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

Final Project

TODAY'S AGENDA:

FIRST:

EVERYONE CREATE A GITHUB WITH THEIR MASON EMAIL NOW, then create a REPO. Write this IN YOUR README FILE (if you'd like, you can write whatever you want.)
Final Group Project for CYSE 465

This Project is about creating a smart vehicle 3D perception demo and proposing a security problem.

We created and simulated this demo using the Autoware system that was provided by our instructor. Once we became familiar with the "environment" and understood the details and intricacies on how this system works, how many sensors it has, what they do and how they work together we "played around" with different attacks to the perception stack.

SECOND:

WE NEED TO DIVIDE THE PROJECT.

Abdulla and I already developed an attack and performed it. In my opinion (and it is open for debate of course) Abdulla can do Task 3 where he demonstrates the attack, making sure he gets all of the details on it. I'll do task 2 (with Abdulla's help) which is the report on the details of the attack. Now I am aware this leaves Ali and Ben with task 1.2 which is optional and 1 part only. I do have a plan and an explanation.

So option one someone can find several defense techniques and the other can try to implement them.

Option 2 is you all collaborate and try different things. So find different defense techniques and implement them on your own, until you find something that works.

THIRD:

This is up to whomever I was thinking one person could be in charge of making sure that everyone's Github's are the same.

What does that mean?

Everyone has the defense code in their repo.

Everyone has the details of the attack.

Everyone has the video.

Everyone has the attack code.

FINAL:

We should all be reading everyone's parts, making sure we are checking on each other, not repeating the information 2 or 3 times or even contradicting each other.

If everything is smooth and we do not need to communicate I say we should meet one last time. Just to make sure we all know what we all did. What was the attack? What was the defense? etc.

THIS IS WHERE THE ACTUAL PAPER STARTS

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

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 will be based on Autoware running on Ubuntu20-Latest-NIVIDIA provided by George Mason University that is 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 login 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 it is 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 comes in handy because the simulation needs a lot of space and power to run. Going back to the program, after we login, we are able to see all of the projects we are working on at the moment.

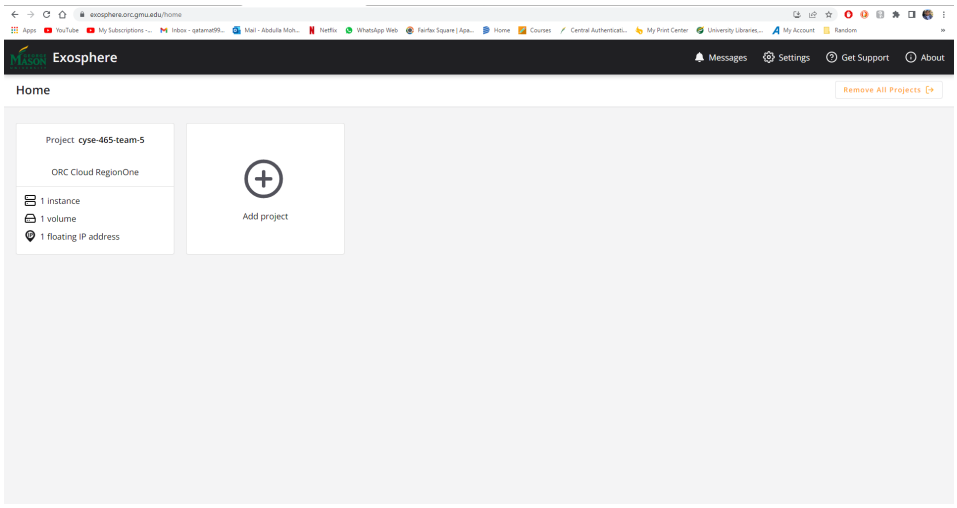


Fig. 1 Shows a screenshot of the Exosphere environment and the project created.

The course instructor has already created the project and instance. Then, we enter the project and view our instances.

As we can see the professor has already created the project and instance for us, since he had to input the code. We are going to enter the project and view the instances created so we can work on it.

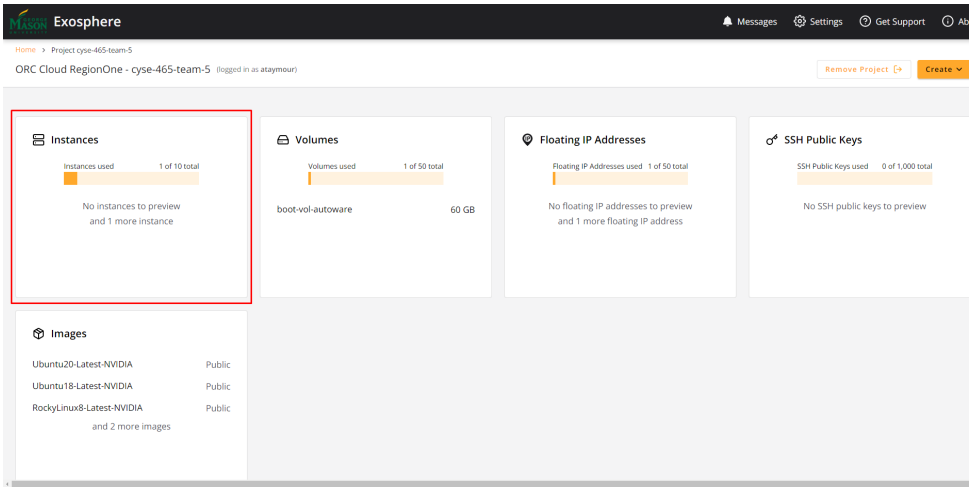


Fig. 2 Shows the Instances within the program. We can see there is 1 available as well as 1 volume and 1 Floating IP Address.

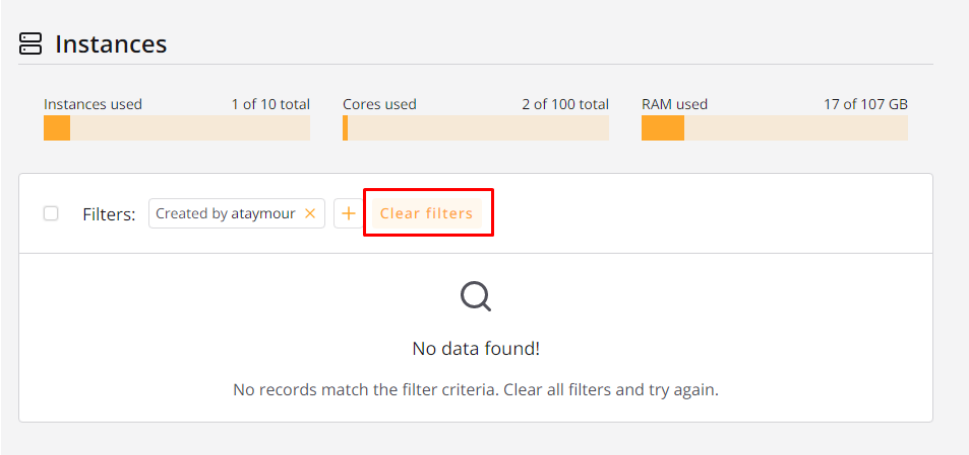


Fig. 3 Shows

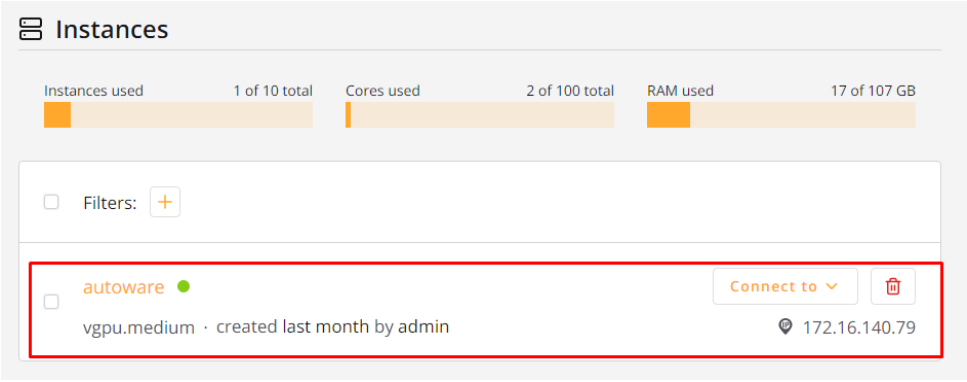


Fig. 4 Shows the instance created by admin (the professor), the IP address it was created on and that it's called autoware.

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

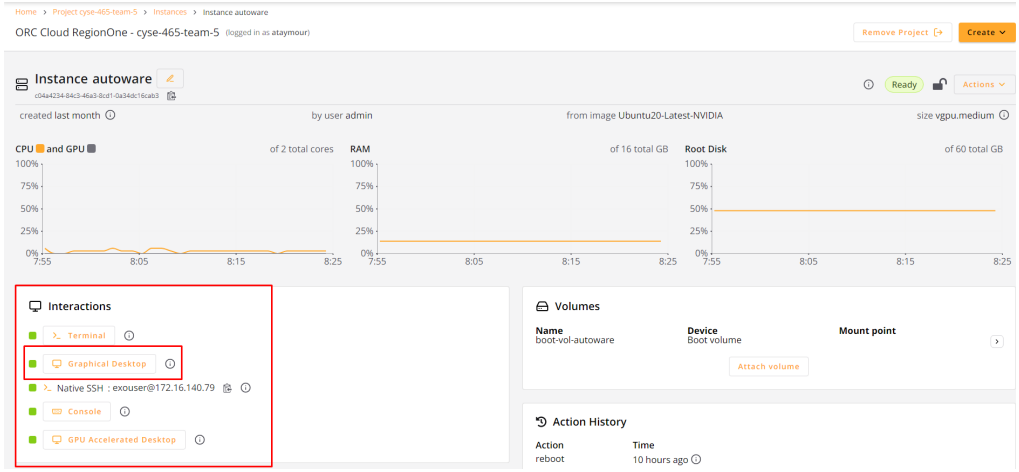
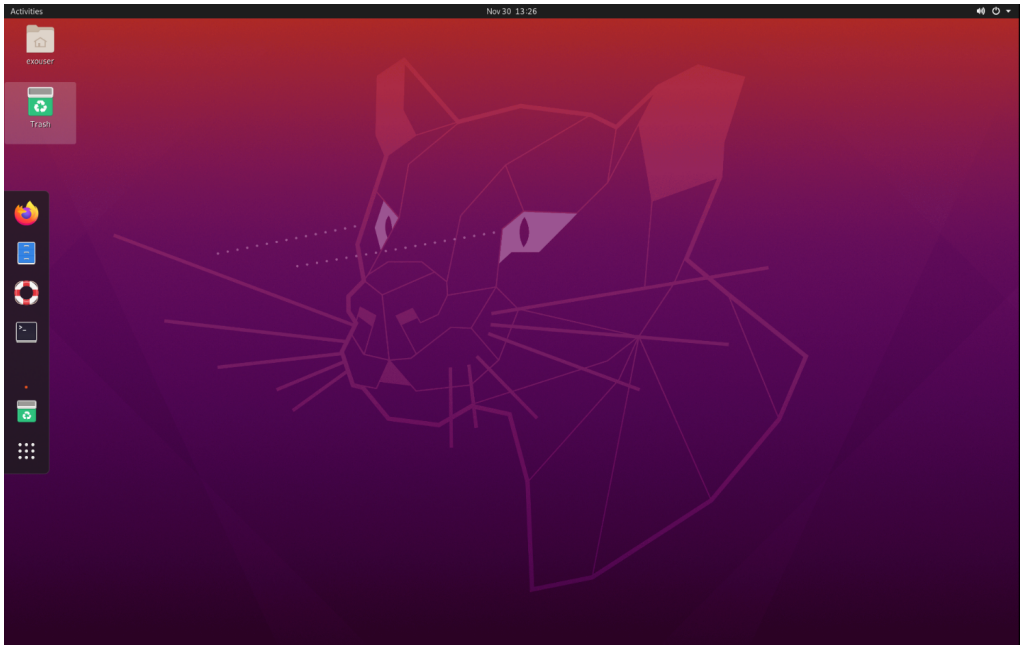


Fig. 5 Shows the interactions tab.

After we click on the Graphical Desktop, we are 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 will be using pre-recorded sensor data in this 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. We inputted this pcap file in this directory because this is where the running 33D perception stack program for the autonomous vehicle is.

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

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

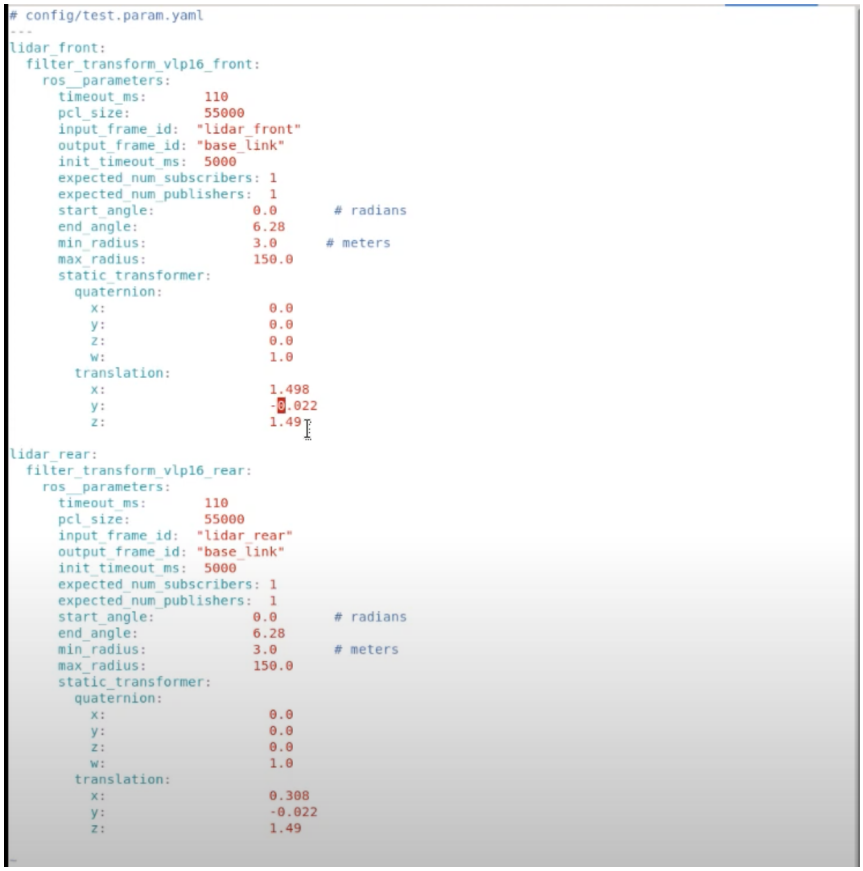
```
exouser@autoware:~/adehome$ cd AutowareAuto/
exouser@autoware:~/adehome/AutowareAuto$ ade --rc .aderc start --update --enter
master: Pulling from autowarefoundation/autoware.auto/autowareauto/amd64/ade-foxy
Digest: sha256:845534e377306b9a35893bc1487ea33e64a50ed8e14f7d92104d8e71441e933e
Status: Image 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 will start ADE service and have it ready for future use. The ADE command creates 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 add 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.

The Attack:

For our Attack we went inside the code that contains the sensor details. This file contains 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. We



```
# config/test.param.yaml
---
lidar_front:
  filter_transform_vlp16_front:
    ros__parameters:
      timeout_ms:       110
      pcl_size:         55000
      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:         55000
      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
```

Fig. 8 Shows the node file with the front and rear lidar sensor. It contains parameters such as the ros_ parameters, which are the velodyne_node, the robot_state_publisher, point_cloud_filter_transport_nodes, ray_ground_classifier_nodes and euclidean_cluster_nodes; all thoroughly explained in the last project. It also contains the timeout ms which is .The pcl_size which is the output point clouds parameter. The quaternion

Threat Model

Attack Implementation

Strength of the Attack

Weakness of the Attack

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