3. Interactive: As honeypot are not what it appears to be but it should be as interactive as possible as a real network.
4. Monitored: To connect all of this together honeypot needs to be properly monitored.

Based on levels of interaction required in a honeypot, it can be classified into three groups:

- Low-Interaction Honeypots (LIHP)
- Medium-Interaction Honeypots (MIHP)
- High-Interaction Honeypots (HIHP)

Low-Interaction Honeypots

This kind of honeypot emulates a limited range of services for the attacker to use. This kind of honeypot does not require many resources and can be maintained very easily. It's implementation only requires set up of a hypervisor which will emulate an operating system and also set up a monitoring system while the actual system remains untouched. Using this kind of honeypot one can easily detect paths and origin of an attack but it can not be used to understand the method of attacker. The above figure shows the attack path

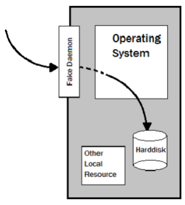
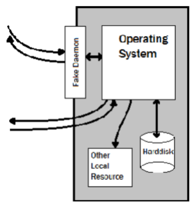


Figure 7. Attack flow in LIH[12]

available to a attacker when targeting a LIH.

High-Interaction Honeypot

Opposite of LIH , HIH offers a full range of systems for the attacker to interact with. It does not emulate any services but has real system and services an organization might have. This way attacker can compromise the whole system and can take control. Bu using this kind of honeypot we can have a better understanding of attack methods and tactics. Although HIH can be very useful it come with some disadvantages also. As it is necessary to connect this honeypot to a real network, there is a chance that the attacker might get access to real network via honeypot. It is also very complicated to configure in the long run. Figure 8a shows the the attack path available for attacker. Here the attacker has access to full range of operating system and its resources. Also, OS has access to near by machines in the network. VMware software can work as a HIH by creating different virtual hosts



(a) Attack flow in HIH[12]

connected to each other. Even an unpatched windows personal computer can work as a HIH.

Medium-Interaction Honeypots

In MIH there is also no operating system present but it is more interactive than LIH. It also emulates OS, resources and services and can interact with the an attacker and can answer commands from attacker. Emualted services are more complex than what LIH have but it is also very safe as seen in LIH. This kind of honeypot is used as a specific and controllable option to detect specific kind of attacks. MIH has the best from both side of the honeypot and can be very scalable and low maintenance.

Based on what purpose it serves honeypot can be separated into two categories.

1. Production Honeypot: This type of honeypot are used to collect data security related information from an actual production network.
2. Research Honeypot: A research honeypot are used to understand the attack methods and tactics of an attacker. This type of honeypot is used by government or security researchers.

Honey token

Security is always more about data than computer resources. The three pillar of cybersecurity is availability, integrity and confidentiality of data. One of the way we can simulate data is by using honeytokens. Honeytokens are security resources that simulates actual data. There are multiple ways we can use honeytokens.

1. Honeydocs: This is an office document that contains a honeytoken named webbug. If an attacker opens this file a external URL will be referenced. By monitoring that web server it will be known who and when these documents were opened.
2. Honey Files: Honeyfiles can be any file that is paired with logging to detect intruder.

2.3 Log Management

Log Management is a service to gather, store, process and analyze data coming from one or more applications. Log is a computer generated file that captures all or specific activities within an operating system , applications, or in our prospective honeypot. These files automatically generates documents specified by the administrator.[13] Log management usually has six categories[13].

1. Collection
2. Monitoring
3. Analysis
4. Retention
5. Indexing
6. Reporting

2.3.1 Rsyslog

Rsyslog is an open source and fast system for log processing[14]. Advantages:

1. High performance
2. Security features and modular design
3. Can accept information from an wide range of services process them and send output to diverse destinations.

2.3.2 ELK Stack

ELK Stack is all in one log management tool that contains E for elastic search, L for Logstash and K for Kibana[15].

1. ElasticSearch ; Stores log
2. LogStash: ships, process and stores log
3. Kibana: Visualization tool hosted in Nginx or Apache.

Simple architecture of ELK stack :

2.4 Previous work

This section will look into previous works in four parts.

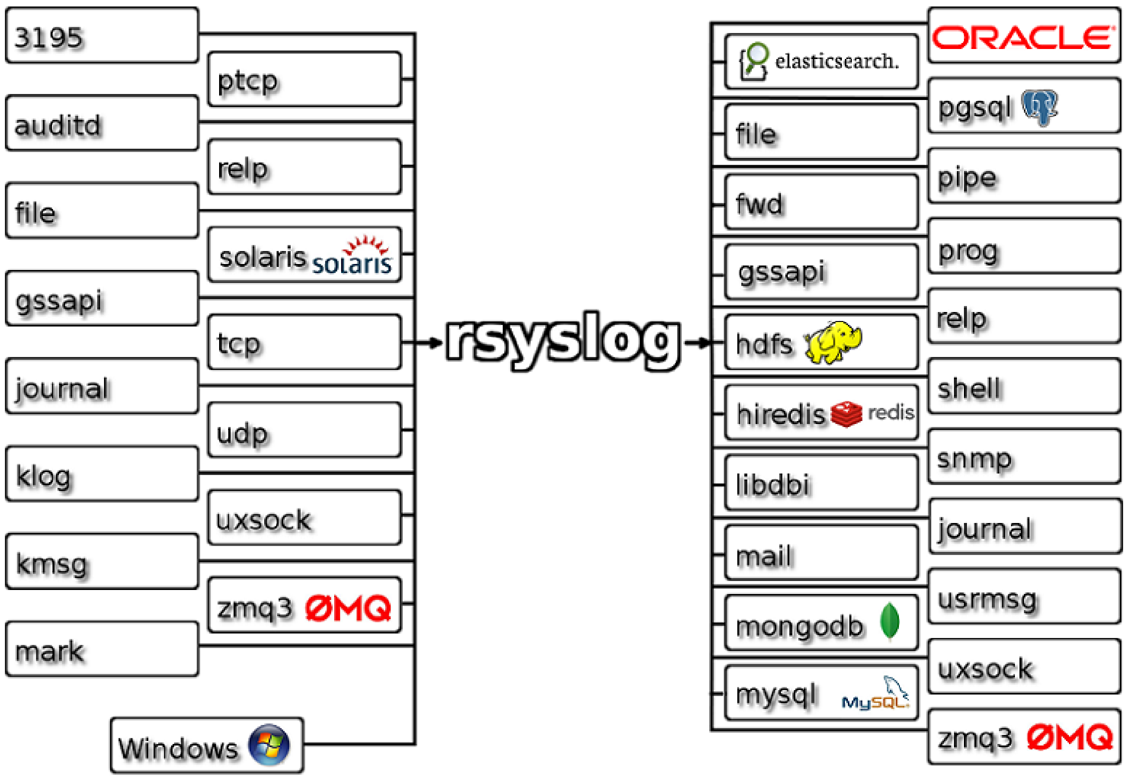


Figure 9. Rsyslog features[14]

1. Attack surface analysis
2. Current security process and practices
3. Cyber deception
4. Research honeypot for IoT devices
5. Production honeypot for UAV and IoT devices

There is currently a lot of works done to identify threats in self-driving cars . But the problem with these researches is that it did not take into account the wide range of component present in self-driving cars. To understand the attack surface of a self-driving car it is needed to take into account three major components. Communication, sensors and data [16]. S.Behere and M.Torngren proposed a functional component of autonomous vehicle in to three categories[17].

1. Perception
2. Decision
3. Vehicle platform manipulation.

The following diagram from[17] shows us a reference architecture for self-driving car:

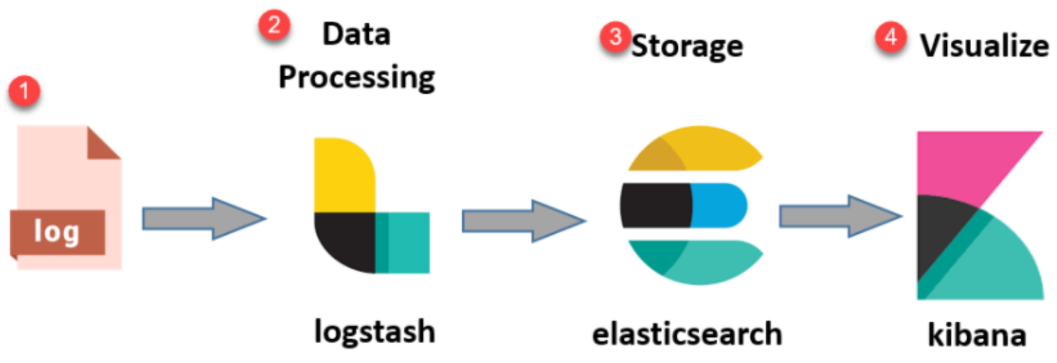


Figure 10. ELK architecture[15]

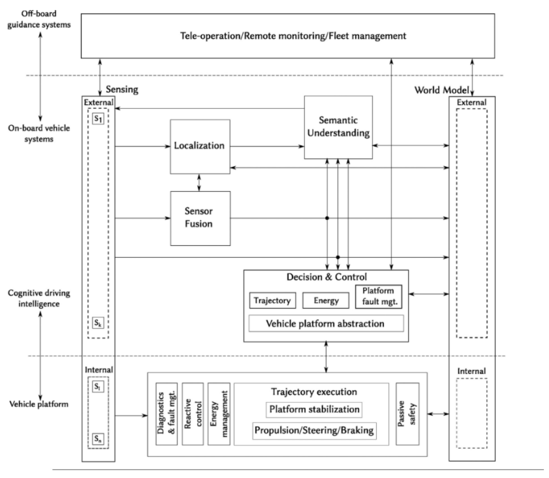


Figure 11. Functional Architecture for SDC[17]

Self-driving vehicles consist of a lot of hardware and software. These include but are not limited to operating systems, computational hardware, middleware and a codebase. Choi in his paper investigated different vulnerabilities that are present in ROS middleware.The main vulnerabilitiy discovered in ROS is a lack of authentication. This kind of platform performs lot of tasks simultaneously to achieve a single task. To manage these processes ROS works as an central management unit. This paper showed a variety of exploits that exist in ROS[18].

1. ROS master spoofing
2. Intercepting and replaying ROS log files
3. Insertion of malicious processes

Another security issue is ransomware dedicated to the automotive vehicles as presented by Weiss. This research presented us with different automotive ransomware properties[19].

1. Data encryption

2. Infection of automotive component
3. Vehicle process interruption
4. Download
5. Payment request

Another vulnerable component of a self-driving car is there major communication protocol between ROS and different sensors. This massaging protocol is based on subscribe/publish scheme. Ivan Vaccari developed a DoS attack that are targeted to MQTT protocol. This attack instantiates high number of connections to the server to seize all available connection[20].In an article published by Avast showed that 49000 exposed MQTT server were accessible from internet. Among them 32000 had no password protection. Researchers were able to get access to all kinds of exposed data[21].This is currently one of the major security concern over MQTT communication, zero or weak authentication . In 2016, ISO and SAE jointly published a cyber security standard for road safety vehicle called ISO/SAE DIS 21434. This publication set out an minimum security criteria for cybersecurity Engineering[22]. Based on this standards a lot of researches had been conducted. In 2021 IEEE 93rd Vehicular Technology Conference , a research paper described a security testing process for automotive technology. They explained the process of creating System Under Test based on attack surface analysis , generating different type of test scenario and proper reporting[23]. Another group of researcher proposed digital twin based security testing for automotive car[24].

Although these testing approaches can provide information regarding components that is vulnerable to existing vulnerability, they will not provide any threat intelligence for zero-day vulnerability. Upstream security conducted a survey in 2018 that found 92 percent of black hat attacks are conducted via long distance wireless attacks. Karamba security[25] in 2018 developed a virtual ECU honeypot that was attacked by 300000 times by 3500 attacks. Although those attacks are not targeted for ECU, major attack target was telnet, SSH and HTTPS services.In 2018, IoT honeypots developed by Kaspersky researchers showed 75percent IoT attacks are targeted towards telnet and 12 percent for SSH[26]. In Metongnon, they found out some attackers are already looking for UpnP or MQTT protocol. A lot of research has been put in place for understanding attack methods on IoT devices through honeypot. Based on these researches Maria Schmitz proposed the following assumption[27]:

1. Cyberattacks on connected cars are increasing
2. Majority of these attacks target Telnet or SSH.
3. Dictionary and brute force attacks are the most common method.

To prevent this kind zero-day attack one of the technique is to have a deception mechanism placed in the network. Ferar and Bahsi in their paper[28], created deception mechanism including some known vulnerability and honeypots and tested this with red team . Their paper concluded that attackers often got confused to identify resources and services , they somewhat did not think that those were honeypots and could not finish the mission due to lack of time. To create a cyber deception operation Attivo networks proposed the following steps[10]:

1. Planning
2. Preparing
3. Monitoring
4. Measuring

One of the examples of a production honeypot for UAV was developed in 2020 by researcher in Technische Universitat Darmstadt. They proposed a novel medium interaction honeypot that can provides a medium interaction honeypot for UAV devices, record and analyze attack and guide attackers away from UAV and delay them [29]. They designed a portable honeypot that based on Rasberry pi supporting SSH/telnet, emulates MAVLink that is virtually indistinguishable from actual drone[29]. The difference between UAV and AV is in there communication protocol . UAV usually uses MAVlink to communicate with its control center where self-driving cars are mostly uses 4G or 5G router to communicate with control center[29].

3. Proposed Solution

3.1 Requirement

3.1.1 Honeypot Requirement

The primary goal of a honeypot is to deceive an attacker into thinking that they are in an actual target environment . As the author is planning to set up a production honeypot for self-driving cars it has to be as realistic as an actual self-driving car. As the author wants to detect attacker activity inside honeypot, low interaction honeypot can not hold an attacker inside much longer as they lack the functionality to keep attacker occupied for longer period of time. For these reasons it is important to develop a high interaction honeypot. For the design author is simulating an operating system and hardware that hosts the middleware i.e ROS. It is also connected to different sensors and executes several functions to perform a task.

Currently, there are different OS's used in self-driving cars. The most common OS are Linux based operating system i.e Ubuntu. Other than this famous automakers use following OS in there cars.

1. QNX Neutrino
2. WindRiver VxWorks
3. NVIDIA DRIVE™ OS

Autonomous vehicles have a lot of ECU's that needs to be communicated to and managed as a single entity. Big automakers like GM, ford uses ROS on there cars. For honeypot it will be essential to install a ROS in main OS .But presence of ROS does not make it a self-driving car. for that we also have to simulate messages generated from different ECU and sensors.

Next thing needed to be done is to give attacker a sense that it is working as a MQTT server. MQTT is an Internet Of Thing connectivity protocol. It is often used to communicate between IoT devices, also to push data from cloud to device.Automotive industries are using this protocol to communicate between sensors and other data point. After setting up MQTT server it is important to simulate different sensors that can be communicated through this protocol. This sensors and raw data coming from it will make attacker think that they are actually inside a self-driving cars HMI that is connected and controlling different sensors.To make attacker slow author needs to set up processes that interact with various commands from the attacker.

Now that a list of services required to run in honeypot it is necessary to make the proposed honeypot attractive to an attacker. One of the first steps of a cyber attack is to Reconnaissance. In this step attacker tries to gather information they can use . One of the technique for reconnaissance is to do active scanning. In this technique, an attacker might scan ip blocks to gather information to be used to target a victim. Usually, attackers gathers public ip's that are allocated to a specific organization or they do sequential ip address scan. For the proposed honeypot author needs to have an open SSH port that will be open and are vulnerable to brute force or dictionary attack or simply having a week authentication set up for attacker to get access to the system. In addition to having SSH open author can also add an open telnet port for the attacker to use.

It is also needed to enable remote logging in the proposed honeypot to send logging data to a remote rsyslog relay for further investigation and analysis. It needs to send authentication logs, command executed logs, downloaded file information to rsyslog relay.

After understanding basic requirement for self-driving car, author has to decide on hardware. The most important thing have to keep in mind is that for this honeypot it is not possible to use high performance computers as those are not cheap and quite often not very portable. For the proposed honeypot author needs something portable and cheap but has manageable RAM and storage.

1. Operating system : Ubuntu
2. Services: ROS
3. Vulnerable port: SSH , Telnet
4. Remote logging: Rsyslog
5. Hardware: Portable, cheap with manageable RAM and storage.

3.1.2 Open source Honeypot

Currently, there are lots of low to high interaction honeypots present in the market. Most of them are open source and free to use. To trap attacker's SSH activity there is Cowrie a low interaction ssh/telnet honeypot. Conpot is a high interaction SCADA honeypot that can mimic IoT devices such as power plants. But the problem with conpot is its templates are limited and impossible to simulate a self-driving cars environment. Glastopf works as web application honeypot to catch malware's and cross-site scripting in web applications. T-pot honeypot is a multipurpose honeypot that contains more then 10 docarized honeypot that all have there own instances. Now, for our self-driving car honeypot author can use above mentioned honeypots but the problem with those honeypots are if a attacker does a targeted attack on a self-driving, honeypots will be very easily exposed as it does not contain enough proof or footprint of a self-driving car. Thus either attacker will leave the system or try to destroy or break through the system. For this reason, we need to develop our own honeypot that will simulate as much services of AV to fool attackers.

3.2 Design

In this section author will design our honeypot based on the requirement described above.

3.2.1 Duckiebot

For the proposed honeypot hardware author will use Duckiebot. Duckiebots are cheap, small size self-driving car that can be built using off-the-shelf parts. The Duckiebot is a minimal anatomy platform that allows to investigate complex behavior and do research on autonomous vehicle[30].

This bot contains following parts:

1. Computation: JN2GB
2. Sensing: Camera, Wheel Encoders, ToF, IMU
3. Actuator: 2xDC motors , 4xRGB LED, Screen
4. Memory: 64GB
5. Power: Battery

3.2.2 Hardware setup

Duckiebot comes in a box with all its components. Following picture gives an overview of all components: Below steps are followed based on Duckiebot documentation to assemble the robot.

1. Step 1: Unboxing
2. Step 2: Drive train
3. Step 3: Battery pack installation

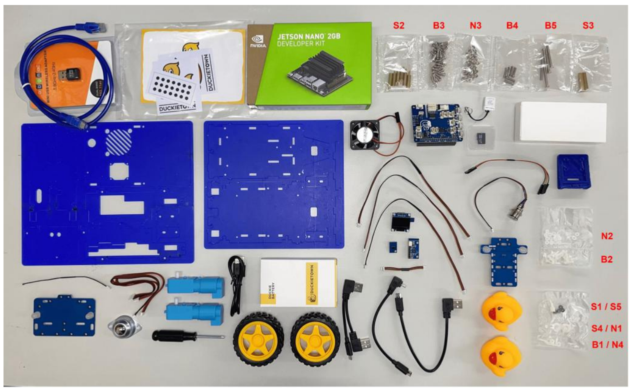


Figure 12. Overview of all parts[30]

4. Step 4: Computational Unit and Rear Assembly
5. Step 5: Cable management
6. Step 6: Front assembly
7. Step 7: Top plate assembly
8. Step 8: Charging

The following figure shows the complete car after assembly.

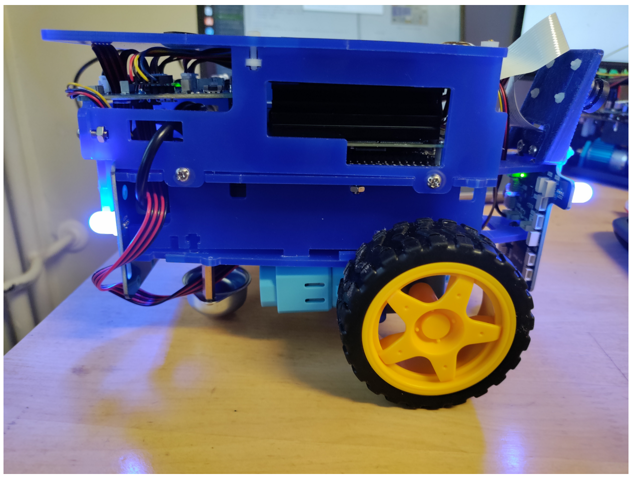


Figure 13. Duckibot

3.2.3 Software setup

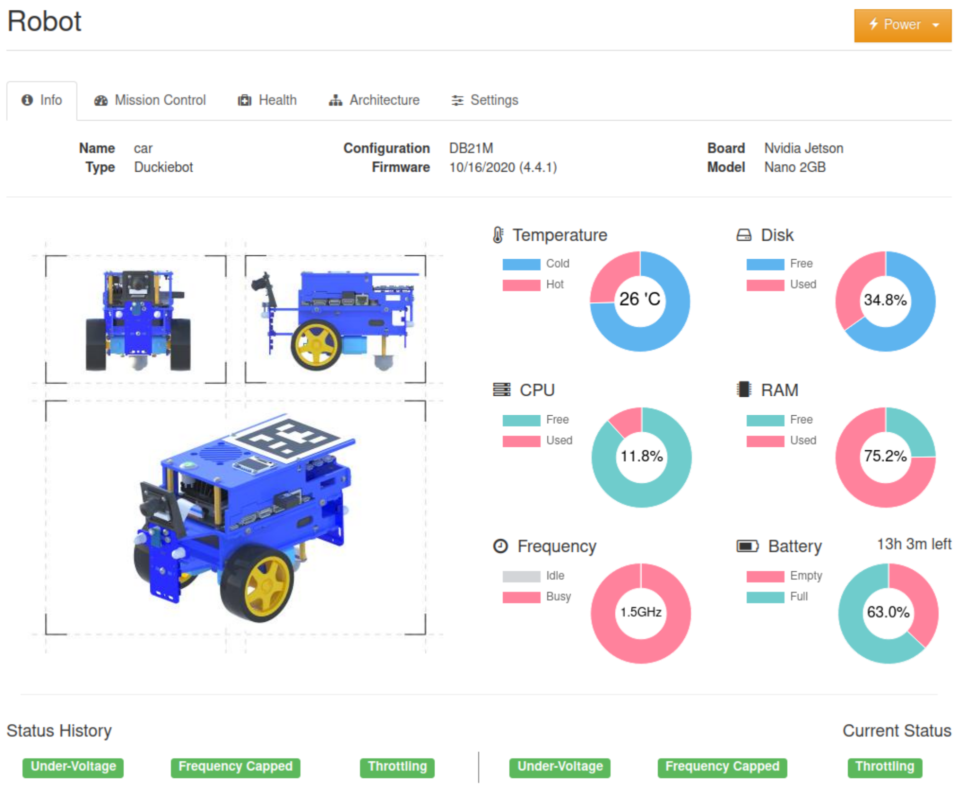
Duckietown software setup requires native Ubuntu 20.04 installed. It will also work in a virtual machine with Ubuntu installed. It can also be installed using windows or mac OS but for the experiment Ubuntu 20.04 LTS will be installed in personal laptop. Before moving forward with software setup the author needs to make sure that all necessary packages are installed such as: pip3, git, git-lfs, curl , docker and docker-compose. Now using pip3 Duckietown shell was installed using following commands:

```
pip3 install --no-cache-dir --user --upgrade duckietown-shell
dts --set-version daffy
```

The author needs to create a account in Duckibot website to acquire a token which will be needed in the later stage. before moving on to flashing the sd card token was validated in Duckitown shell. Following commands in dts shell was used to initialize SD card:

```
 dts init_sd_card --hostname Car --type duckiebot \
--configuration DB21M
```

After sometimes it will give a confirmation massage that SD card is ready , it can then be removed and place it in the SD card holder in Duckiebot. To check that Duckibot is ready we can go to http://Car.local to visit its dashboard. After initializing Duckiebot



Robot

Name	car	**Configuration**	DB21M	**Board**	Nvidia Jetson
Type	Duckiebot	**Firmware**	10/16/2020 (4.4.1)	**Model**	Nano 2GB

Figure 14. dashboard

understanding of its Hardware architecture is required.Figure 17 shows the hardware diagram of Duckiebot:

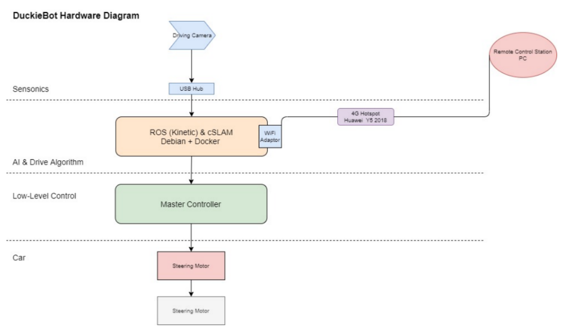


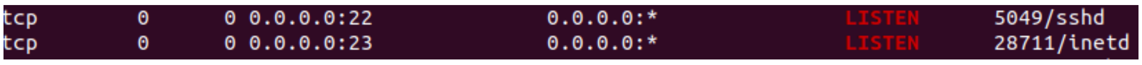
Figure 15. Duckiebot architecture[30]

3.2.4 SSH and Telnet Configuration

After successfully booting the car for the first time author can SSH to it from lab environment . Initially default SSH username and password were used. After successfully logged into the car author changed password. Then to enable weaker SSH authentication author created a user "Admin" and give it a password "admin123". respectively several different users such as "root1" , "systemAdmin" etc was created with different password that will be easily breakable using dictionary attacks or brute force attack. Following steps are taken to create this process:

```
# usermod -aG sudo admin
```

For SSH author is using default port 22. Author also installed telnet and enabled it to listen to in port 23. So, now it has two ports 22 and 23 listening to SSH and telnet connections.



```
tcp        0      0 0.0.0.0:22            0.0.0.0:*               LISTEN      5049/sshd
tcp        0      0 0.0.0.0:23            0.0.0.0:*               LISTEN      28711/inetd
```

Figure 16. SSH and telnet

3.2.5 File system

Duckiebot comes with standard Ubuntu file system .After logged in to the machine user will land in regular Ubuntu user home directory.

```
$pwd
$/home/admin
```

By changing directory to /data ROS directory can be accessed. in this directory there are six additional sub directory.

```
$ cd /data
$ ls
autoboot bags config logs proc stats
```

config directory contains Duckibot configuration and calibration files. It also contains hardware information.

```
$ cd /data/config
$ ls
calibration permissions robot_hardware robot_configuration
```

In the configuration files, it is possible to add malicious ROS codes to change robot configuration. Calibration folder contains different calibrations done for the self-driving car. It will be possible for the attacker to change those calibrations to make the car misbehave. All flight logs of the bot is saved in logs files in the same directory. In this logs attacker can check previous log data of the car, analyze it and even send it to a remote location for further analysis. For privilege escalation attacker can view user information and change passwords for different user to get root privilege. With this privilege attacker can encrypt data to act as a ransomware.

3.2.6 MQTT simulation

Most of modern day cars use MQTT protocol to communicate with different sensors. But Duckiebot does not have this functionality, rather it works using different docker images to control its camera and kinetics. For that author needs to simulate MQTT client environment in Duckiebot. For this part Mosquitto broker is used to implement MQTT protocols. For this part author will need a broker and client will be Duckiebot. Author installed mosquitto in personal computer and installed client in Duckiebot.

```
$ sudo apt-get install mosquitto
$ sudo apt-get install mosquitto-clients
```

Now it is needed to publish sensor data in the broker. For that author first decided on four sensor data we will simulate.

1. Mass airflow sensor

2. Engine Speed Sensor
3. Voltage sensor
4. Vehicle Speed Sensor

Now from broker author publish messages for all four topics listed above. Below is published message on four topics :

```
$ mosquitto_pub   -t /Car/sensors/MAS -m " 18g/s"
$ mosquitto_pub   -t /Car/sensors/ESS -m " 700 rpm"
$ mosquitto_pub   -t /Car/sensors/VS  -m " 20V"
$ mosquitto_pub   -t /Car/sensors/VSS -m " 65m/s"
```

For a constant massage publishing cronjob was added that will publish this message with 10s interval with slightly changed data. Now we need to subscribe to this topic from our Duckiebot. Following commands are used to subscribe to the massages published by broker.

```
$ mosquitto_sub  -t /Car/sensors/MAS
$ mosquitto_sub  -t /Car/sensors/ESS
$ mosquitto_sub  -t /Car/sensors/VS
$ mosquitto_sub  -t /Car/sensors/VSS
```

A cronjob was also added in the Duckiebot to receive published message from the broker.

3.2.7 Malware detection

Author also wants to detect if any attacker downloaded malware or virus in our system . For that author needs to set up malware detection in honeypot. For that clamAV was installed in Duckiebot. This will detect and analyze files that are downloaded for any kind of malware or virus. A cron job is added in /etc/crontab to run a scan every night and infected file will be quarantined in a specific folder. For experiment author will use /home/duckie/clanav to move infected files. With this tool if attacker downloads any virus or malware in to our honeypot , it will be detected and all files will be stored in a specific location. Following crontab was added to run daily at 12 am.

```
    * */12 * * * clamscan -r --move=/  /home/duckie/clanav
    >/dev/null 2>&1
```

3.2.8 Logging

One of the important design part is to implement remote logging of our honeypot. For that author installed and updated rsyslog in Duckiebot. It will work as a rsyslog client and will send its log to a remote server for further analysis. For that author set up a rsyslog relay server in home computer. It will listen in port 6514 . Communication between client and server will be done over TLS to prevent any man in the middle attacks. To enable TLS author created own self signed certificates in server side and copy the chain.pem file to client. After TLS connection is established client rsyslog configuration file is modified so that following files are sent to remote server. Author also set up a UDP connection on port 514 for troubleshooting .

1. auth*
2. commands.log
3. scan.txt

Here , commands.log contains all commands done by a user in one SSH session. As Ubuntu does not provide any built in service to log this info, we added a script in bashrc file to save all commands as a local6 facility and the in rsyslog.conf author instructed it to save it to /var/log/command.log. auth log contains all information related to users SSH activity. It can be either successful or unsuccessful attempts, ip address and time. In syslog relay author configured it to send all log files to logstash. Author then configured grafana to visualize and send alert.

3.2.9 Restriction

The author also needed to restrict attacker for reading and writing certain folders and files such as our log files, log configuration etc . For that author implemented SSH restriction to individual users except primary user. Author did this by creating a new group called "restrictions" and then put following configurations lines in /etc/ssh/sshd :

```
Match Group restrictions
  ForceCommand internal-sftp
  ChrootDirectory /var/log/
  # Disable tunneling, authentication agent, TCP and X11 forwarding.
  PermitTunnel no
  AllowAgentForwarding no
  AllowTcpForwarding no
```

```
X11Forwarding no
```

Author created similar rule for /etc/rsyslog.conf and /var/spool/cron .

3.2.10 Backup and restore

After setting up all services in Duckiebot it is needed to create a backup of the whole system. As it is known, Duckiebot uses Jetson nano developer kit, which uses SD card rather then standard HDD or SSD , tools like clonezilla will not work here. Opensource debian based snapshot tool such as backintime is a GUI based application so it also will also not work in this case. Another commonly used tool is timeshift, but its package does not exit for arm64/aarch64/ARMv8-a architecture. To use this tool author have to compile the package from source code. For these reasons author will resort to dd tool found in linux to create a backup image of SD card. For this author will save image in a central repository so that it can be cloned to a new SD card at any moment of time. Following command is used to create a backup of SD card.

```
$ sudo dd if=/dev/sda conv=sync,noerror bs=64K
| gzip -c > ~/backup_image.img.gz
```

here /dev/sda is device name. Following figure shows our newly created backup image.

Figure 17. Backup image

It is now possible to the restore image to a fresh SD card using following command.

```
$ gunzip -c ~/backup_image.img.gz
| dd of=/dev/sdc bs=64K
```

3.3 Implementation

As the proposed honeypot is a high interaction it is not safe to place it inside cars architecture. Rather it can be connected to the outside network where data center, remote monitor server are placed as these services are primarily more attractive to attacker. If some one gets access to those services rather then letting the attacker get access to the car we can deceive them to access our Duckiebot instead. While attacker gets inside our honeypot an alert system will be triggered to protect our actual car infrastructure.

3.4 Discussion

In this chapter, author developed a list of requirement to develop a high interaction honeypot. It was seen that to develop a HIH, it is not very feasible to use current honeypot tools that are currently being used as honeypots in different network infrastructure services.Due to self-driving cars have different complex services we developed a prototype honeypot that can be used as a honeypot. Author also developed remote monitoring and alerting for this honeypot. In case an intruder take control of the honeypot a alert will be triggered in email and discord. This shows that honeypot can be used as a detection honeypot for self-driving car.

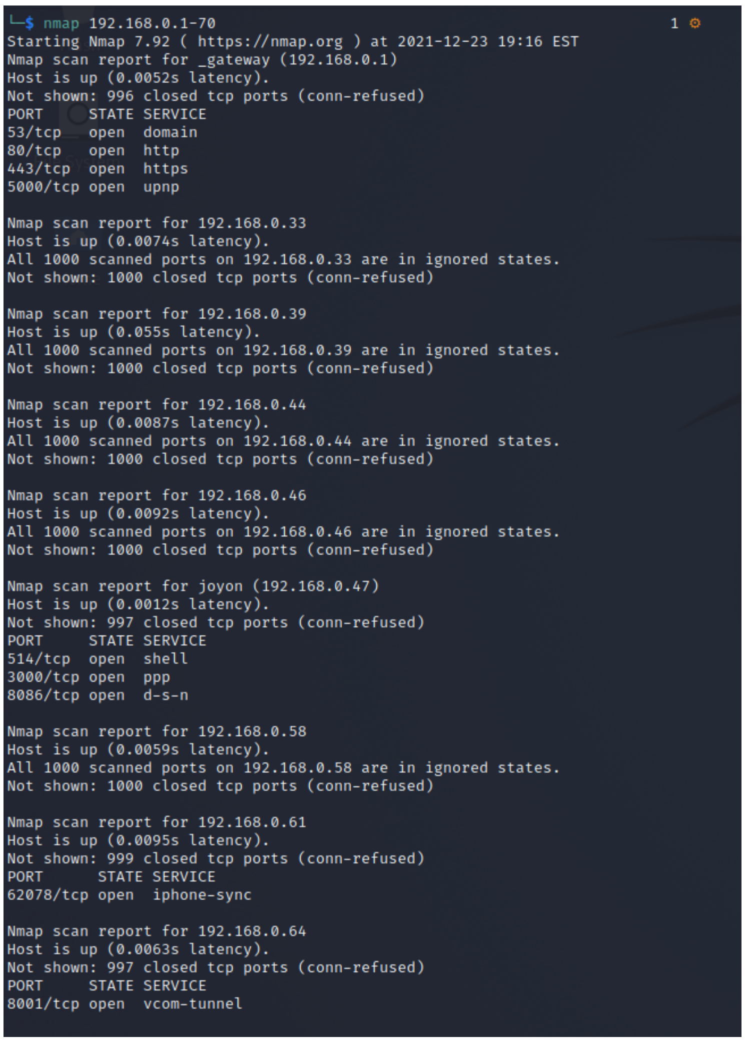
3.5 Future Work

In future work, it will be beneficial to add additional services to our honeypot so that attacker can interact with ROS and simulate an environment so that the car can be controlled from the honeypot. As author used Duckiebot as honeypot it can be also a important research to do it in a conventional micro-controller i.e Rasberry pi and try to simulate additional services such as CAN and embedded component.

4. Evaluation

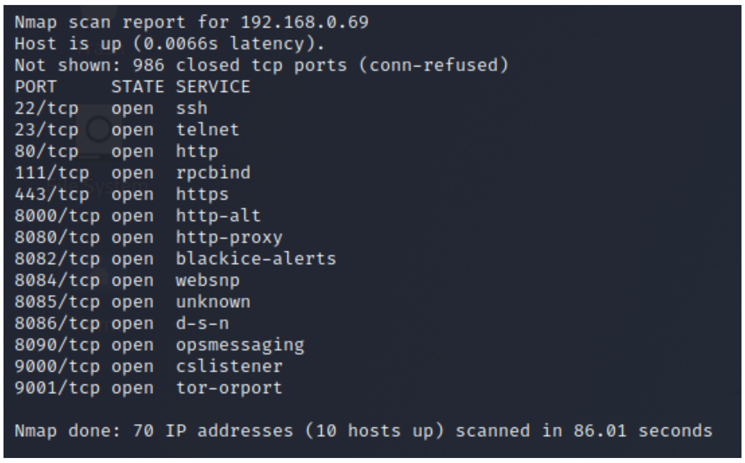
4.1 Experiment

In this section, author will evaluate the proposed honeypot. For this experiment it is assumed that attacker is on the same network as our car. This experiment will be conducted using a Kali Linux virtual machine. According to MITTRE ATT&CK matrix, first step of a successful attack is to do a active scanning. A nmap scanning with host range was conducted.Following figure 19 and 20 shows us nmap scanning and its result.



```
└─$ nmap 192.168.0.1-70                                          1 ⚙
Starting Nmap 7.92 ( https://nmap.org ) at 2021-12-23 19:16 EST
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0052s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT     STATE SERVICE
53/tcp   open  domain
80/tcp   open  http
443/tcp  open  https
5000/tcp open  upnp

Nmap scan report for 192.168.0.33
Host is up (0.0074s latency).
All 1000 scanned ports on 192.168.0.33 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.0.39
Host is up (0.055s latency).
All 1000 scanned ports on 192.168.0.39 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.0.44
Host is up (0.0087s latency).
All 1000 scanned ports on 192.168.0.44 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.0.46
Host is up (0.0092s latency).
All 1000 scanned ports on 192.168.0.46 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for joyon (192.168.0.47)
Host is up (0.0012s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE
514/tcp  open  shell
3000/tcp open  ppp
8086/tcp open  d-s-n

Nmap scan report for 192.168.0.58
Host is up (0.0059s latency).
All 1000 scanned ports on 192.168.0.58 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.0.61
Host is up (0.0095s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE
62078/tcp open  iphone-sync

Nmap scan report for 192.168.0.64
Host is up (0.0063s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE
8001/tcp open  vcom-tunnel
```

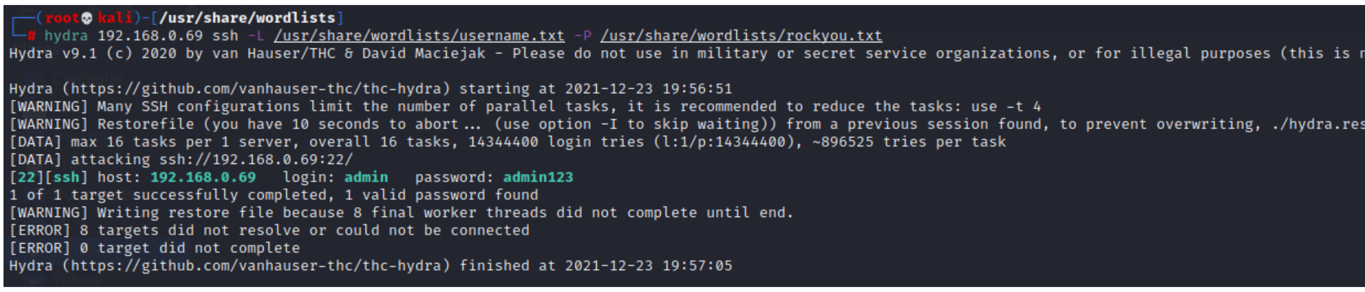
Figure 18. Active scanning of network-1



```
Nmap scan report for 192.168.0.69
Host is up (0.0066s latency).
Not shown: 986 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
80/tcp   open  http
111/tcp  open  rpcbind
443/tcp  open  https
8000/tcp open  http-alt
8080/tcp open  http-proxy
8082/tcp open  blackice-alerts
8084/tcp open  websnp
8085/tcp open  unknown
8086/tcp open  d-s-n
8090/tcp open  opsmessaging
9000/tcp open  cslistener
9001/tcp open  tor-orport

Nmap done: 70 IP addresses (10 hosts up) scanned in 86.01 seconds
```

Figure 19. Active scanning of network-2

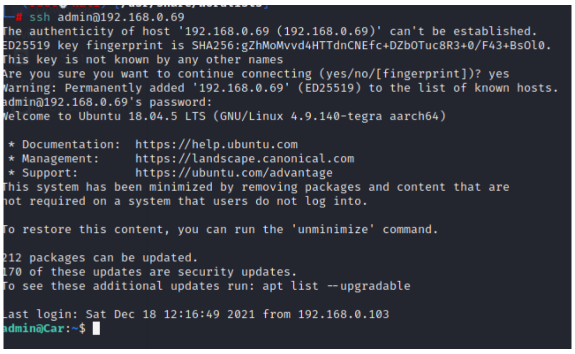
From these it is visible that only one host 192.168.0.69 has open SSH and telnet port open. For next step , a brute force login attempt on port 22 was conducted using hydra.For this a list of usernames and passwords were used. Figure 21 shows hydra found username and password for host 192.168.0.69. Username is admin and password admin123. Now using



```
┌──(root💀kali)-[/usr/share/wordlists]
└─# hydra 192.168.0.69 ssh -L /usr/share/wordlists/username.txt -P /usr/share/wordlists/rockyou.txt
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is n

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-12-23 19:56:51
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.res
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344400 login tries (l:1/p:14344400), ~896525 tries per task
[DATA] attacking ssh://192.168.0.69:22/
[22][ssh] host: 192.168.0.69   login: admin   password: admin123
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 8 final worker threads did not complete until end.
[ERROR] 8 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-12-23 19:57:05
```

Figure 20. Brute Force attempt

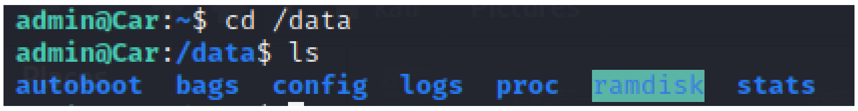
this username and password successful SSH connection was established to the host. After



```
└─# ssh admin@192.168.0.69
The authenticity of host '192.168.0.69 (192.168.0.69)' can't be established.
ED25519 key fingerprint is SHA256:gZhMoMvvd4HTTdnCNEfc+DZbOTuc8R3+0/F43+BsOl0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.69' (ED25519) to the list of known hosts.
admin@192.168.0.69's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.140-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

212 packages can be updated.
170 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Sat Dec 18 12:16:49 2021 from 192.168.0.103
admin@Car:~$ █
```

Figure 21. Successful SSH login

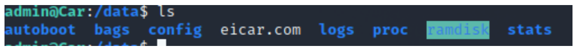
successful login going to the /data folder all important file system of ROS is visible. Here, bags subscribed to different ROS topics and store massage data. From here it is possible to insert fake data in the bag, encrypt or even delete the data . This operation will make the car behave according to the attackers data or even shut down or misbehave suddenly.



```
admin@Car:~$ cd /data
admin@Car:/data$ ls
autoboot  bags  config  logs  proc  ramdisk  stats
```

Figure 22. ROS file system visible to attacker

Due to lack of resource this part of the experiment was not conducted but theoretically in a running duckiebot all this tasks can be possible by the attacker. For next step, a malware was downloaded in the file-system . For safety purpose, rather then downloading a actual malware , a fake malware eicar.com was downloaded from eicar.org and saved in the /data folder. As this user does not have superuser privileges , any kind of privilege escalation



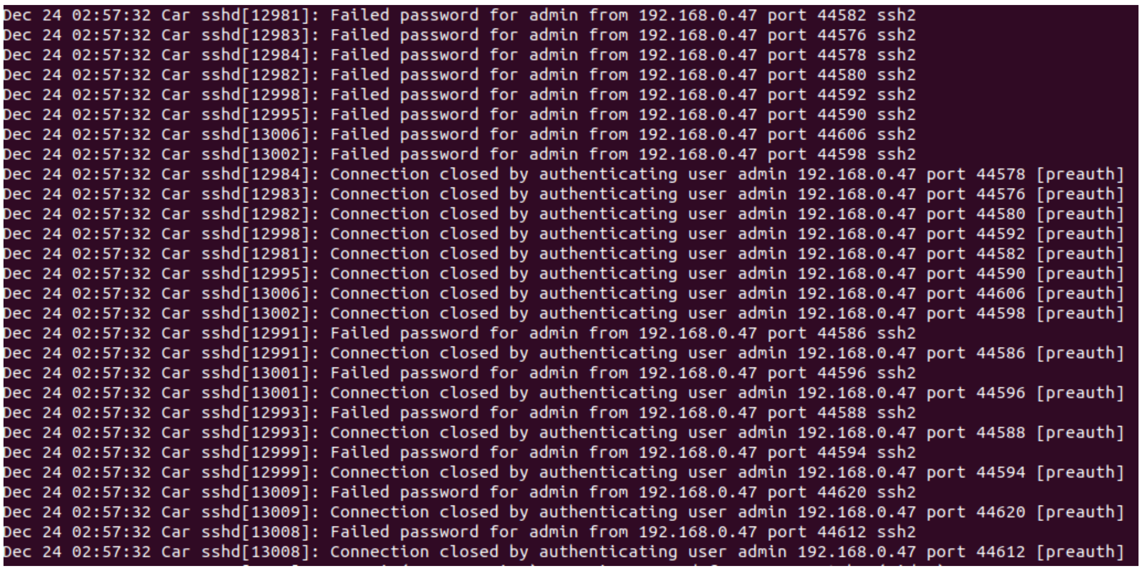
```
admin@Car:/data$ ls
autoboot  bags  config  eicar.com  logs  proc  ramdisk  stats
```

Figure 23. Malware file

was not possible.From the car any kind of scanning was also not possible .

4.2 Result

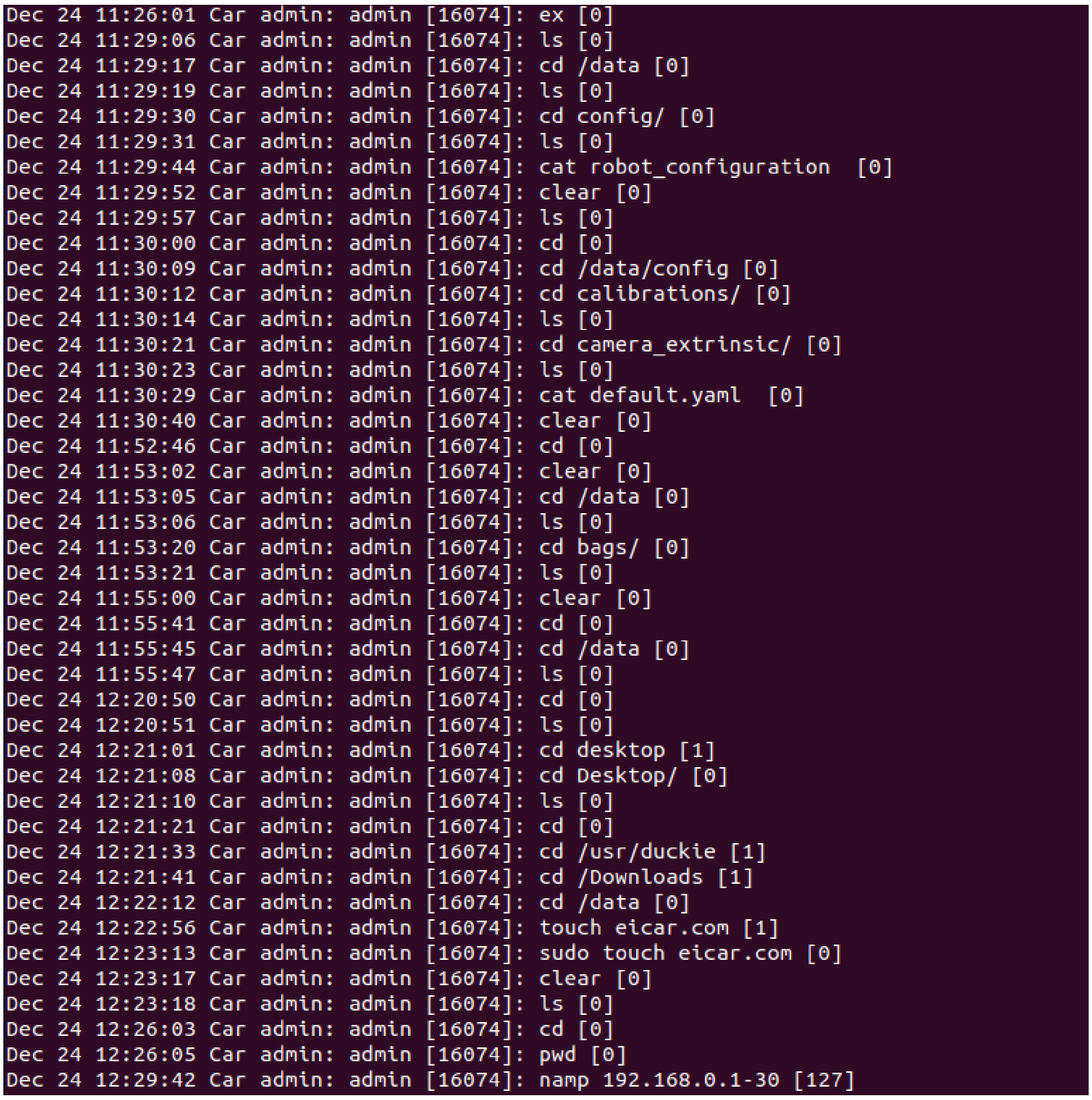
Above experiment shows us that our honeypot is working as intended, user can interact with it and everything is logged in remote syslog server.Following figure shows the logging data from our syslog relay: The user can also download files over http using wget. But



```
Dec 24 02:57:32 Car sshd[12981]: Failed password for admin from 192.168.0.47 port 44582 ssh2
Dec 24 02:57:32 Car sshd[12983]: Failed password for admin from 192.168.0.47 port 44576 ssh2
Dec 24 02:57:32 Car sshd[12984]: Failed password for admin from 192.168.0.47 port 44578 ssh2
Dec 24 02:57:32 Car sshd[12982]: Failed password for admin from 192.168.0.47 port 44580 ssh2
Dec 24 02:57:32 Car sshd[12998]: Failed password for admin from 192.168.0.47 port 44592 ssh2
Dec 24 02:57:32 Car sshd[12995]: Failed password for admin from 192.168.0.47 port 44590 ssh2
Dec 24 02:57:32 Car sshd[13006]: Failed password for admin from 192.168.0.47 port 44606 ssh2
Dec 24 02:57:32 Car sshd[13002]: Failed password for admin from 192.168.0.47 port 44598 ssh2
Dec 24 02:57:32 Car sshd[12984]: Connection closed by authenticating user admin 192.168.0.47 port 44578 [preauth]
Dec 24 02:57:32 Car sshd[12983]: Connection closed by authenticating user admin 192.168.0.47 port 44576 [preauth]
Dec 24 02:57:32 Car sshd[12982]: Connection closed by authenticating user admin 192.168.0.47 port 44580 [preauth]
Dec 24 02:57:32 Car sshd[12998]: Connection closed by authenticating user admin 192.168.0.47 port 44592 [preauth]
Dec 24 02:57:32 Car sshd[12981]: Connection closed by authenticating user admin 192.168.0.47 port 44582 [preauth]
Dec 24 02:57:32 Car sshd[12995]: Connection closed by authenticating user admin 192.168.0.47 port 44590 [preauth]
Dec 24 02:57:32 Car sshd[13006]: Connection closed by authenticating user admin 192.168.0.47 port 44606 [preauth]
Dec 24 02:57:32 Car sshd[13002]: Connection closed by authenticating user admin 192.168.0.47 port 44598 [preauth]
Dec 24 02:57:32 Car sshd[12991]: Failed password for admin from 192.168.0.47 port 44586 ssh2
Dec 24 02:57:32 Car sshd[12991]: Connection closed by authenticating user admin 192.168.0.47 port 44586 [preauth]
Dec 24 02:57:32 Car sshd[13001]: Failed password for admin from 192.168.0.47 port 44596 ssh2
Dec 24 02:57:32 Car sshd[13001]: Connection closed by authenticating user admin 192.168.0.47 port 44596 [preauth]
Dec 24 02:57:32 Car sshd[12993]: Failed password for admin from 192.168.0.47 port 44588 ssh2
Dec 24 02:57:32 Car sshd[12993]: Connection closed by authenticating user admin 192.168.0.47 port 44588 [preauth]
Dec 24 02:57:32 Car sshd[12999]: Failed password for admin from 192.168.0.47 port 44594 ssh2
Dec 24 02:57:32 Car sshd[12999]: Connection closed by authenticating user admin 192.168.0.47 port 44594 [preauth]
Dec 24 02:57:32 Car sshd[13009]: Failed password for admin from 192.168.0.47 port 44620 ssh2
Dec 24 02:57:32 Car sshd[13009]: Connection closed by authenticating user admin 192.168.0.47 port 44620 [preauth]
Dec 24 02:57:32 Car sshd[13008]: Failed password for admin from 192.168.0.47 port 44612 ssh2
Dec 24 02:57:32 Car sshd[13008]: Connection closed by authenticating user admin 192.168.0.47 port 44612 [preauth]
```

Figure 24. Log file-1

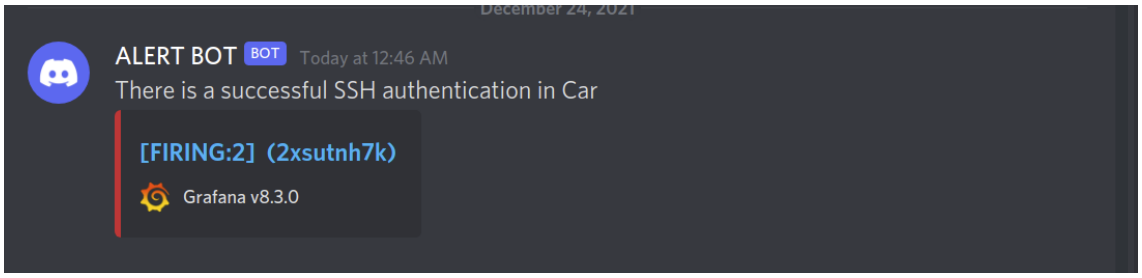
user was prevented from actually executing anything as a non root user. After pen testing is completed tells us that attacker will not think it is a honeypot as he can do a lot of things inside it as non root user which will not be possible in a honeypot. It is also possible that it is running some kind of ROS as /data shows some configuration and log files .Attacker can



```
Dec 24 11:26:01 Car admin: admin [16074]: ex [0]
Dec 24 11:29:06 Car admin: admin [16074]: ls [0]
Dec 24 11:29:17 Car admin: admin [16074]: cd /data [0]
Dec 24 11:29:19 Car admin: admin [16074]: ls [0]
Dec 24 11:29:30 Car admin: admin [16074]: cd config/ [0]
Dec 24 11:29:31 Car admin: admin [16074]: ls [0]
Dec 24 11:29:44 Car admin: admin [16074]: cat robot_configuration  [0]
Dec 24 11:29:52 Car admin: admin [16074]: clear [0]
Dec 24 11:29:57 Car admin: admin [16074]: ls [0]
Dec 24 11:30:00 Car admin: admin [16074]: cd [0]
Dec 24 11:30:09 Car admin: admin [16074]: cd /data/config [0]
Dec 24 11:30:12 Car admin: admin [16074]: cd calibrations/ [0]
Dec 24 11:30:14 Car admin: admin [16074]: ls [0]
Dec 24 11:30:21 Car admin: admin [16074]: cd camera_extrinsic/ [0]
Dec 24 11:30:23 Car admin: admin [16074]: ls [0]
Dec 24 11:30:29 Car admin: admin [16074]: cat default.yaml  [0]
Dec 24 11:30:40 Car admin: admin [16074]: clear [0]
Dec 24 11:52:46 Car admin: admin [16074]: cd [0]
Dec 24 11:53:02 Car admin: admin [16074]: clear [0]
Dec 24 11:53:05 Car admin: admin [16074]: cd /data [0]
Dec 24 11:53:06 Car admin: admin [16074]: ls [0]
Dec 24 11:53:20 Car admin: admin [16074]: cd bags/ [0]
Dec 24 11:53:21 Car admin: admin [16074]: ls [0]
Dec 24 11:55:00 Car admin: admin [16074]: clear [0]
Dec 24 11:55:41 Car admin: admin [16074]: cd [0]
Dec 24 11:55:45 Car admin: admin [16074]: cd /data [0]
Dec 24 11:55:47 Car admin: admin [16074]: ls [0]
Dec 24 12:20:50 Car admin: admin [16074]: cd [0]
Dec 24 12:20:51 Car admin: admin [16074]: ls [0]
Dec 24 12:21:01 Car admin: admin [16074]: cd desktop [1]
Dec 24 12:21:08 Car admin: admin [16074]: cd Desktop/ [0]
Dec 24 12:21:10 Car admin: admin [16074]: ls [0]
Dec 24 12:21:21 Car admin: admin [16074]: cd [0]
Dec 24 12:21:33 Car admin: admin [16074]: cd /usr/duckie [1]
Dec 24 12:21:41 Car admin: admin [16074]: cd /Downloads [1]
Dec 24 12:22:12 Car admin: admin [16074]: cd /data [0]
Dec 24 12:22:56 Car admin: admin [16074]: touch eicar.com [1]
Dec 24 12:23:13 Car admin: admin [16074]: sudo touch eicar.com [0]
Dec 24 12:23:17 Car admin: admin [16074]: clear [0]
Dec 24 12:23:18 Car admin: admin [16074]: ls [0]
Dec 24 12:26:03 Car admin: admin [16074]: cd [0]
Dec 24 12:26:05 Car admin: admin [16074]: pwd [0]
Dec 24 12:29:42 Car admin: admin [16074]: namp 192.168.0.1-30 [127]
```

Figure 25. Log file-2

modify this ros files to take control over the ROS. Any kind scanning inside the system and to use wireshark to detect any network activity as a non root user was prevented, so it was not possible achieve that. User initially used nmap to scan open ssh and telnet connection inside the network. After that he used Hydra package in Kali Linux to brute force our ssh password, it took hydra using basic user:password combination txt file several attempts to get a matching username and password. As our user could not do much as a not root user we can add these user to sudo group to give them chance to use wide range of services inside our honeypot such as wireshark, burpsuit or nmap to see and exploit network information we simulated such as MQTT. When attacker successfully connected to the car , a alert was sent to our discord server. Author created this alert system using grafana and discord webhook.


December 24, 2021

ALERT BOT BOT Today at 12:46 AM
There is a successful SSH authentication in Car

[FIRING:2] (2xsutnh7k)
Grafana v8.3.0

Figure 26. Alert

4.3 Analysis

In this chapter author tested the proposed honeypot on a controlled environment.It was assumed that attacker is already inside the self driving car network and rather than letting him get inside actual self driving car operating system, attacker was lured inside the honeypot. It responded properly with attackers commands. Everything attacker did were recorded and sent to remote server. Depending on the log and alert was also generated and sent to email and discord channel.

4.4 Limitation

Although some of proposed requirements are meet in proposed prototype honeypot , due to lack of permission attacker can not do much interaction with the device. It was also clearly visible that attacker does not get proper control of ROS feature of the Duckiebot.To successfully archive those it is necessary to run the Duckiebot in a controlled lab environment.

5. Summary

This thesis sought to develop a high interaction honeypot as an intrusion detection system for self-driving car. During this research, author presented a list of requirements needed to create a honeypot for self-driving car.In this thesis author designed a honeypot as an intrusion detection system and evaluated it to analyze its services. During research it was discovered that using a conventional honeypot in a self-driving car is not very feasible as those honeypots do not have necessary services to simulate operating services running in SDC. Rather than installing those services we used Duckiebot to develop it as a honeypot. Author then evaluated it using MITRE ATT&CK matrix and generated logs and created alert to detect intruder activity inside proposed honeypot. Proposed honeypot worked as an intrusion detection system inside self-driving car network.While analyzing those data it shows some limitation, such as this honeypot is working as it is intended but it lacks some feature to actually understand attacker strategy for a targeted attack. This honeypot can be used as an intrusion detection system but more work needed to be done to make it actually worthwhile to implement it inside a self-driving car network.

Bibliography

[1] Charlotte Kosche. *How many connected cars are sold worldwide?* URL: https://smartcar.com/blog/connected-cars-worldwide/. (accessed: 19.10.2021).

[2] Chris Sanders. *Intrusion Detection Honeypot: Detection Through Deception*. 2020.

[3] Jan vom Brocke and Alan Henver. "Intruduction to Design Science Research". In: (2020). URL: https://www.researchgate.net/publication/345430098.

[4] William Law. *An Introduction to Autonomous Vehicles*. URL: https://towardsdatascience.com/an-introduction-to-autonomous-vehicles-91d61ff81a40. (accessed: 19.10.2021).

[5] *Perception Projects from the Self-Driving Car Nanodegree Program*. URL: https://medium.com/udacity/perception-projects-from-the-self-driving-car-nanodegree-program-51fb88a38ff9. (accessed: 19.12.2021).

[6] Stuff reporter. *More Efficient Lidar Sensing for Self-Driving Cars*. URL: https://www.shunlongwei.com/more-efficient-lidar-sensing-for-self-driving-cars/. (accessed: 19.10.2021).

[7] *ROS documentation*. URL: https://towardsdatascience.com/an-introduction-to-autonomous-vehicles-91d61ff81a40. (accessed: 19.12.2021).

[8] Yara Ahmed. "Preventing Vulnerabilities and Mitigating Attacks on the MQTT Protocol". In: (2020). DOI: https://www.diva-portal.org/smash/get/diva2:1540107/FULLTEXT01.pdf.

[9] URL: https://attack.mitre.org/. (accessed: 30.11.2021).

[10] Geoff Hancock. "Cyber Deception: How To Build A Program". In: (). URL: https://www.attivonetworks.com/documentation/Attivo_Networks-Cyber_Deception_GH.pdf.

[11] Savita Paliwal. “Honeypot: A Trap for Attackers”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 6 (2017). DOI: https://ijarcce.com/upload/2017/march-17/IJARCCE%20197.pdf.

[12] Dandy Kalma Rahmatullah, Suiya Michrandi Nasution, and Fairuz Azmi. “Implementation of low interaction web server honeypot using cubieboard”. In: *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)* (2016).

[13] Humio Glossary. *What is Log Management?* URL: https://www.humio.com/glossary/log-management/. (accessed: 19.10.2021).

[14] URL: https://www.rsyslog.com/. (accessed: 19.10.2021).

[15] URL: https://www.elastic.co/what-is/elk-stack. (accessed: 19.10.2021).

[16] Carsten Maple and Anh Tuan le. “A Connected and Autonomous Vehicle Reference Architecture for Attack Surface Analysis”. In: (2019).

[17] Sagar Behere and Martin Torngren. “A functional reference architecture for autonomous drivingA functional reference architecture for autonomous driving”. In: (2015). DOI: https://reader.elsevier.com/reader/sd/pii/.

[18] Se-Yeon Jeong and Yeong-Jin Kim. “A study on ros vulnerabilities and countermeasure”. In: (2017).

[19] Se-Yeon Jeong and Yeong-Jin Kim. “On threat analysis and risk estimation of automotive ransomware”. In: (2019).

[20] Ivan Vaccari and Maurizio Aiello. “SlowITe, a Novel Denial of Service Attack Affecting MQTT”. In: (2020).

[21] Martin Hron. *Are smart homes vulnerable to hacking?* URL: https://blog.avast.com/mqtt-vulnerabilities-hacking-smart-homes. (accessed: 30.10.2021).

[22] Georg Macher and Omar Veleder. “ISO/SAE DIS 21434 Automotive Cybersecurity Standard - In a Nutshell”. In: (2020). DOI: https://www.researchgate.net/publication/343790924_ISOSAE_DIS_21434_Automotive_Cybersecurity_Standard_-_In_a_Nutshell.

[23] Stefan Marksteiner, Nadja Marko, and Andre Smulders. “A Process to Facilitate Automated Automotive Cybersecurity Testing”. In: (2021).

[24] Stefan Marksteiner, Slava Bronfman, and Eddie Lazebnik. “Using Cyber Digital Twins for Automated Automotive Cybersecurity Testing”. In: (2021).

[25] Karamba Security. *"Karamba Security introduces ThreatHive solution for expedited detection of automotive cybersecurity vulnerabilities,* URL: `https://karambasecurity.com/static/pdf/Karamba-Security-ThreatHive-Announcement.pdf`. (accessed: 30.10.2021).

[26] Y.Shemblev M.Kuzin and V.Kuskov. *New trends in the world of IoT threats*. URL: `https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/`. (accessed: 30.10.2021).

[27] Maria Schmitz. "A Strategy for Vehicular Honeypot". In: (2019).

[28] Alexandria Farar and Hayretdin Bahsi. "A Case Study About the Use and Evaluation of Cyber Deceptive Methods Against Highly Targeted Attacks". In: (). URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8054640`.

[29] Emmanouil Vasilomanolakis and Jorg Daubert. "Don't Steal my Drone: Catching Attackers with an Unmanned Aerial Vehicle Honeypot". In: (2018).

[30] URL: `https://www.duckietown.org/guides`. (accessed: 30.10.2021).

Appendices

Appendix 1 - Non-exclusive licence for reproduction and publication of a graduation thesis

Appendix 2 - TalTech IseAuto Hardware Architecture

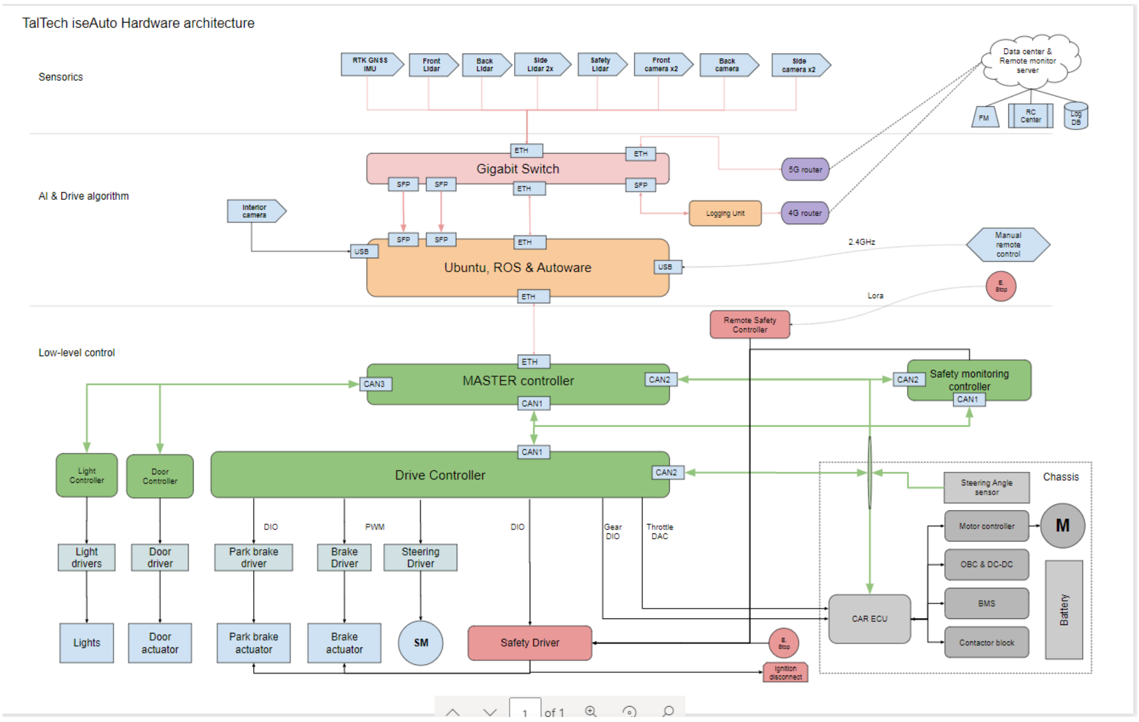


Figure 27. Hardware components of Taltech's SDC