Exploiting Vulnerabilities in Intrusion Detection Systems

Sai Prasath, Ankush Meshram, Anne Borcherding, Markus Karch
IIT Bhubaneswar, Fraunhofer IOSB

Overview:

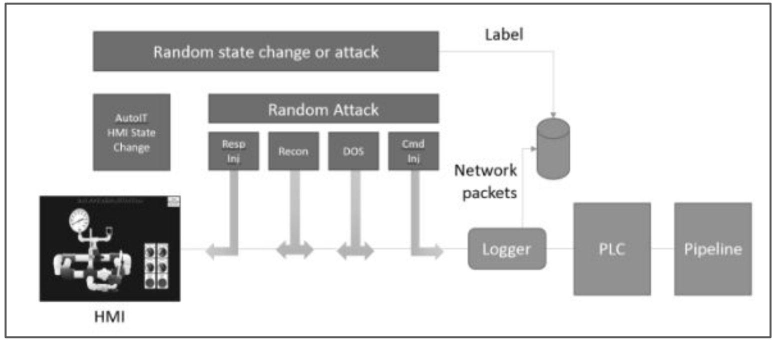
Overview:

Introduction

- In this research work we analyze the vulnerabilities and the shortcomings of various Intrusion Detection Systems(IDS) in Industrial Control Systems(ICS).
- Many works focus on building robust IDS but fail to address their weaknesses.
- We build and test various adversarial algorithms and their defenses on state-of-the-art models.
- We extend the ideas of adversarial attacks from the image domain to cyber security and deal with additional constraints imposed while performing this task.

Overview:

Dataset Analysis:

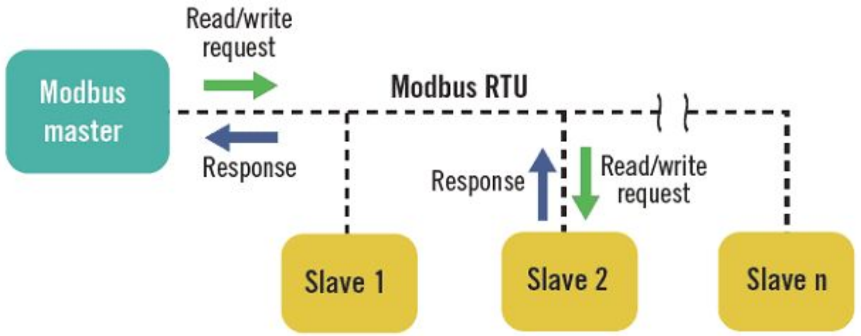


Gas Pipeline Dataset:

- Realistic environment; multiple different randomized attacks
- Tested for inherent weaknesses; records both network and controller data

Dataset Generation

- The ICS uses the **Serial Modbus Protocol** for communication between the master(SCADA) and the slave(controller units).
- Only the master can initiate the conversation and the slave always replies with a response.
- The packet from the master is distributed to all the slaves and contains an Slave ID used to identify the particular slave for which the message was intended for. Other slaves ignore the message.
- These packets between the master and the slave are captured for analysis. **Most of the packets repeat in the sets of 4: Read request, read response, write request, write response.**



Example of a datapoint

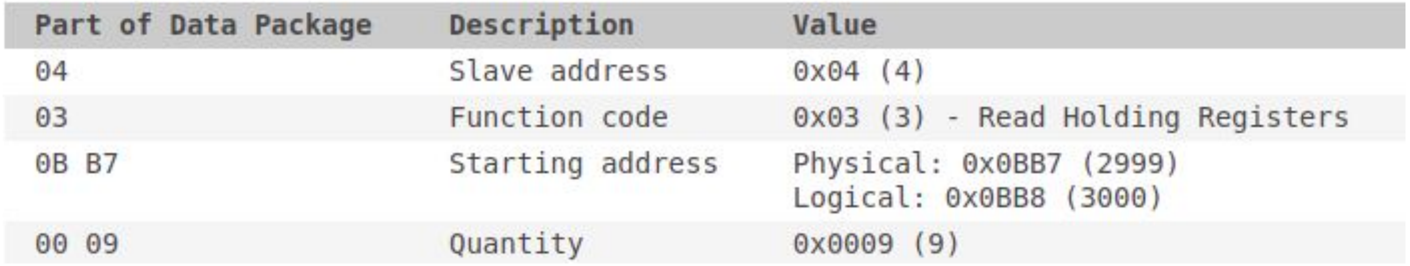
04030bb700093245,0,0,1,3,1418682163.170388

Basic Information:

04030bb700093245: Modbus Packet; 0,0: Attack category and Specific; 3, 1: Source and Destination; 1418682163.170388 : Time stamp

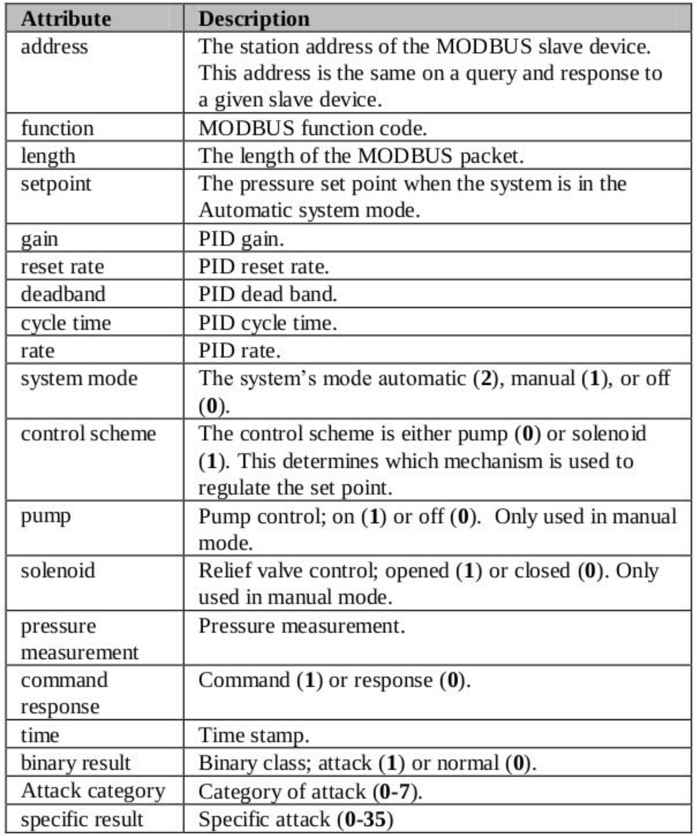
Source and Destination: 1(Master), 2(MITM), 3(Slave)

---> Read Request (contains the slave address, function code to read, starting address of the register and the number of registers to read and the CRC value)



Part of Data Package	Description	Value
04	Slave address	0x04 (4)
03	Function code	0x03 (3) - Read Holding Registers
0B B7	Starting address	Physical: 0x0BB7 (2999) Logical: 0x0BB8 (3000)
00 09	Quantity	0x0009 (9)

Extracting features



Attribute	Description
address	The station address of the MODBUS slave device. This address is the same on a query and response to a given slave device.
function	MODBUS function code.
length	The length of the MODBUS packet.
setpoint	The pressure set point when the system is in the Automatic system mode.
gain	PID gain.
reset rate	PID reset rate.
deadband	PID dead band.
cycle time	PID cycle time.
rate	PID rate.
system mode	The system's mode automatic (**2**), manual (**1**), or off (**0**).
control scheme	The control scheme is either pump (**0**) or solenoid (**1**). This determines which mechanism is used to regulate the set point.
pump	Pump control; on (**1**) or off (**0**). Only used in manual mode.
solenoid	Relief valve control; opened (**1**) or closed (**0**). Only used in manual mode.
pressure measurement	Pressure measurement.
command response	Command (**1**) or response (**0**).
time	Time stamp.
binary result	Binary class; attack (**1**) or normal (**0**).
Attack category	Category of attack (**0-7**).
specific result	Specific attack (**0-35**)

4,3,16,?,?,?,?,?,?,?,?,?,?,?,12869,1,1418682163.1,0,0,0

4,3,46,?,?,?,?,?,?,?,?,?,?,0.689655,12356,0,1418682163.2,0,0,0

4,16,90,10,115,0.2,0.5,1,0,0,1,0,0,?,17219,1,1418682164.9,0,0,0

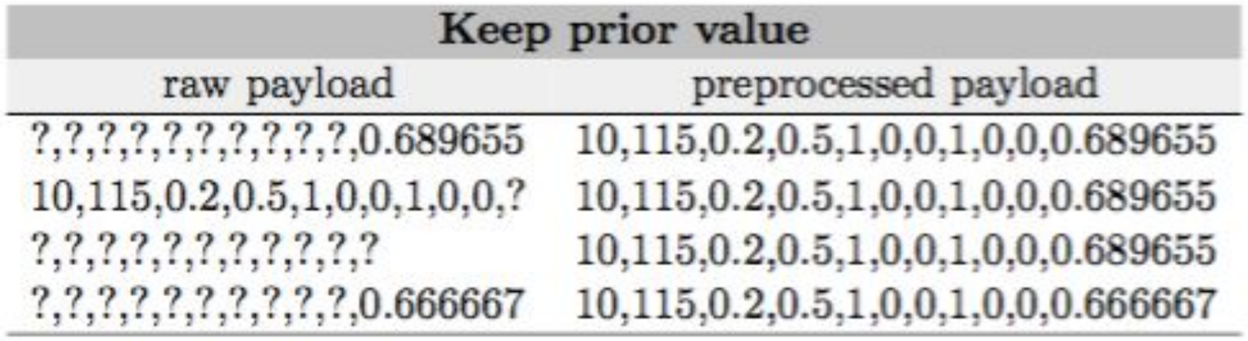
4,16,16,?,?,?,?,?,?,?,?,?,?,?,17718,0,1418682165.1,0,0,0

Network: Address, Length, CRC Rate(check for errors), Command Response, Time

Payload: All the rest

Dataset Preprocessing

- Fill the missing values: Keep prior values
- One-hot encoder: Function Code, System Mode
- 0-1 Scaler: For all continuous features
- Remove biased features: Time and Slave Address



Keep prior value	
raw payload	preprocessed payload
?,?,?,?,?,?,?,?,?,?,0.689655	10,115,0.2,0.5,1,0,0,1,0,0,0.689655
10,115,0.2,0.5,1,0,0,1,0,0,?	10,115,0.2,0.5,1,0,0,1,0,0,0.689655
?,?,?,?,?,?,?,?,?,?,?	10,115,0.2,0.5,1,0,0,1,0,0,0.689655
?,?,?,?,?,?,?,?,?,?,0.666667	10,115,0.2,0.5,1,0,0,1,0,0,0.666667

Overview:

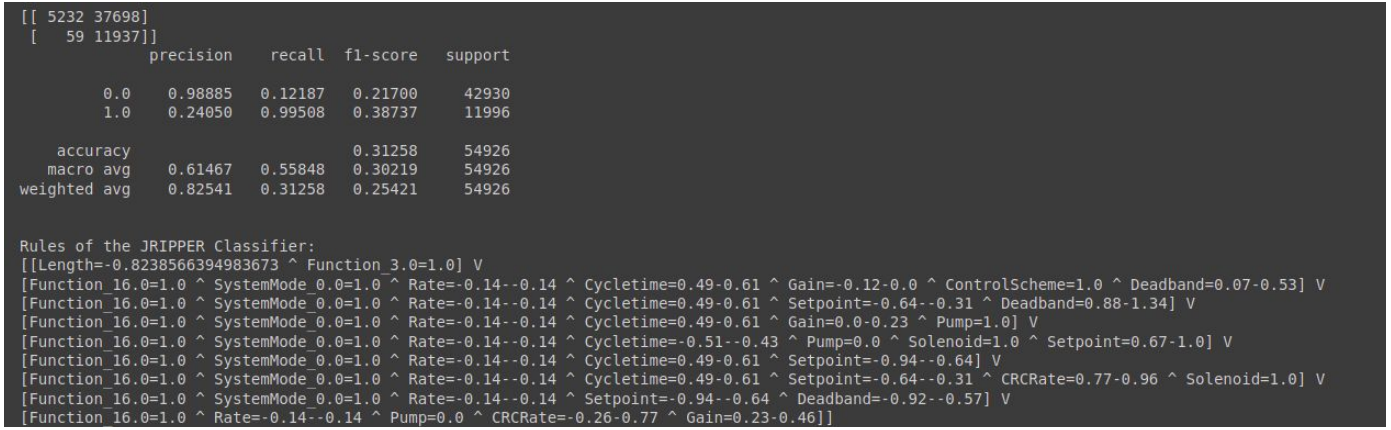
1. Introduction
2. Dataset Analysis
3. Rule Based Models
4. Deep Learning Models (+ Gradient Based Models)
5. Tree Based Models
6. Conclusion

Rule Based Models

Create rules in Disjunctive Normal Form(DNF i.e. Or of And)

Example of a ruleset: [R11 ^ R12 ^ R13 … ^ R1x] OR [R21 ^ R22 ^ R23 .. ^ R2x] OR ….

The model learns the rules for the normal class. If the packet satisfies the rules it's classified as normal else it's anomalous.



```
[[ 5232 37698]
 [   59 11937]]
              precision    recall  f1-score   support

         0.0    0.98885   0.12187   0.21700     42930
         1.0    0.24050   0.99508   0.38737     11996

    accuracy                        0.31258     54926
   macro avg    0.61467   0.55848   0.30219     54926
weighted avg    0.82541   0.31258   0.25421     54926


Rules of the JRIPPER Classifier:
[[Length=-0.8238566394983673 ^ Function_3.0=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Gain=-0.12-0.0 ^ ControlScheme=1.0 ^ Deadband=0.07-0.53] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.64--0.31 ^ Deadband=0.88-1.34] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Gain=0.0-0.23 ^ Pump=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=-0.51--0.43 ^ Pump=0.0 ^ Solenoid=1.0 ^ Setpoint=0.67-1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.94--0.64] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.64--0.31 ^ CRCRate=0.77-0.96 ^ Solenoid=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Setpoint=-0.94--0.64 ^ Deadband=-0.92--0.57] V
[Function_16.0=1.0 ^ Rate=-0.14--0.14 ^ Pump=0.0 ^ CRCRate=-0.26-0.77 ^ Gain=0.23-0.46]]
```

How to attack these models?

Ruleset: All the rules don't use all the features. The features which aren't used in each rule can be used to manipulate the output. **(Considered for NORMAL)**

Ex: [R11 ^ R12 ^ R13 … ^ R1x] OR [R21 ^ R22 ^ R23 .. ^ R2x] OR ….

Let us assume that a packet passes through the IDS and is labelled as Normal which means that one of the above rules were satisfied by the packet.

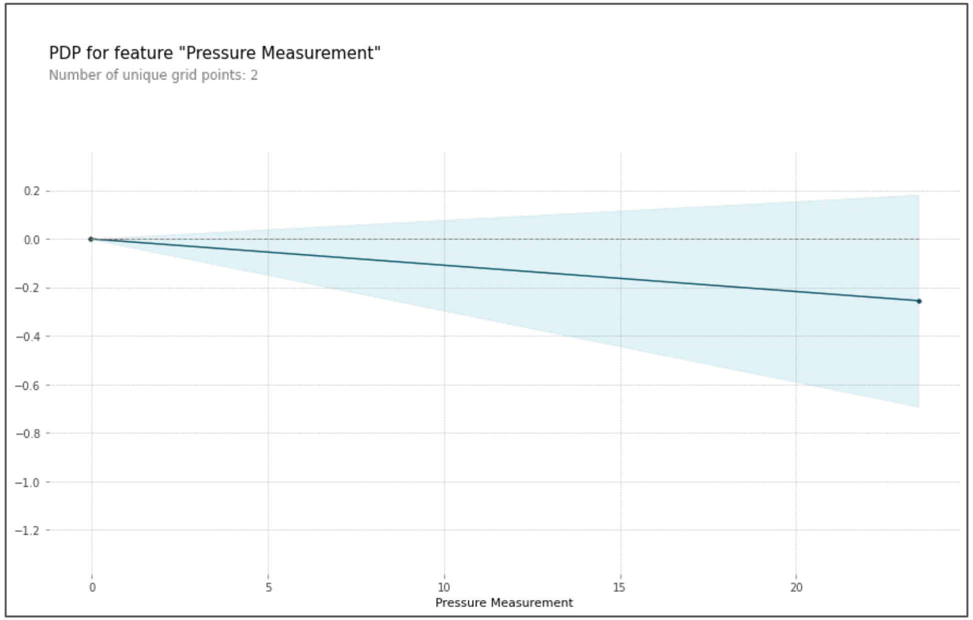
Say the packet satisfied the first rule: R11 ^ R12 … ^ R1x

Let us assume x=5 (Most rule only have 5 to 7 features in them) which means that the other features can be manipulated to be outside the normal zone(extracted from PDP).

To identify the features in the Rule we use the SHAP Values.

Therefore to successfully attack the model we need to know the feature intervals that are being monitored and the features contributing to that decision.

Detecting the intervals: PDP Plots



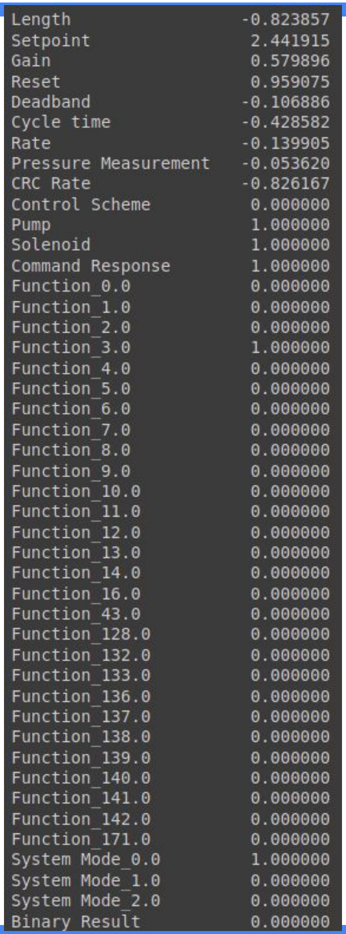
From the PDP plots we can find the intervals that are in consideration.

Ex: Here the Pressure Measurement from PDP plot can be extracted as

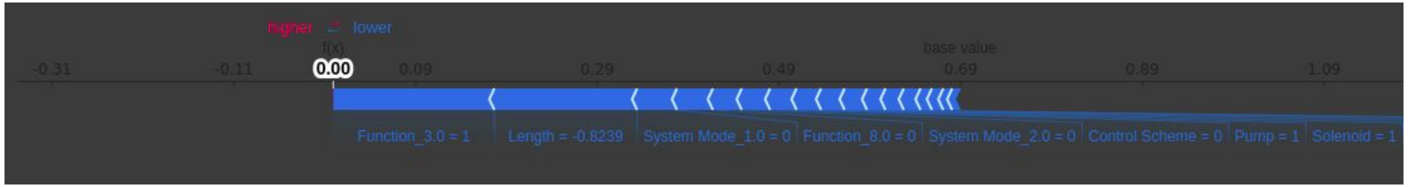
[-0.054, 23.94]

Actual value: [-0.05, 24.72]

Detecting the features: SHAP Values



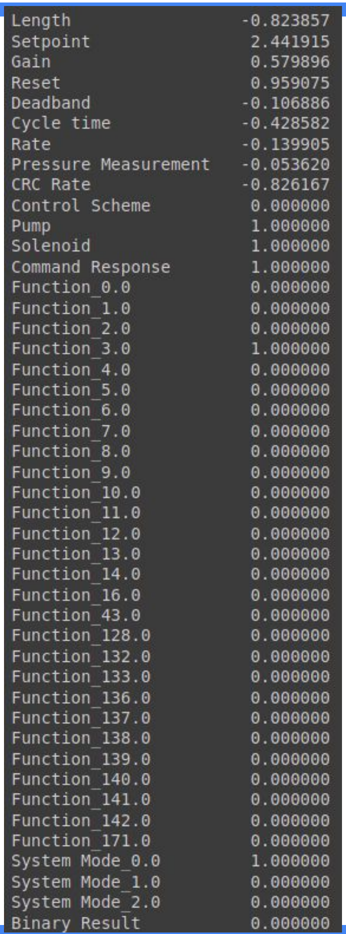
```
Length                  -0.823857
Setpoint                 2.441915
Gain                     0.579896
Reset                    0.959075
Deadband                -0.106886
Cycle time              -0.428582
Rate                    -0.139905
Pressure Measurement    -0.053620
CRC Rate                -0.826167
Control Scheme           0.000000
Pump                     1.000000
Solenoid                 1.000000
Command Response         1.000000
Function_0.0             0.000000
Function_1.0             0.000000
Function_2.0             0.000000
Function_3.0             1.000000
Function_4.0             0.000000
Function_5.0             0.000000
Function_6.0             0.000000
Function_7.0             0.000000
Function_8.0             0.000000
Function_9.0             0.000000
Function_10.0            0.000000
Function_11.0            0.000000
Function_12.0            0.000000
Function_13.0            0.000000
Function_14.0            0.000000
Function_16.0            0.000000
Function_43.0            0.000000
Function_128.0           0.000000
Function_132.0           0.000000
Function_133.0           0.000000
Function_136.0           0.000000
Function_137.0           0.000000
Function_138.0           0.000000
Function_139.0           0.000000
Function_140.0           0.000000
Function_141.0           0.000000
Function_142.0           0.000000
Function_171.0           0.000000
System Mode_0.0          1.000000
System Mode_1.0          0.000000
System Mode_2.0          0.000000
Binary Result            0.000000
```



The SHAP values help us determine the features that contribute to the decision of the rule-based classifier.

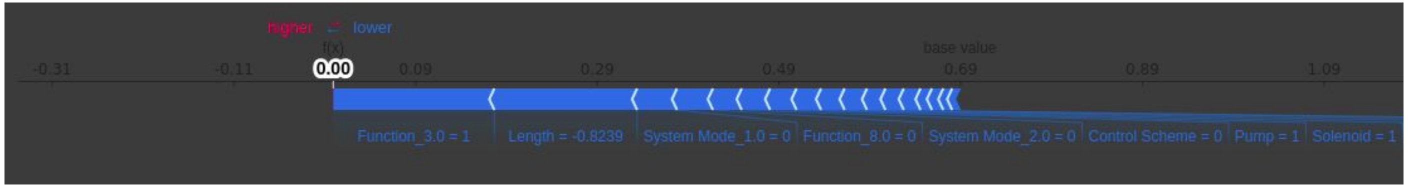
Ex: In this case Function_3.0, Length, SYstemMode_1.0 etc. have contributed.

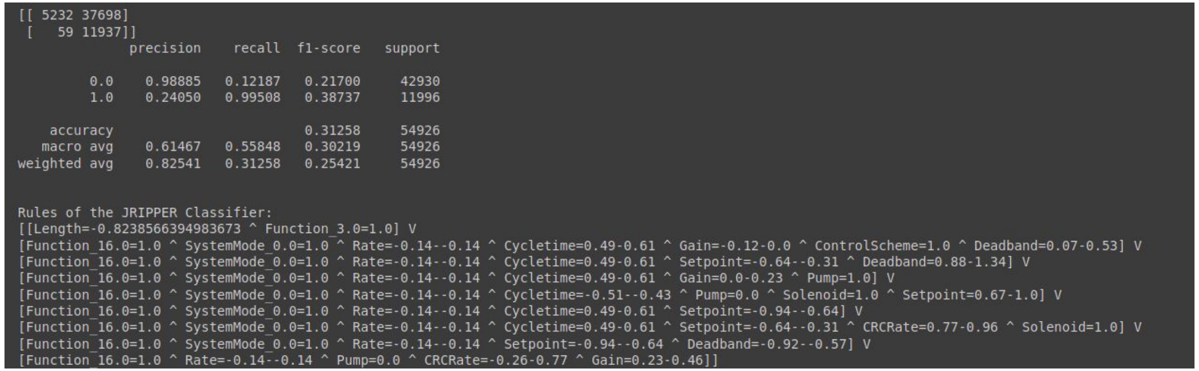
Example of an attack



```
Length                   -0.823857
Setpoint                  2.441915
Gain                      0.579896
Reset                     0.959075
Deadband                 -0.106886
Cycle time               -0.428582
Rate                     -0.139905
Pressure Measurement     -0.053620
CRC Rate                 -0.826167
Control Scheme            0.000000
Pump                      1.000000
Solenoid                  1.000000
Command Response          1.000000
Function_0.0              0.000000
Function_1.0              0.000000
Function_2.0              0.000000
Function_3.0              1.000000
Function_4.0              0.000000
Function_5.0              0.000000
Function_6.0              0.000000
Function_7.0              0.000000
Function_8.0              0.000000
Function_9.0              0.000000
Function_10.0             0.000000
Function_11.0             0.000000
Function_12.0             0.000000
Function_13.0             0.000000
Function_14.0             0.000000
Function_16.0             0.000000
Function_43.0             0.000000
Function_128.0            0.000000
Function_132.0            0.000000
Function_133.0            0.000000
Function_136.0            0.000000
Function_137.0            0.000000
Function_138.0            0.000000
Function_139.0            0.000000
Function_140.0            0.000000
Function_141.0            0.000000
Function_142.0            0.000000
Function_171.0            0.000000
System Mode_0.0           1.000000
System Mode_1.0           0.000000
System Mode_2.0           0.000000
Binary Result             0.000000
```

Correctly classified as normal by the IDS. Now we can find the features responsible for the decision.





```
[[ 5232 37698]
 [   59 11937]]
              precision    recall  f1-score   support

         0.0    0.98885   0.12187   0.21700     42930
         1.0    0.24050   0.99508   0.38737     11996

    accuracy                        0.31258     54926
   macro avg    0.61467   0.55848   0.30219     54926
weighted avg    0.82541   0.31258   0.25421     54926


Rules of the JRIPPER Classifier:
[[Length=-0.8238566394983673 ^ Function_3.0=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Gain=-0.12-0.0 ^ ControlScheme=1.0 ^ Deadband=0.07-0.53] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.64--0.31 ^ Deadband=0.88-1.34] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Gain=0.0-0.23 ^ Pump=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=-0.51--0.43 ^ Pump=0.0 ^ Solenoid=1.0 ^ Setpoint=0.67-1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.94--0.64] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Cycletime=0.49-0.61 ^ Setpoint=-0.64--0.31 ^ CRCRate=0.77-0.96 ^ Solenoid=1.0] V
[Function_16.0=1.0 ^ SystemMode_0.0=1.0 ^ Rate=-0.14--0.14 ^ Setpoint=-0.94--0.64 ^ Deadband=-0.92--0.57] V
[Function_16.0=1.0 ^ Rate=-0.14--0.14 ^ Pump=0.0 ^ CRCRate=-0.26-0.77 ^ Gain=0.23-0.46]]
```

Therefore other features like Gain, Reset, Deadband, etc. can be attacked.

Overview:

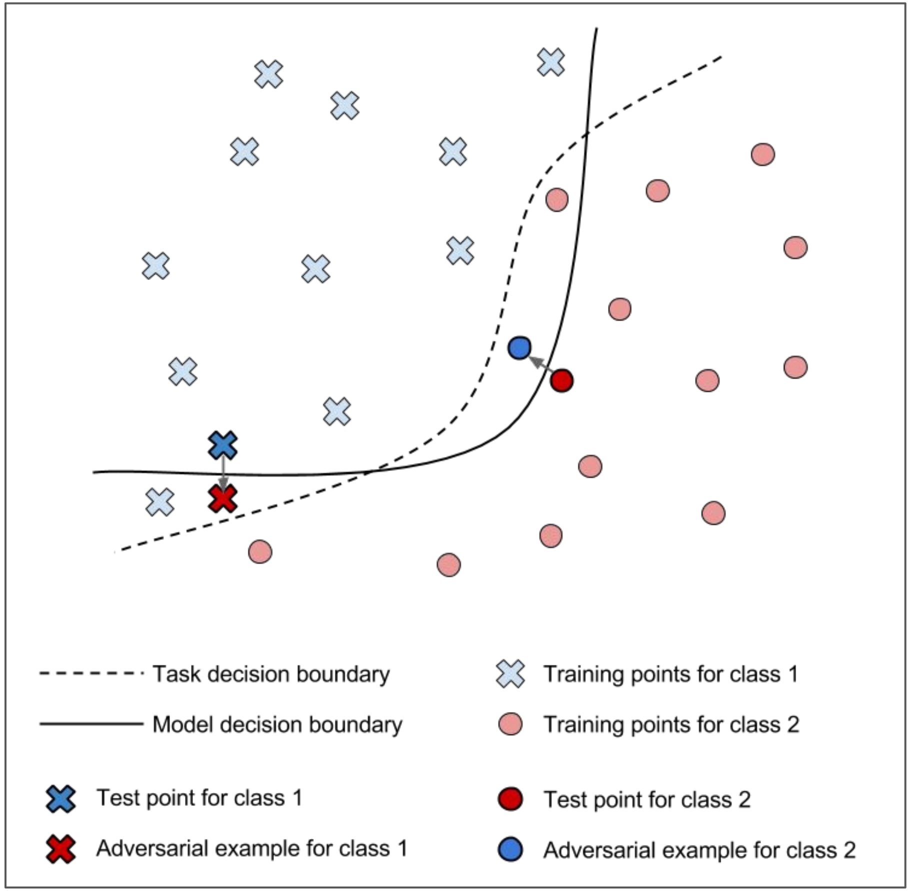
Deep Learning Models

DNNs are currently the state of the art models for IDS solutions. They help extract complex features and understand rigorous relationships between various input features.

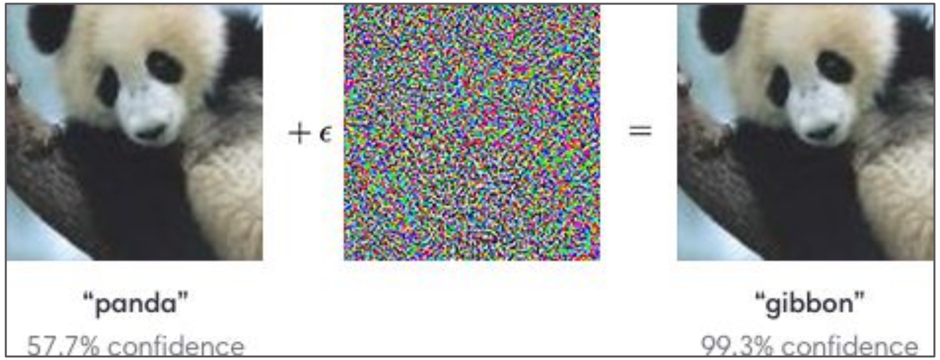
Achieves high accuracy rates and low false positive rates. In our experiment the DNN models achieved an accuracy around 96%-97%.

Problem: Vulnerable to adversarial attacks

Adversarial attacks



Task decision boundary
Model decision boundary
Training points for class 1
Training points for class 2
Test point for class 1
Adversarial example for class 1
Test point for class 2
Adversarial example for class 2



How to generate adversarial samples?

- FGSM : Attacking directly without any constraints on the features

$$x^{adv} = x + \varepsilon \cdot \mathtt{sign}(\nabla_x J(x, y_{true})),$$

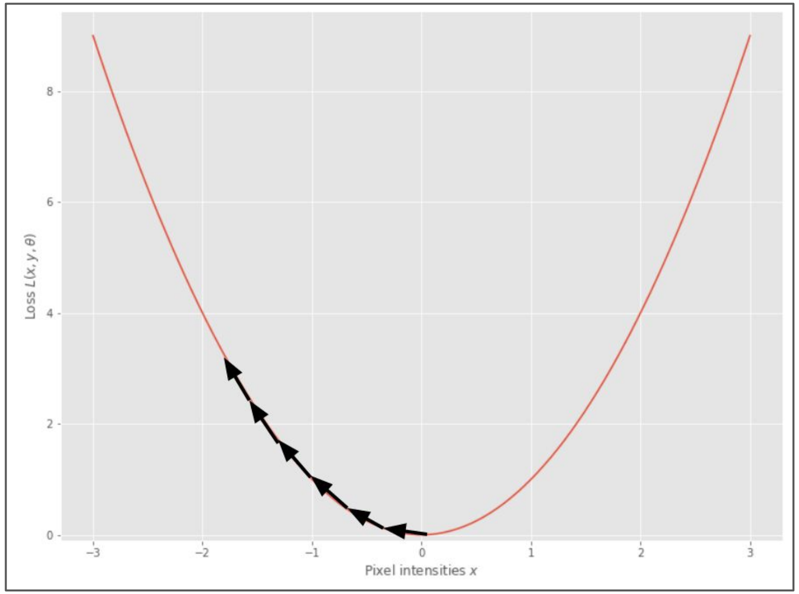
where

x is the input (clean) image,

x^{adv} is the perturbed adversarial image,

J is the classification loss function,

y_{true} is true label for the input x.



Types of Adversarial Attacks

- White Box Attacks:
  - Direct access to the underlying model
  - Aware of all the weights/parameters and their associated gradients can be directly computed
- Black Box Attacks:
  - Access to the model only with inputs and outputs
  - No knowledge of the underlying model
  - Aware of the input features and their properties(binary and one-hot encoded)

In our research work we focus on black box approaches for adversarial samples as they are more realistic in nature.

Black Box Attacks

- Substitute Models:
  - Build a surrogate model, then perform white boac attacks on the surrogate model
  - Transfer the adversarial samples to the original model
  - Problems: nature of the model; transferability of generated attack samples

- ZOO (Zeroth Order Optimization Attacks):
  - Utilize the gradient approximation to compute the grad values and directly generate adversarial samples.
  - No need to know the underlying model and the performance of samples are also guaranteed.

$$\hat{g}_i := \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h}, \qquad f(\mathbf{x}) = \max\{\log[F(\mathbf{x})]_{t_0} - \max_{i \neq t_0} \log[F(\mathbf{x})]_i, -\kappa\},$$

Additional Constraints

- **Importance of features:**
  - Using the gradient values to select the features. We compute the gradients and then use the absolute value of the gradients to determine which feature will give us the best results.
- **Handling binary and continuous values:**
  - Ensures that binary values can only change from 0→ 1 or from 1→ 0.
  - Also the constraints on the one-hot encoded values are handled by restricting the direction of change on the binary values.
- **No changing features under attack:**
  - Features that are currently under attack can't be modified. This will preserve the attack characteristics of the packet.
- **Constraints on the value of epsilon and other update_restrictions** were analysed. Tradeoff between accuracy and change in packet information(L1).

Adversarial Sample Generation Algorithm (Based on ZOO)

```
X* = X (this is classified by the model as an anomaly)
While(model(X*)==1 and update_constraints):
    Get the data_grad of the input (Via Black Box models)
    Select the top = sort(|data_grad|).get_indices(). Traverse this list from highest to lowest.
        If ind is the feature under attack use the next index
        If ind is a continuous feature modify the value using ep * sign(data_grad[ind])
        If ind is a binary feature: (Only 0 → 1 update is possible and the other index that is 1 is made 0)
            If X*[ind]==0 and data_grad[ind]>0:
            For each one-hot encoded feature set {S1, S2, S3} find which set Si the current index belongs
            ind_prev = Index of the feature that was already 1 in that particular set Si
            temp = X*; temp[ind_prev] = 0; temp[ind] = 1
            if(model(temp) - model(X*) < 0 ): X* = temp
            Else continue to the next index
```

Comparing Attack Performance

Black Box

Correct : 0.49379977770561573

Failed: 0.01723964107547168

Mis: 0.12985179944565803

Zero: 0.35910878076508845

Total: 9919

Percentage of adversarial examples generated = 96.62 (correct/(correct + failed))

Total number of adversarial examples = 4898

Average change per sample 1.1654961408036444

White Box

Correct : 0.5028732735154754

Failed: 0.00816614578082468

Mis: 0.12985179957657023

Zero: 0.35910878112712974

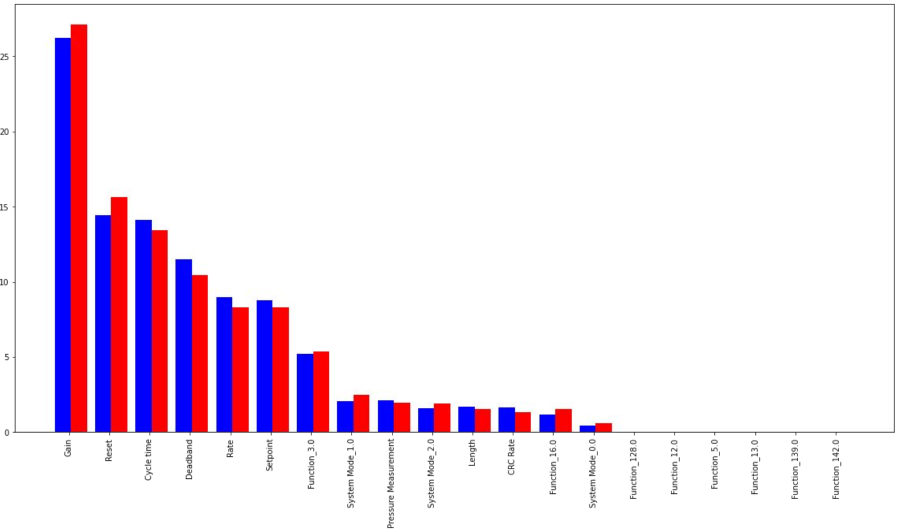
Total: 9919

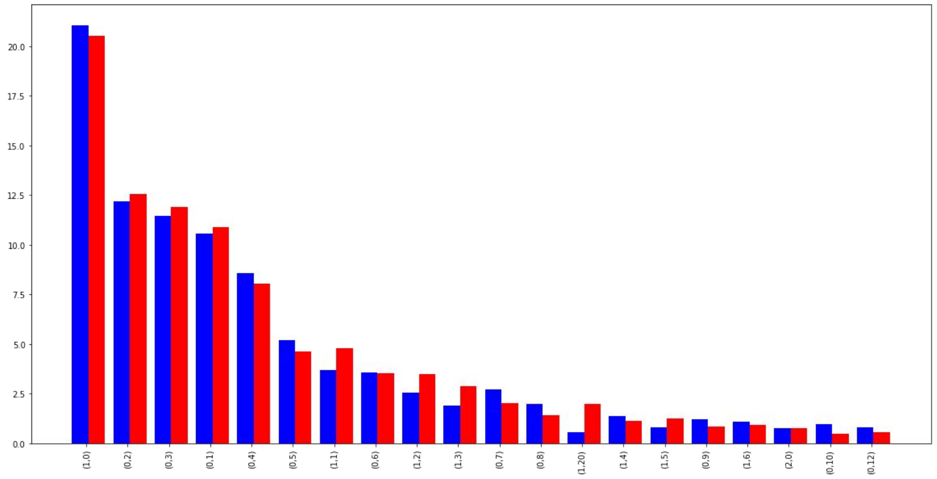
Percentage of adversarial examples generated = 98.40

Total number of adversarial examples = 4988

Average change per sample 1.1078

Feature Selection





We can achieve similar levels of performance like the white box attacks with zero knowledge about the underlying model and only black box access to outputs.

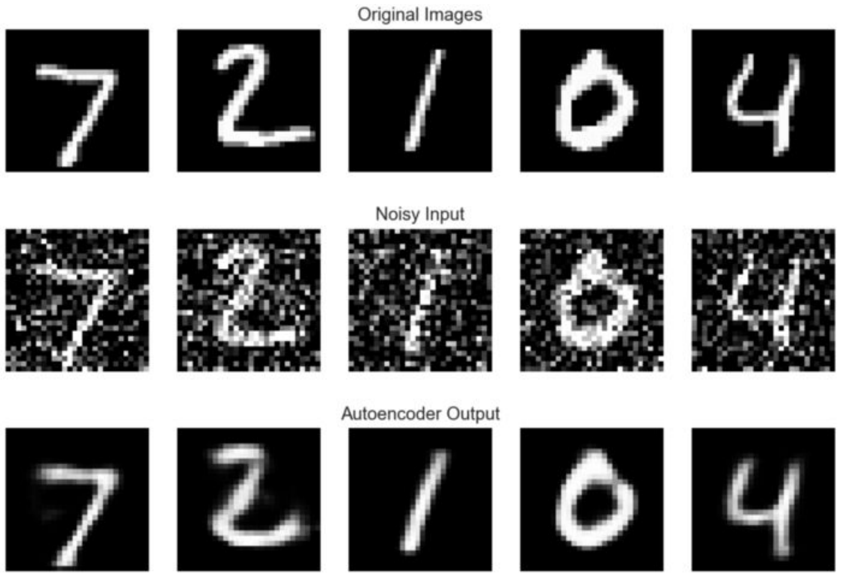
Defense against adversarial attacks

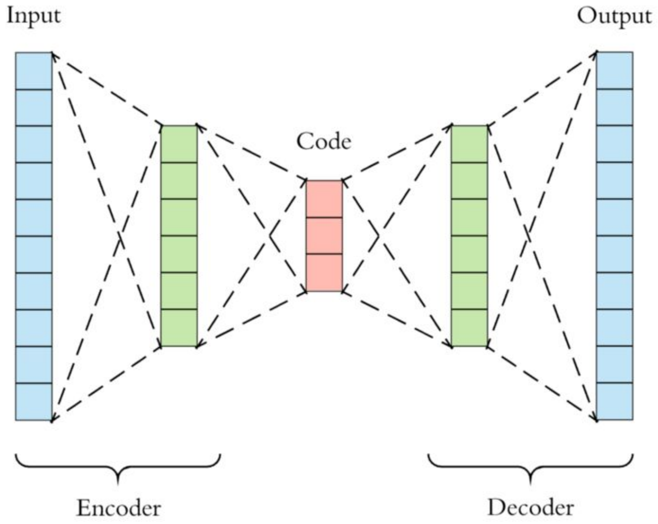
- **Data Based Defenses:**
  - The model is unchanged but the input to the model is processed beforehand.
  - Ex: Denoising autoencoder

- **Model Based Defenses:**
  - We modify the model parameters by training with samples/labels that will make the model robust towards adversarial attacks.
  - Ex: Defensive Distillation, Adversarial training

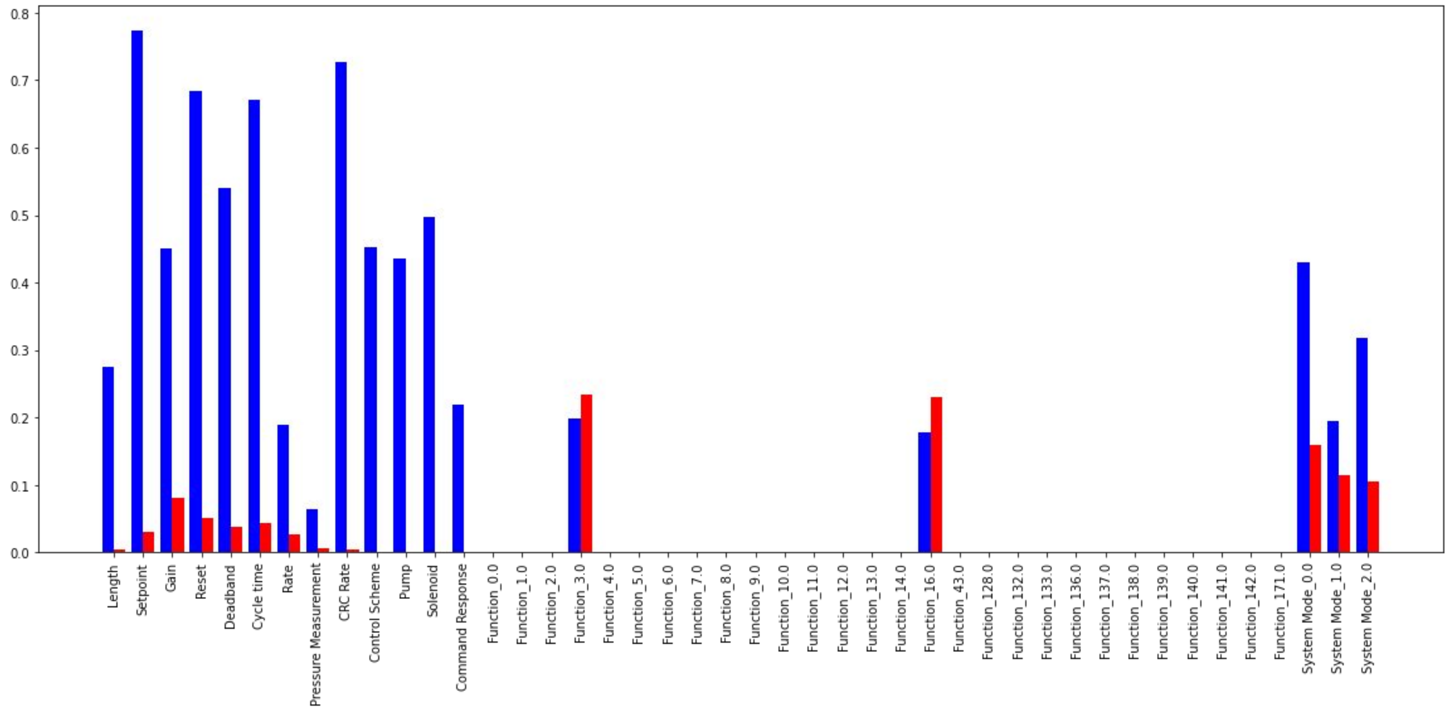
Data Based Defenses: DAE

In this method we use a denoising autoencoder to remove the perturbations in the input sample. AE are known to process the noisy data and output clean data.



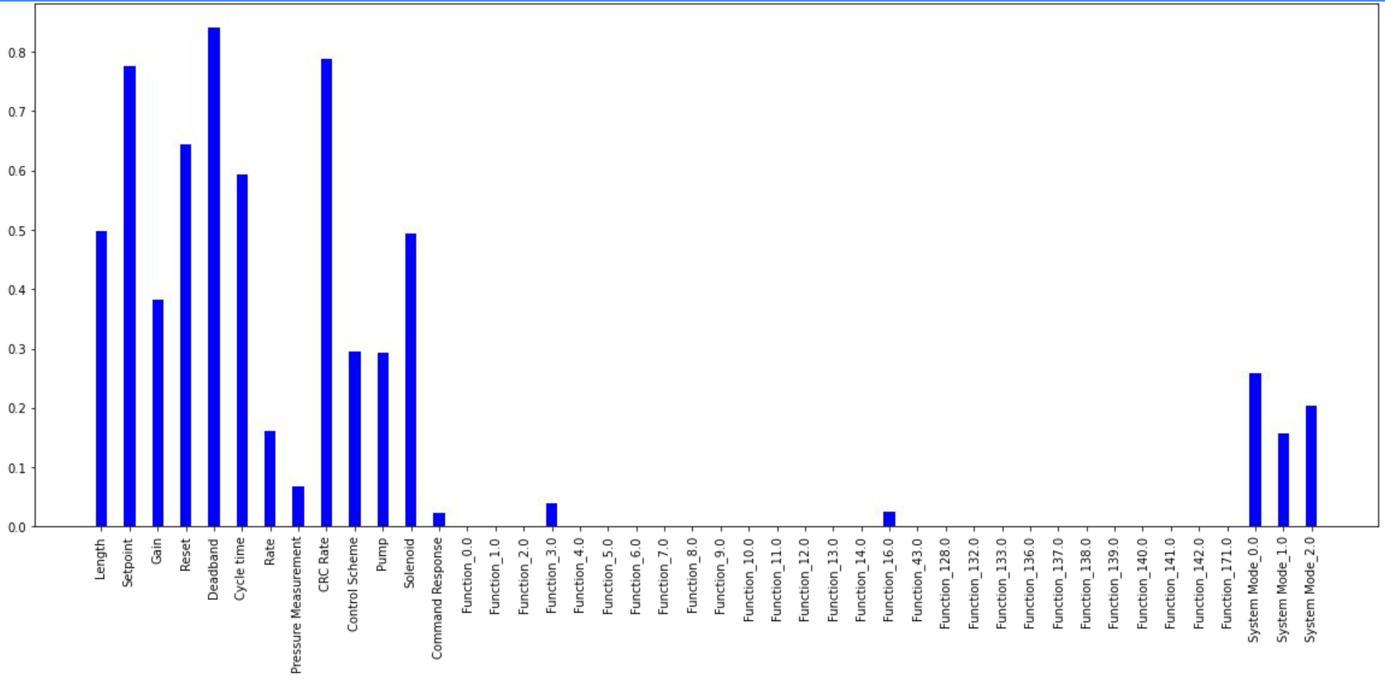


Removing adversarial perturbations



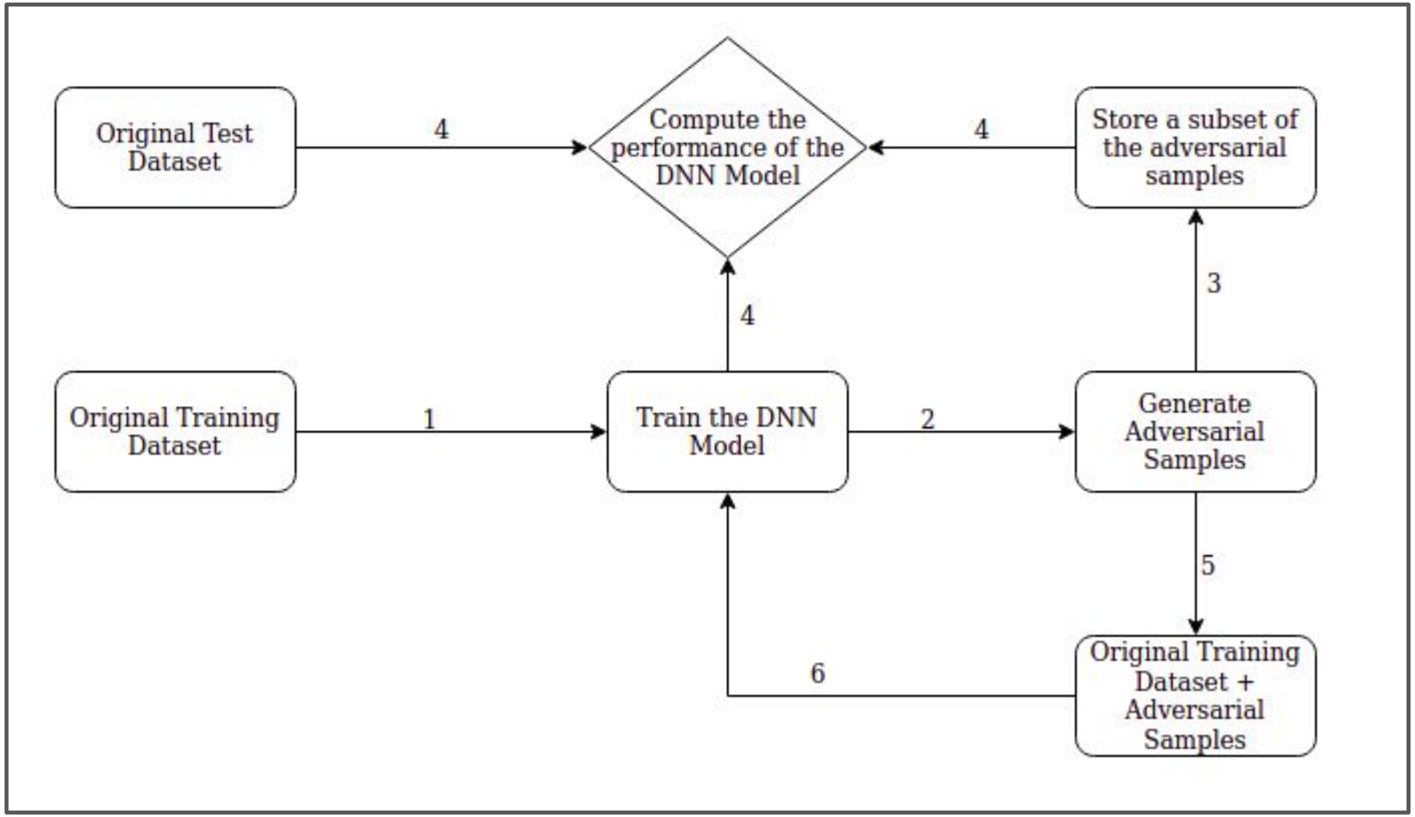
After processing the model removes most of the adversarial perturbations.

Adding perturbations to original data



Unfortunately even original samples are perturbed because of which the attack capabilities might be lost. Unlike images in this case each feature has meanings therefore perturbing will change the nature of those data points.

Model Based Defenses: Adversarial Training

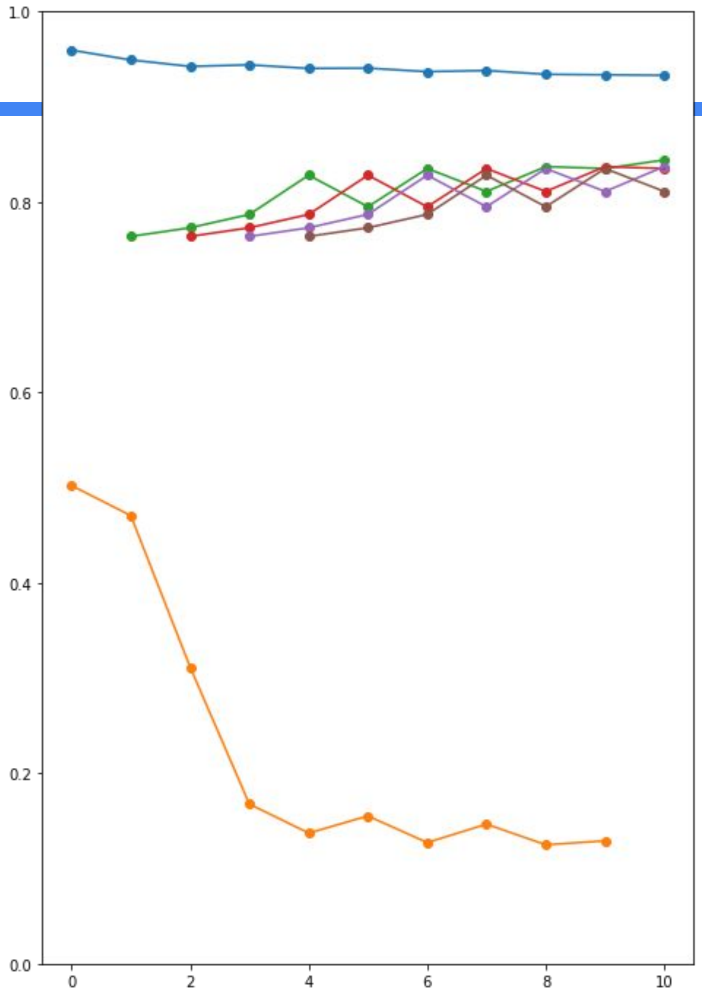

Original Test Dataset
4
Compute the performance of the DNN Model
4
Store a subset of the adversarial samples
4
3
Original Training Dataset
1
Train the DNN Model
2
Generate Adversarial Samples
5
6
Original Training Dataset + Adversarial Samples

Results: Adv. Training

Blue: Accuracy of the model post each training

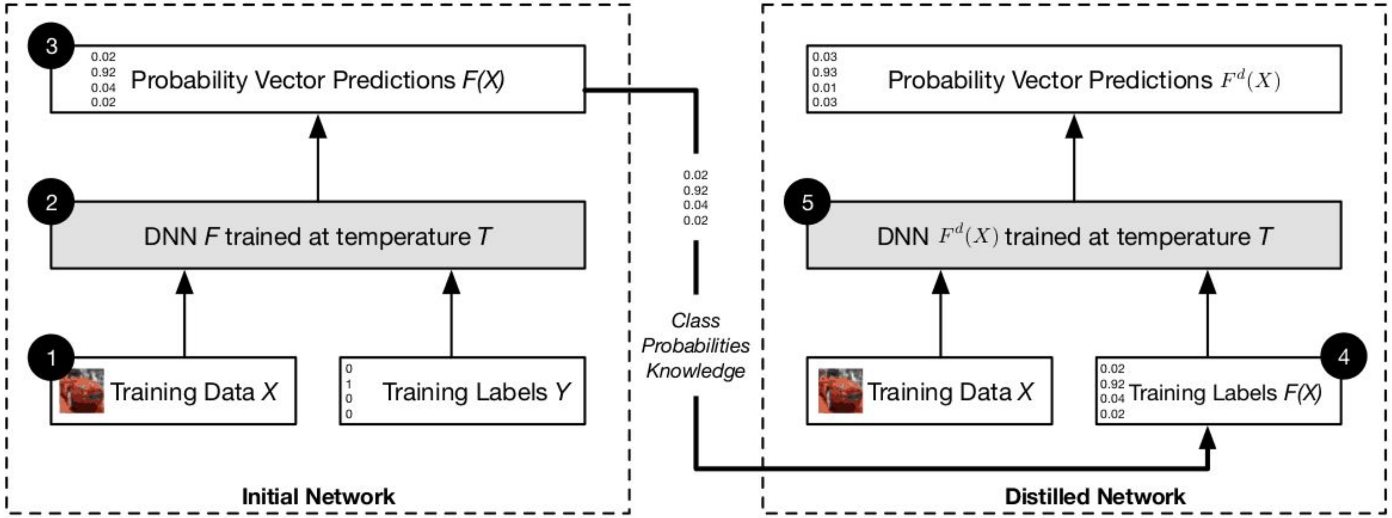
Green, Red, Purple, green: Accuracy of detecting the adversarial samples

Orange: % adversarial examples generated from the original dataset



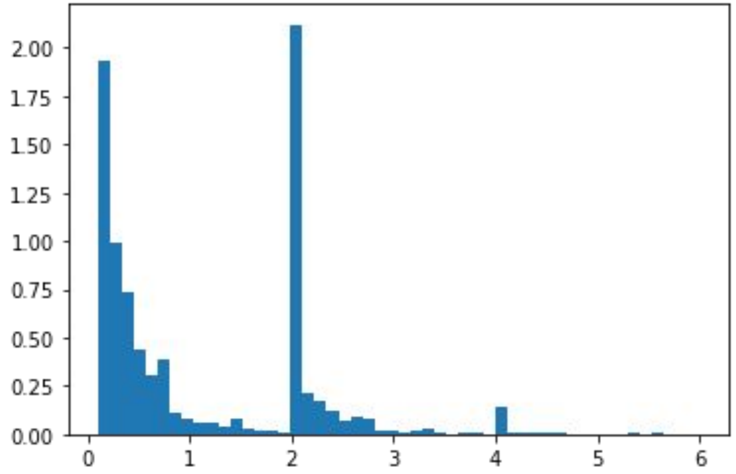
Model Based Defense: Defensive Distillation

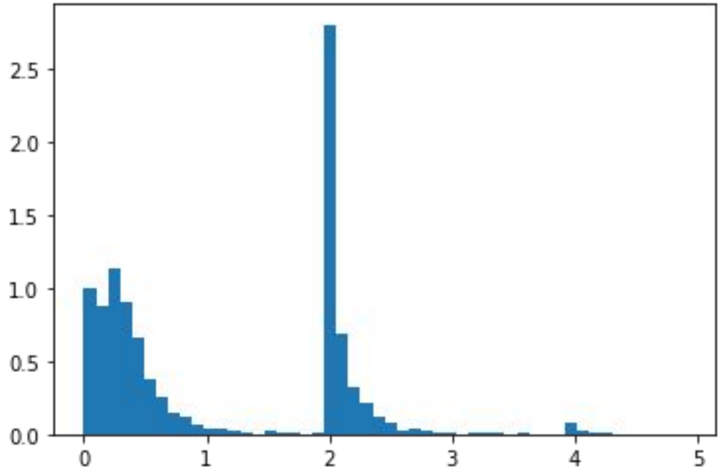
Using soft labels instead of hard ones helps to smooth the decision boundaries thus making the gradients of the model lesser in magnitude making the adversarial sample generation process difficult.



Results: Defensive Distillation

Model Used	Accuracy on test set	Adversarial Samples
Original Model	95.97	4988
Defensive Distillation	94.93(-1.04%)	2606(-47.75%)



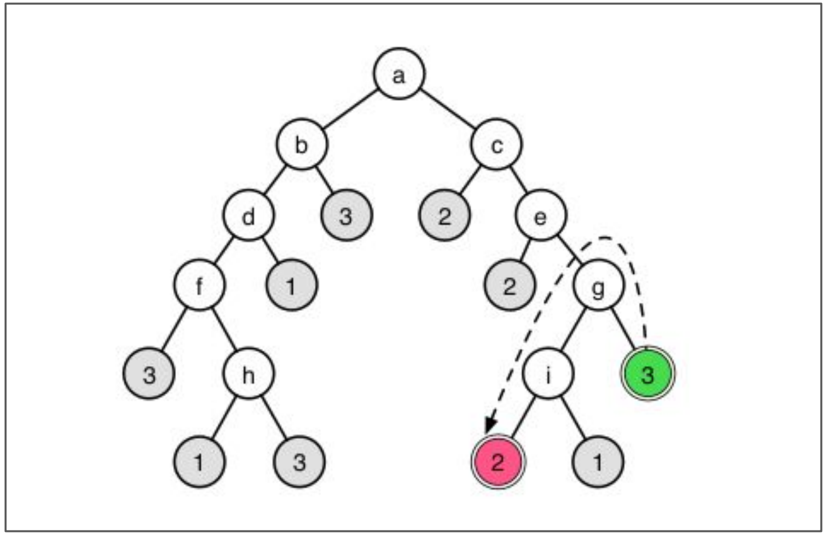


Overview:

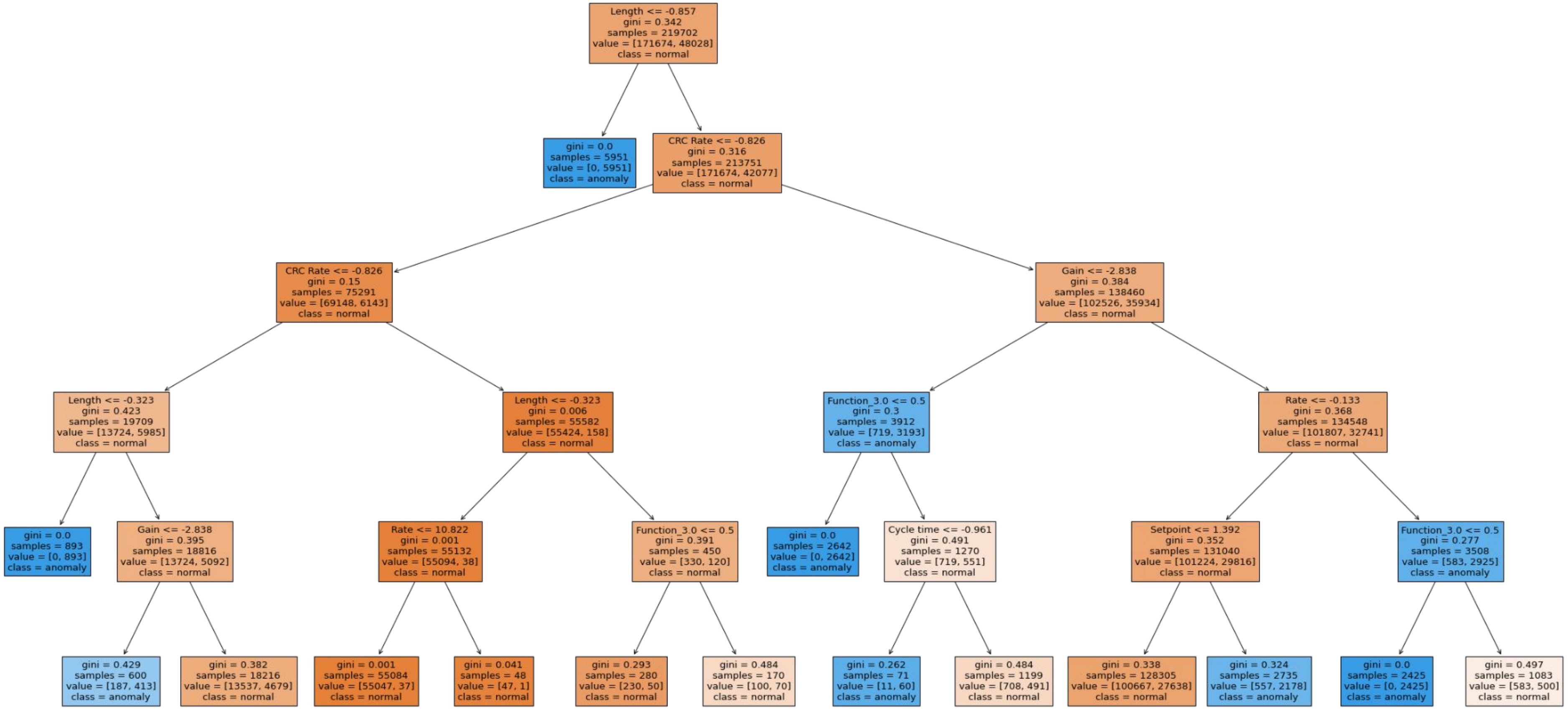
Tree Based Models

In our research task, RF and DT models achieve very high performance around 96-97% due to the relative smaller size of our dataset.

Attacking tree based models using gradients is not possible. Also significant feature changes might be required to generate adversarial samples.



Sample DT Attack



Overview:

Conclusion

- Gas Pipeline Dataset was select after careful analysis
- Multiple different types of models were created and their performance were analyzed.
- The vulnerabilities of each models are exploited and suitable defense techniques were discussed. [Black Box Approach]
- Utilized understandable AI techniques to attack rule-based models.
- Extended the ideas in CV to cybersecurity by desinging algorithms that satisfy additional constraints.