**Cybersecurity for AI**

Use case on Evasion in a distributed IoT system

Sergi Pla

Jordi Otal

Arnau Garcia

Enric Pedrico

**Task 1: System Design and Security Analysis**

**[Q1] Decide a placement (edge node or centralized controller) for each of the**

**processes involved in the workflow: moving object detection, object classification,**

**rule-based policy. Assume that image processing must be done in the edge node.**

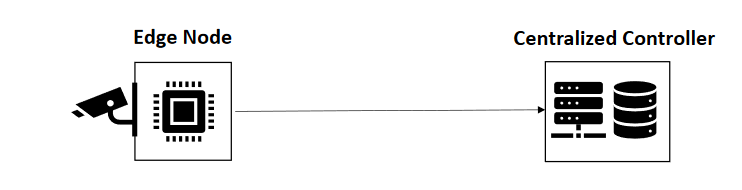
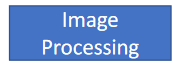
**Justify your choice and analyse the strengths and weaknesses from the point of view**

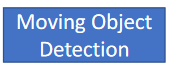
**of security, as well as comment on the requirements and limitations of your approach**

**in terms of performance. Indicate which type of data is going to be transmitted**

**between edge node and centralized controller, commenting on its characteristics and**

**how frequent is going to be transmitted.**

****



**Image processing** → Edge node

* **Justification**: We process images locally to reduce data transmission.
* **Pros**: Only the needed data is sent to the centralized controller, reducing network usage.
* **Cons**: Higher computational load on the edge node, potential vulnerabilities if compromised.

**Moving object detection** → Edge Node

* **Justification**: Needs to be close to the camera for real-time execution.
* **Pros**: Fast detection, avoids transmitting full video and just send the information to analyze.
* **Cons**: Requires robust security for the edge node.

**Object Classification** → Centralized controller

* **Justification**: Computationally demanding and better handled centrally. We can have better and more powerful equipment and hardware here than on the node.
* **Pros**: Allows the use of more complex models and centralized updates.
* **Cons**: Adds latency due to data transmission, this transmission can as well be vulnerable to Man in the Middle attacks.

**Rule-Based Policy** → Centralized controller

* **Justification**: Centralized decision-making ensures consistent actions across the system.
* **Pros**: Better coordination and security oversight.
* **Cons**: Creates dependency on the central controller which might fail or turn into a bottleneck.

#### **Overall security and performance:**

* **Strengths**:
  + Processing at the edge node minimizes transmitted sensitive data.
  + Centralized classification ensures better supervision and consistency.
* **Weaknesses**:
  + Vulnerability during data transmission between edge nodes and the controller.
  + Edge node compromise could disrupt initial image processing or detection.

**Task 2: System implementation**

**[Q2] Prepare a code to pre-process the data and train a DNN classifier that distinguishes the 10 different labels that you can find in CIFAR10 dataset according to**

**the classes of our model:**

| **Model class** | **CIFAR10 Label** |
| --- | --- |
| Authorised vehicle | ‘airplane’, ‘automobile’ |
| Non-authorised vehicle | ‘ship’, ‘truck’ |
| Animal | ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’ |

**Explain the model structure (type of model, number of layers and characteristics), and show performance (accuracy) results of the trained model. Once you have it ready, save it model.**

We directly translated the labels before train the model. We’ve done this in order to increment the accuracy of the model, as there are less labels.

Type of model: It’s a CNN (convolutional neural network).

Number of layers: There’s 9 layers (the 8 explicit and the input one)

Characteristics: It takes the input, activates 2 layers with ReLU, and then it does a final layer with 32 neurons, finishing with an output layer (transforming it to our classes).

Performance: It has a mean accuracy per image of 94.6%. (100 executions)

Results: The model predicts correctly 91 images. (out of 100 executions)

The performance is how much insurance is the model, and the 91 images correctly is how many images the model predicted correctly.

**[Q3] Implement a function called “edge\_process” that will read random samples from CIFAR10 datasets and return them. Then, implement a function called “controller\_process” that receives the output sample from “edge\_process” and performs object classification and rule-based policy actions. The latter must contain and make use of the model trained in [Q2]. Verify that the process runs properly and that the actions are in line with the performance of the trained model.**

The function works correctly. (included in functions.py file). To execute them and run the rest of the practice, go to the main.py file.

**Task 3: Man-In-The-Middle (MiTM) Attack**

**[Q4] Explain (without code) what is the method you used and how it works. Then, prepare a new function called “MiTM\_process” that receives a normal sample and returns a sample with adversary noise. Evaluate the function for different configurations of the parameters that you identify. Show how classification is deceived as a function of the amount of noise added. Decide the best configuration of the hyperparameters of your method so that you can get the target malfunctioning generating images that are not easy to detect as altered. Plot some images before and after adversary noise addition in order to visually evaluate its impact**

The method uses the gradient of the image and the ‘non-auth’ label to try to trick the real model in a way that it thinks that it isn’t a ‘non-auth’ vehicle.

We’ve done the experimentation only having non-auth vehicles, in order to reduce the random factor.

Multiplier = 0: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

errors:

1, 5%

This is our model.

Multiplier = 0.005: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1]

errors:

8, 40%

With 0.005, it is very difficult to notice that the image has noise.

Multiplier = 0.01: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[2, 0, 2, 1, 0, 0, 2, 0, 2, 0, 1, 1, 1, 1, 2, 1, 2, 0, 2, 1]

errors:

13, 65%

With 0.01, it is also very difficult to notice that the image has noise.

Multiplier = 0.05: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 2, 2, 0, 0]

errors:

20

With 0.05, it is clear that there is noise, but the error rate is very high.

We also have tried with more values above 0.1, but in all of them you can notice that there’s noise. So, if you want to maximize the error rate, the best one is 0.05. If you want to be undetectable by human eyes, 0.01 is the best one.

**[Q5] Analyse critically (pros, cons, requirements, assumptions, side attacks that are required) the three options that the attacker has in order to place the “MiTM\_process” function:**

**1) In the edge node, i.e., the sample leaves the edge node with the adversary noise already added.**

* **Process Flow:** The adversarial noise is added directly to the sample at the edge node before it is transmitted.
* **Advantages:**
  + Simplifies the attack because the adversary needs access to just one node.
  + The attacker doesn’t need to intercept data in transit, reducing the complexity of the attack.
* **Disadvantages:**
  + Requires physical or software compromise of the edge node, which may be difficult if the edge devices are secured.
  + Detection mechanisms (e.g., monitoring edge node activity) can mitigate such attacks.
* **Requirements/Assumptions**:
  + The attacker must have full control over the edge node.
* **Side Attacks:**
  + Disabling logs or alerts on the compromised node to avoid detection.

**2) In an intermediate/remote node/site, i.e., the attacker intercepts the packet traffic containing the image while in transit, perform the noise addition, injects the modified image to the packets, and forward them to the controller.**

* **Process Flow:** The attacker intercepts the image data packets in transit, adds adversarial noise, and re-injects the altered packets to the controller.
* **Advantages:**
  + Does not require access to edge or centralized nodes, only the network.
  + Harder to detect the packet modification.
* **Disadvantages:**
  + Requires access to the network infrastructure, which may be secured by encryption.
  + Real-time modifications are computationally demanding and may introduce some delay, which can signal the presence of the attack, thus being detected.
* **Requirements/Assumptions:**
  + The attacker needs tools to intercept and modify encrypted network traffic.
  + The system may need to lack encryption or use weak encryption protocols.
* **Side Attacks:**
  + Network sniffing tools (like Wireshark) and spoofing attacks to intercept traffic.
  + Attacks on encryption, since traffic on the net may be encrypted.

**3) In the controller, i.e., the received normal samples are altered before running the classifier**

* **Process Flow:** The adversarial noise is added after the sample reaches the controller, before classification.
* **Advantages:**
  + The attacker gains direct access to the controller’s internal processes, potentially enabling additional manipulations beyond just adversarial noise, so it grants more control than other options.
* **Disadvantages:**
  + Requires a full compromise of the controller, which is typically well-secured.
  + Detection is likely, as the controller usually has extensive monitoring and logging.
* **Requirements/Assumptions:**
  + High-level access to the controller’s system, potentially requiring sophisticated attacks or the need of credentials obtained by social engineering or other means.
* **Side Attacks:**
  + Disabling security measures on the controller.
  + Using social engineering to gain access credentials.

**[Q6] Modify the “edge\_process” code in order to randomly select pictures of authorized vehicles with a probability equal to p and pictures of non-authorized vehicles and animals with probability 1-p. Comment “MiTM\_process” line to check that the script works well and classifier still behaves properly. Then, uncomment “MiTM\_process” and run the script for different values of p. Analyse the obtained results in the way you think more interesting. Among the different cases, consider a large value of p (i.e., most of the images are authorized vehicles). Guess if the obtained results can lead to find a way to detect that a MiTM attack is happening.**

The script works correctly. For all experiments, we will have 0.05 multiplied by the perturbations. Auth = 0, non-auth = 2, animals = 1

For p = 0.1, this are the real labels vs predicted labels:

real=[1, 1, 0, 1, 1, 1, 1, 1, 1, 1]

pred=[2, 2, 0, 0, 2, 0, 0, 0, 0, 1]

errors:

8

For p = 0.25:

real=[0, 0, 0, 0, 1, 1, 1, 0, 0, 0]

pred=[0, 0, 0, 0, 2, 0, 2, 0, 2, 0]

errors:

4

For p = 0.75

real=[0, 0, 1, 0, 1, 0, 0, 0, 0, 0]

pred=[0, 0, 2, 2, 2, 0, 0, 0, 0, 0]

errors:

3

It makes sense that the number of errors decreases as the p increases, because we programmed the adversary function to work as a dodger of non-auth images. Also note that only in one case it classifies as a non-auth vehicle, so the adversary noise tricks it well.

If we obtain a lot of errors with a low p, and less errors with a larger p, we can deduce that we are being attacked.

**Task 4: Detecting or mitigating the effects of a MiTM attack**

**[Q7] Define ONE method that could work to detect or mitigate the effect of a MiTM attack. Redo the drawing of task 2 with the addition of the new process/es. Implement it/them as part of edge\_process and/or controller\_process functions. Evaluate its performance by running them apart and/or included in the main script. Report all the explanation and methodology, results and findings/conclusions, jointly with the code.**

We decided to use **Adversarial training,** which incorporates adversarially perturbed samples during the training phase of the model. By doing this the model learns to classify correctly even when adversarial noise is added.

**How It Works**:

* Generate adversarial samples during training.
* Add these samples to the training dataset with their correct labels, generating n models, each one with the original samples and the samples with k noise.
* Retrain the model to handle both clean and adversarial inputs.
* The image then goes through all n models, and then we keep the most common response. (After validating that it works well).

**Advantages**:

* Directly enhances model robustness to adversarial attacks.
* Does not require changes to the architecture or workflow. Only the training changes.

**Disadvantages**:

* Training becomes computationally expensive due to the larger and more complex dataset.
* Adversarial training may not generalize well to unseen types of perturbations or new attack methods, however we know what perturbation is used always in this case so it’s a good solution.

**Effectiveness Against MiTM Attacks**:

* Very effective for perturbations similar to those used during training.
* Less effective if attackers use new strategies or higher perturbation magnitudes beyond what the model was trained on.

We executed the process with adversary noise = 0.05 (the one that gave us the most error rate) and 10 samples, all non-auth (they were the problematic ones) [the models are the ones with “cifar10\_model\_N.keras”, respectively]. We obtained this results:

Initial model = 96 error/s out of 100

Model with 0.005 noise = 12 error/s out of 100

Model with 0.01 noise = 33 error/s out of 100

Model with 0.05 noise = 0 error/s out of 100

And if we mix all the results (without having the initial model, because we “noticed” that it doesn’t work well), we obtain:

7 error/s out of 100

7% is not a very low number, but it is definitely better than the first model that we trained. We can also see that if we have trained the model with the same noise as the adversary does, the error rate is almost 0.

As mentioned before the architecture does not change because the difference is on the training phase, so we have the same infrastructure as in Q2:

