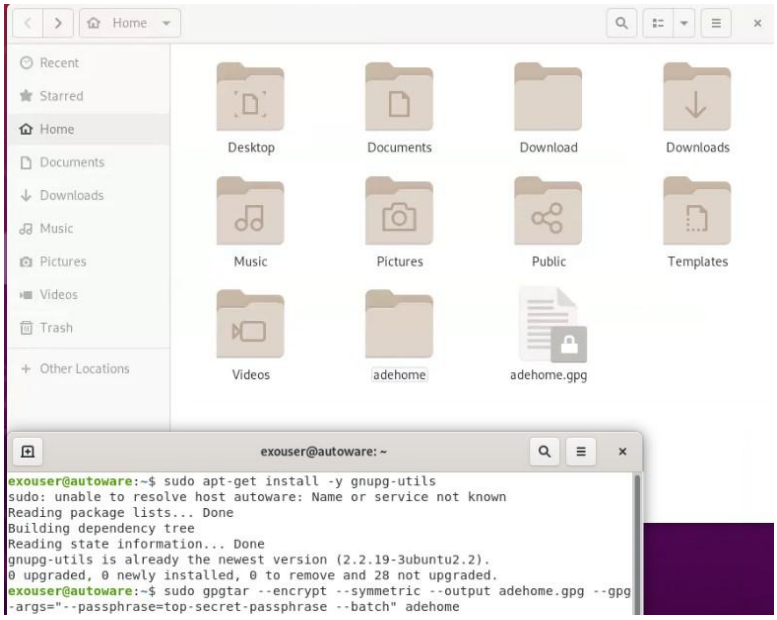
Abdulla Taymour
Ruth Delgado
Benjamin Thong
Ali Nadhaif
CYSE 465
December 7th 2022

Final Project

Task 1.2 (Optional) If you have proposed a defense technique, try to implement and test the effectiveness of the technique in Autoware.

This attack uses ros2's commands which normally maps the data fed from the sensors' nodes to the parameter files located in /opt/AutowareAuto/share/. In our case, the perpetrator was able to create a parameter file similar to the original one located in /opt/AutowareAuto/share/point_cloud_filter_transform_nodes/param/vlp16_sim_lexus_filter_transform.param.yaml, and remapped the nodes files to that file. The perpetrator was able to gain access to the files of this system, obtain the information and take away its integrity, by running malicious commands. Therefore, this clearly showed us that the system had no encryption and root access is easily gained. To mitigate this, we proposed a plan to encrypt the entire file, rendering only authorized entities able to run the program and have access to files. In our proposal,we used the tool 'gpgtar', which is a powerful cyber security encryption tool that offers symmetric encryption and decryption of directories in linux, which comes in handy because our virtual environment is a linux machine. After installing this tool we encrypted all of the contents inside the 'adehome' folder, this was necessary because all of its content is crucial to successfully run 'ade' and it is also where the attacker was able to create its malicious code and run it in our environment.



We also propose root access denial, this can be implemented to deny unauthorized entities access to the system. Additionally, it is well known that the human factor in cyber security plays a great role when it comes to defending against malicious attacks. One of the techniques we propose is checking the OBD-2 port every time the user starts the car to check for malicious attachments. Additionally linking node files with parameter files so that they can only run together, or embedding parameters into node files as one could be implemented by design is an extra layer of protection that can be added into the system.

Task 2. Write a report on the details of the attack. Your report should explain the threat model, attack implementation details, strength of the attack, and weakness/defense against the attack. The report should be at least 2000 words long. Your report should also contain the links to the github repos for the group members.

Running the Simulation:

This paper's computer simulation was based on Autoware that was run on Ubuntu20-Latest-NIVIDIA provided by George Mason University that was being run on Guacamole server. We used the Autoware own user manual found at this link, with it in the last project we implemented the 3D simulation successfully as well as analyzed how components of this program worked harmoniously to give us the result that it does at the moment. Dissecting and analyzing all of the sensors, operations, commands, etc. from this simulation was one of the strategies we utilized when it came to understanding the inner workings of this program. When running this simulation we have a full grasp of each element and we are capable of analyzing details that otherwise would be a daunting task due to all of its complexities. This leads us to the place at the project that we are at the moment where we understand the simulation, therefore we also understand its vulnerabilities.

Let us begin to explain how we run this simulation: We logged in to Exosphere provided by George Mason University to view our instance. The exosphere is an OpenStack-based web interface that provides us with a consistent user interface. For this specific purpose was allowing us to deploy our code and run it in a virtual machine that simulates a better computational power than some of our computers might have. This came in handy because the simulation needed a lot of space and power to run. Going back to the program, after we logged in, we were able to see all

of the projects we were working on at the moment.

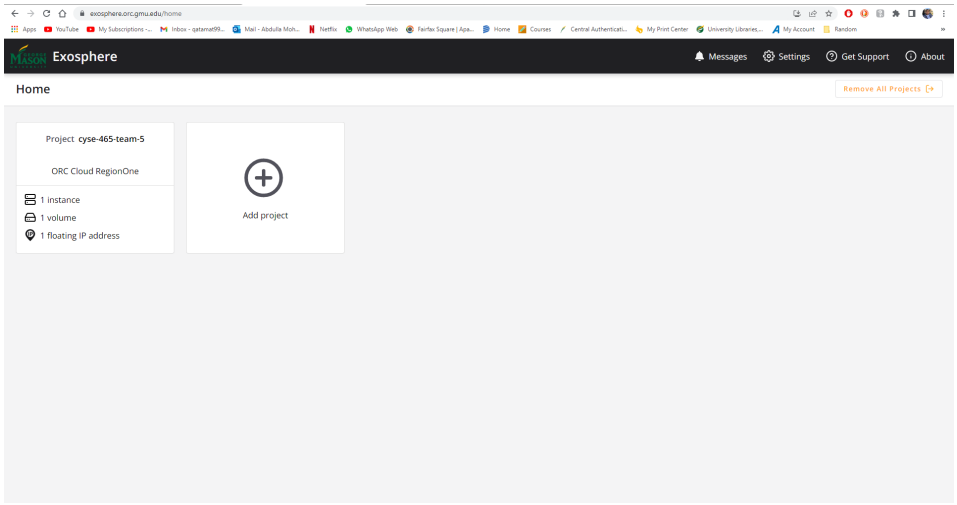


Fig. 1 Shows a screenshot of the Exosphere environment and the project created.

The course instructor had already created the project and instance. Then, we entered the project and viewed our instances.

As we can see in the figure above the professor had already created the project and instance for us, since he had to input the code. We entered the project and viewed the instances created so we could then work on it.

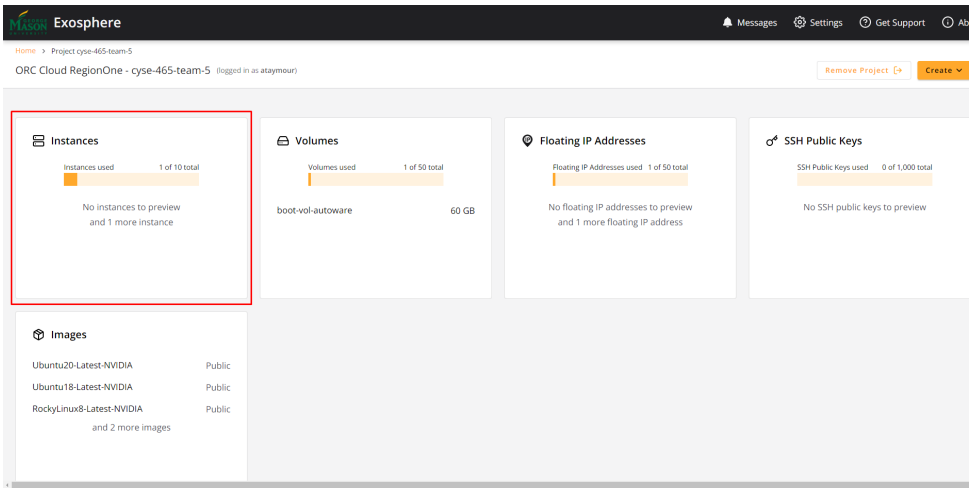


Fig. 2 Shows the Instances within the program. We can see there is 1 available as well as 1 volume and 1 Floating IP Address.

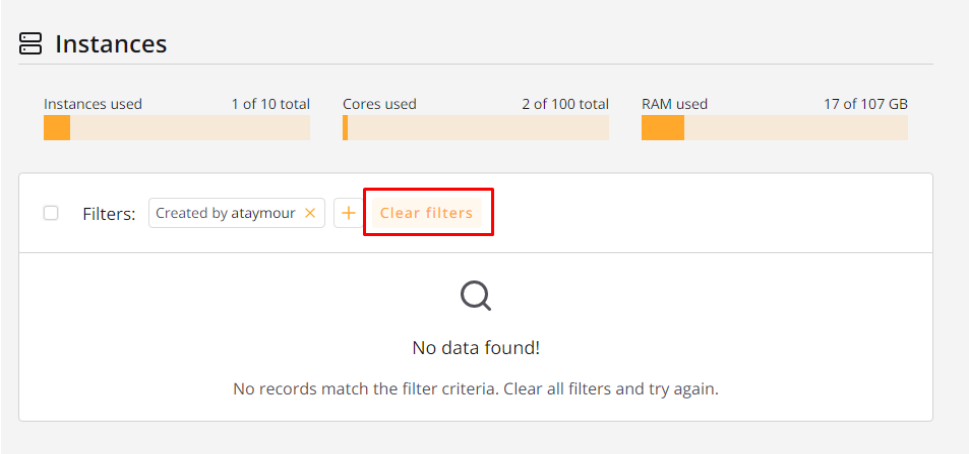


Fig. 3

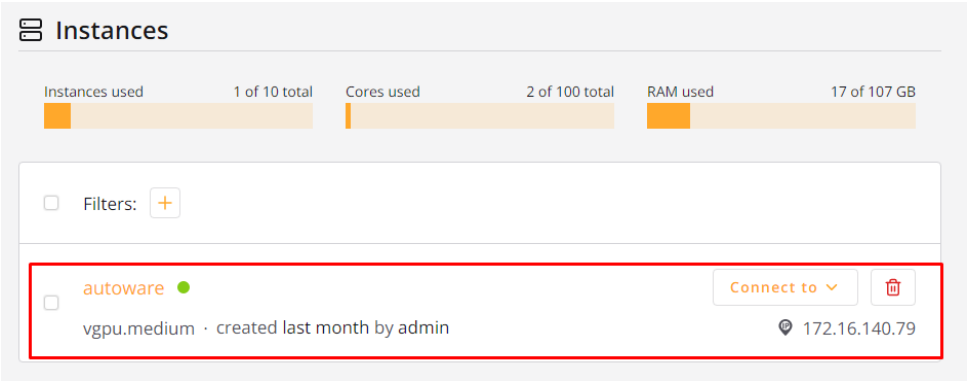


Fig. 4 Shows the instance created by admin (the professor), the IP address it was created on and that it's called autoware.

Once we were able to enter the selected instance, we were able to see a lot of information pertaining to the connected server. In the interactions tab, we clicked on the graphical desktop so we could obtain more information on this instance. All of this is so we were able to understand the program better and be able to get more information on its vulnerabilities.

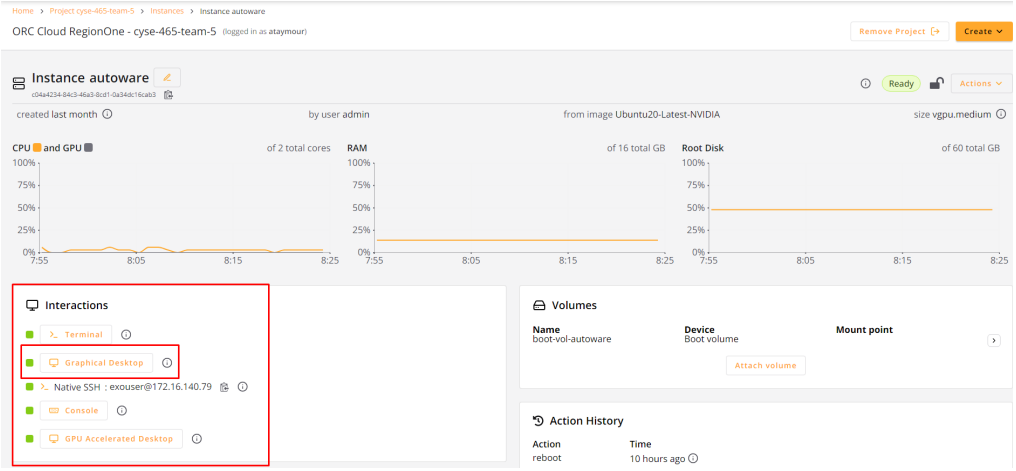
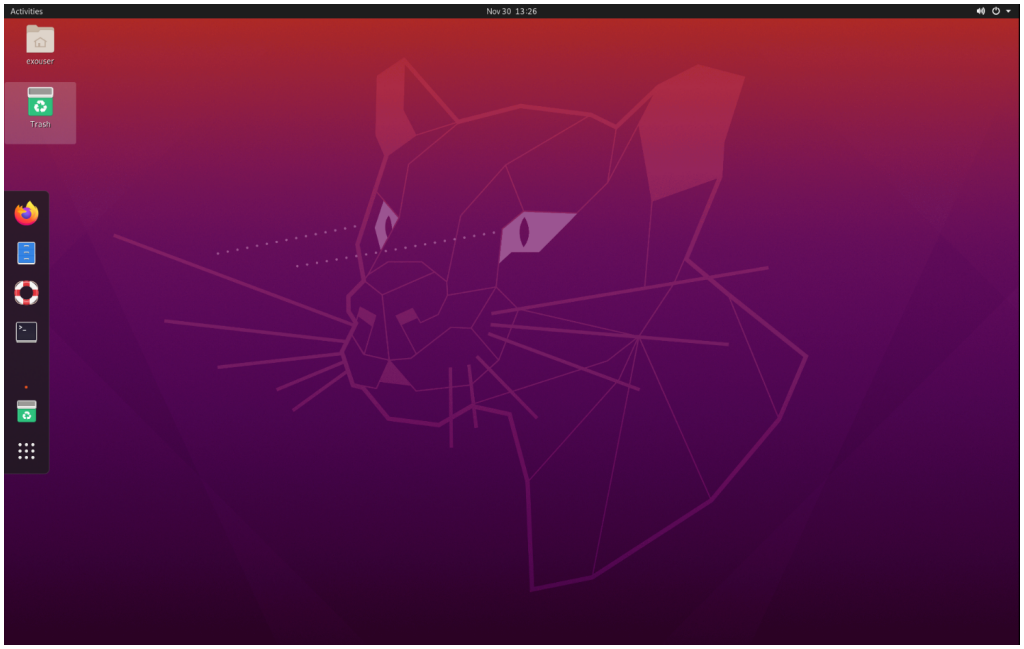


Fig. 5 Shows the interactions tab where we have many ways to connect to the instance.

After we clicked on the Graphical Desktop, we were connected to the remote server.



First, we set the 3D visual sensor by following the directions given by the professor, which we explained in the project before. We can now start with the simulation.

We had root access to the device using ADA, and we made sure it was ran within the same version of our virtual machine. We used the pre-recorded sensor data that was given to us from the last project. We downloaded the PCAP file from [Dual VLP-16 Hi-Res pcap file] provided by the professor and moved it to the /adehome directory.

```
exouser@autoware:~/adehome$ ls -l
total 890056
drwxrwxr-x 9 exouser exouser      4096 Nov  2 08:07 AutowareAuto
drwxrwxr-x 2 exouser exouser      4096 Nov  2 07:03 data
-rw-rw-r-- 1 exouser exouser 911395932 Nov 27 23:39 route_small_loop_rw-127.0.0.1.pcap
exouser@autoware:~/adehome$
```

Fig. 7 shows the pcap file in the adehome directory. This pcap file is what contains the information of the simulation. We inputted this pcap file in this directory because this is where the running 3D perception stack program for the autonomous vehicle is.

From here, we began the ade command. We went into the /adehome/AutowareAuto directory and ran the following,

```
ade --rc .aderc start --update --enter
```

```
exouser@autoware:~/adehome$ cd AutowareAuto/
exouser@autoware:~/adehome/AutowareAuto$ ade --rc .aderc start --update --enter
master: Pulling from autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy
Digest: sha256:845534e377306b9a35893bc1487ea33e64a50ed8e14f7d92104d8e71441e933e
Status: Image is up to date for registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy:master
registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy:master
master: Pulling from autowarefoundation/autoware.auto/autowareauto/amd64/binary-foxy
Digest: sha256:3a75d93538d418d43616db807589b6e3c8c177deec4527222a367deb60b9f607
Status: Image is up to date for registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/binary-foxy:master
registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/binary-foxy:master
Starting ade with the following images:
ade-foxy    | 1641aaf554d9 | master | registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy:master
binary-foxy | b5fb77ce958d | master | registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/binary-foxy:master
ade_registry.gitlab.com_autowarefoundation_autoware.auto_autowareauto_amd64_binary-foxy_master
non-network local connections being added to access control list

Current default time zone: 'Etc/UTC'
Local time is now:      Wed Nov 30 13:53:55 UTC 2022.
Universal Time is now:  Wed Nov 30 13:53:55 UTC 2022.

Adding user exouser to group video
Adding user exouser to group dialout
ADE startup completed.
Entering ade with following images:
ade-foxy    | 1641aaf554d9 | master | registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy:master
binary-foxy | b5fb77ce958d | master | registry.gitlab.com/autowarefoundation/autoware.auto/autowareauto/amd64/binary-foxy:master
exouser@ade:~$
```

This started ADE service and has it ready for future use. The ADE command created an "environment" that ensures it is consistent, which it's important for this type of program since it is a simulation of the real world. Before replaying our sensor data, let's understand what the tags do. The tag `-rc` specifies a different configuration file. In our case, we specified the file `.aderc`. Next, we tell ade to start its service by using `start`. Then, we added the `-update` flag in order to make sure we are using the latest updated version. Finally, we add the `-enter` flag to let the system know that we want to enter ade after starting.

From here we wanted to replay the attack on a loop, for that we utilized the command

```
udpreplay -r -1 route_small_loop_rw-127.0.0.1.pcap
```

udpreplay is used to replay the attack on a loop, which is why we used -r, we play it once -1, and we utilized the same pcap file since it is what contains the virtual environment we wanted to attack.

We opened a new terminal, where we go into ade. We inputted a source that is the file /opt/AutowareAuto/setup.bash in order to help us run the front lidar sensor. In order to do this we used the command

```
Ros2 runvelodyne_nodes velodyne_cloud_node_exe -ros-args -p model:=vlp16 -remap __ns:=/lidar_front -params-file /opt/AutowareAuto/share/velodyne_nodes/param/vlp16_test.param.yaml
```

Ros2 is runvelodyne_nodes starts the cpp node, the model of the lidar sensor is vlp16, the "params-file.." is so the program can file our parameters on the file /opt/AutowareAuto/share/velodyne_nodes/param/vlp16_test.param.yaml

We did the same process but instead of it being for the front lidar sensor, we did it for the rear lidar sensor. In a new terminal window we went into ade and after that ran a very similar command.

```
Ros2 runvelodyne_nodes velodyne_cloud_node_exe –ros-args -p
model:=vlp16 -remap __ns:=/lidar_rear -params-file
/opt/AutowareAuto/share/velodyne_nodes/param/vlp16_test.param.ya
ml
```

And as you can see where it says “remap …” it says lidar_rear instead of lidar_front.

Now, in a new terminal, we entered ade, inputting the setup.bash source, we ran the following command.

```
Ros2 run robot_state_publisher robot_state_publisher
/opt/AutowareAuto/share/lexus_rx_450h_description/urdf/lexus_rx_
450h_pcap.urdf
```

This command was set to determine the sensor's positions in respect to the vehicle.

The visualization was then launched. We entered ade, inputted the setup.bash source and used the command.

```
Rviz2 -d
/opt/AutowareAuto/share/autoware_auto_examples/rviz2/autoware_pe
rception_stack.rviz
```

The rviz2 -d command contains the simulation with the graphical tools, this opened in a new terminal window. Once this simulation opened, we were able to see it lacked calibration data, because it lacked perception nodes. In order to fix this we opened a new terminal, entered ade, inputted the setup.bash source and used the command.

```
Ros2 run point_cloud_filter_transform_nodes
point_cloud_filter_transform_node_exe --ros-args --remap
__ns:=/lidar_front --params-file
/opt/AutowareAuto/share/point_cloud_filter_transform_nodes/param
/vlp16_sim_lexus_filter_transform.param.yaml --remap
__node:=filter_transform_vlp16_front --remap
points_in:=/lidar_front/points_xyzi
```

Ros2 run point_cloud_filter… this node transforms the output of the velodyne_node into a common frame. This means that it makes it easier for the program to reuse the data, we needed to remap the sensors for the lidar_front and the parameters are going to come from /opt/AutowareAuto/share/point_cloud_filter_transform_nodes/param/vlp16_sim_lexus_filter_transform.param.yaml

Then we went back to the simulation and saw that it was running, but not within the right parameters because it was not aware where the ground was. We opened a new terminal, opened ade, inputted the setup.bash source and ran the command.

```
Ros2 run ray_ground_classifier_nodes ray_ground_classifier_cloud
node_exe --ros-args --params-file
/opt/AutowareAuto/share/point_cloud_filter_transform_nodes/param
/vlp16_sim_lexus_filter_transform.param.yaml --remap
points_in:=/lidar_front/points_filtered
```

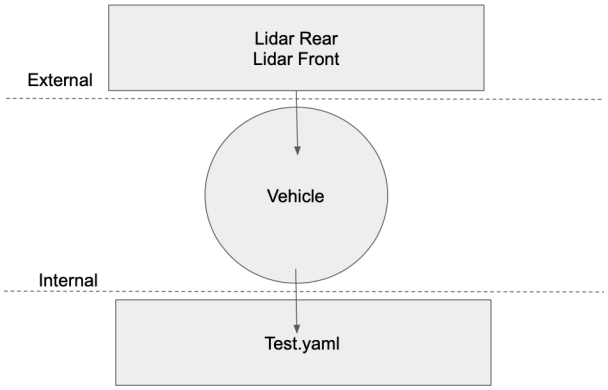
This node ros2 run ray_ground_classifier… indicates whether an object belongs on the ground or not. The parameters are within the same file as the last command we executed since we are running it in the same virtual environment and we are just adding this node into it. And we are adding this sensor into the lidar front. We were able to see that more colors were added into our environment, green objects are classified as ground and white is classified as non-ground. The car will avoid everything white and will follow the green.

The Attack:

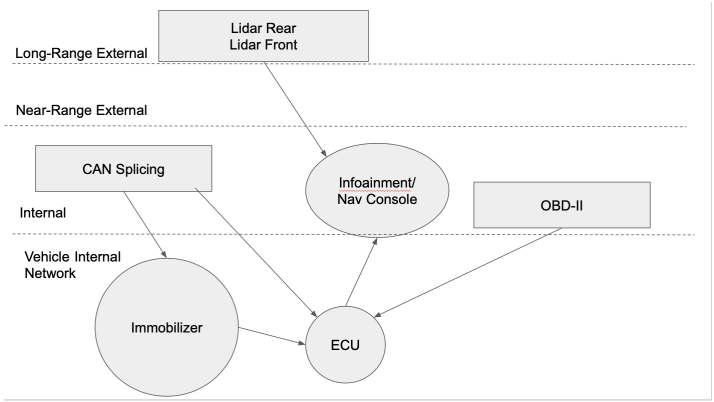
Threat Model

In order to create this threat model we had to be able to find our attack surfaces. Since this is a virtual environment we do not have components such as key fobs, audio inputs, if the car is electric, etc. The main source of vulnerabilities that we possess are the Lidar sensors, these are the “eyes” of the car when it comes to navigating and if an attacker messes with the sensors it could mean really bad news.

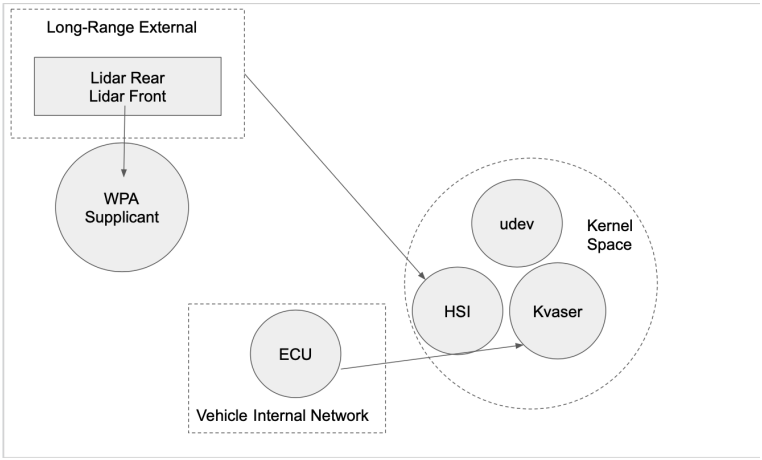
Level 0: Bird’s-Eye View



Level 1: Receivers



Level 2: Receiver Breakdown

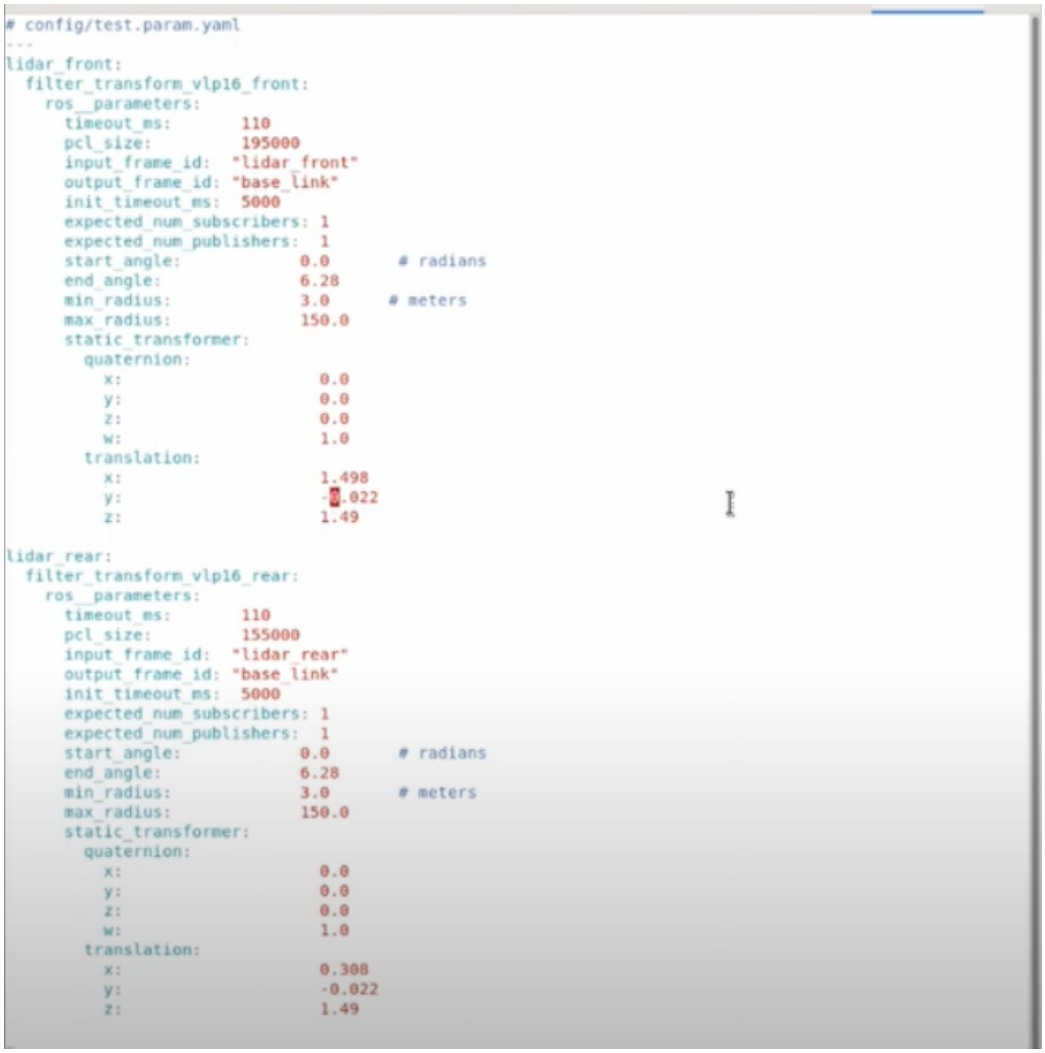


Attack Implementation

For our Attack we went inside the file “test.yaml” that contained the sensor details, this was a specific file we made for the attack. This file contained the lidar based perceptions which is what helps the vehicle calculate the distances from obstacles so it avoids a collision. It helps map its surroundings. This file is a node file, param.yaml which contains the front and rear lidar sensor. In order to access this file and perform the attack we did the same process as before, initiated and inputted the setup.bash source so then we could run the command.

```
Ros2 run point_cloud_filter_transform_nodes
point_cloud_filter_transform_node_exe --ros-args --remap
__ns:=/lidar_front --params-file test.yaml --remap
__node:=filter_transform_vlp16_front --remap
points_in:=/lidar_front/points_xyzi
```

Which as we can see no longer is redirecting us to the parameters of the /opt/AutowareAuto… file, now it is showing the attack file. When we ran it, nothing changed, this is because we hadn't done anything malicious to our file, yet.



```
# config/test.param.yaml
---
lidar_front:
  filter_transform_vlp16_front:
    ros__parameters:
      timeout_ms:       110
      pcl_size:         195000
      input_frame_id:  "lidar_front"
      output_frame_id: "base_link"
      init_timeout_ms:  5000
      expected_num_subscribers: 1
      expected_num_publishers:  1
      start_angle:            0.0       # radians
      end_angle:              6.28
      min_radius:             3.0      # meters
      max_radius:             150.0
      static_transformer:
        quaternion:
          x:                   0.0
          y:                   0.0
          z:                   0.0
          w:                   1.0
        translation:
          x:                   1.498
          y:                   -0.022
          z:                   1.49

lidar_rear:
  filter_transform_vlp16_rear:
    ros__parameters:
      timeout_ms:       110
      pcl_size:         155000
      input_frame_id:  "lidar_rear"
      output_frame_id: "base_link"
      init_timeout_ms:  5000
      expected_num_subscribers: 1
      expected_num_publishers:  1
      start_angle:            0.0       # radians
      end_angle:              6.28
      min_radius:             3.0       # meters
      max_radius:             150.0
      static_transformer:
        quaternion:
          x:                   0.0
          y:                   0.0
          z:                   0.0
          w:                   1.0
        translation:
          x:                   0.308
          y:                   -0.022
          z:                   1.49
```

In the front sensor, we changed the y-axis of the translation to -10. Then we ran the simulation again. We can see that the car is now running in a different lane, the left lane, which is now in an area where buildings and other objects are going to be on the way, activating the sensors of the vehicle. In response the vehicle is going to swerve to the right in order to avoid said objects, but in actuality it is now further right than it thinks it is. What does that mean? We ran the command again.

```
Ros2 run point_cloud_filter_transform_nodes
point_cloud_filter_transform_node_exe --ros-args --remap
__ns:=/lidar_front --params-file
/opt/AutowareAuto/share/point_cloud_filter_transform_nodes/param
/vlp16_sim_lexus_filter_transform.param.yaml --remap
```

```
__node:=filter_transform_vlp16_front --remap
points_in:=/lidar_front/points_xyzi
```

This lets us see our vehicle within the simulation that has not been tampered with, and we were able to see that the vehicle was driving too far to the right, and it was running into incoming traffic.

Strength of the Attack

The attack's strengths are that it will go undetected by the syntax, uses legitimate data with minor adjustments, and can be simulated before the real attack. The affected file contains the same data and syntax. This means that we don't need to have a buffer overflow attack or an over engineered attack. Over engineered attacks require a lot of work and time to get it right. In addition, the changed file can be simulated and tested in a lab condition.

Weakness of the Attack

Our attack required some assumptions before execution. The attack requires admin privileges which can be achieved in other ways. This is a weakness because entering a system will be logged and can be thwarted with access control. In addition, our changed file is written in plaintext which is a problem because systems can detect a change. A simple hashing algorithm can be used to detect any changes to a system file. If the system built in file got changed, the hash value would be different, and thus, error codes will pop up on the system console informing it of such a change.

Task 3. Create a 15 minute video demonstration of your attack and submit the video. Make sure your video documents all the details in the demo (i.e., share the terminal on how you are launching the attack and explain the details).

https://www.youtube.com/watch?v=0DITqNjOwQg

Individual effort:

Abdulla Taymour: Worked on the attack and video recording.

Ruth Delgado: Worked with Abdulla to write the paper and introduction.

Benjamin Thong: Communication with team and scheduling. Helped Ali with his proposal.

Ali Nadhaif: Worked on possible solutions to defend from the attack.

Github repos:

Abdulla: https://github.com/ataymourgmu/cyse465-final-fall2022

Ruth: https://github.com/rdelgad1998/GMU-CYSE-465-

Ali: https://github.com/AliNadhaif/Final-Group-Project-for-CYSE-465

Benjamin: https://github.com/bthongmason/CYSE-465-Final-Group-Project