CONTENTS

Background and Context

The project began with the initial idea of building a computer vision module that would use machine learning to classify users and allow or deny access to a particular system. The team successfully trained the machine learning model and implemented the basic functionality of the module on a machine rig. However, the team soon realized the need to expand the scope and vision of the project to include more robust security implementations.

With team members' expertise in areas such as IOT, cybersecurity, AI/ML, and networking, a stream of new module ideas emerged. To streamline the project and prevent scope creep, the team decided to set an endpoint for additional module implementations.

The team then developed a framework that would allow different modules to work together in a coordinated manner. The final set of modules included Network Gateway, Honeypot Module, IOT Module, Blynk Cloud Module, Computer Vision Module, Surveillance Module, and Android App Module. The team estimated the project requirements and cost within their budget and expertise and worked on creating the first functional model and workflow.

To make the information easily accessible, the team decided to deploy the entire system onto various hosting platforms and display relevant information on an android application in real-time. This approach provided users/admins with a single platform for all network and premises security implementations, making the IDS a comprehensive and effective solution.

Overall, the team's focus on expertise, cost, and implementation led to the successful development of a highly functional and efficient IDS system that can be used in various settings, including homes, businesses, and government organizations.

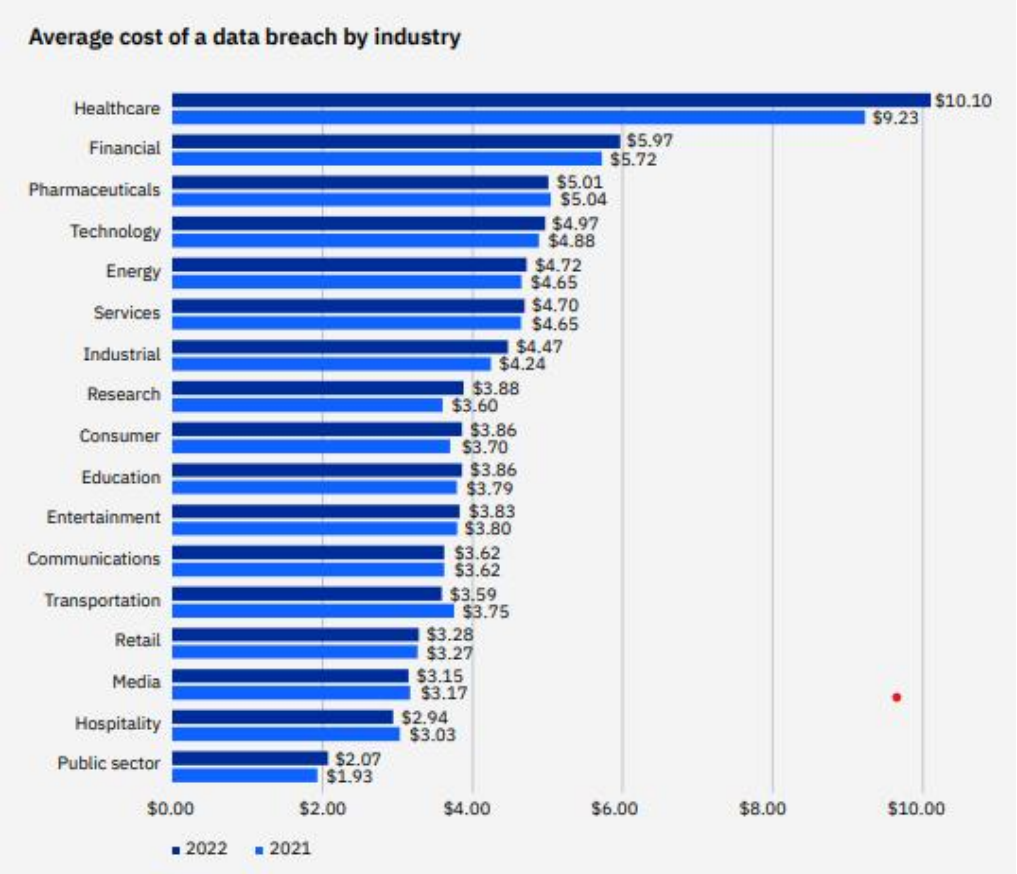
Problem Statement and Justification

Small-scale industries, general public, and unaware users are particularly vulnerable to cyber attacks due to their limited resources, lack of awareness, and reliance on technology. Small-scale industries, in particular, often lack the cybersecurity infrastructure and resources of larger organizations, making them more susceptible to attacks. According to a recent survey, 43% of cyber attacks target small businesses.

General public and unaware users are also at risk, as they may not be familiar with basic cybersecurity practices and may unknowingly expose themselves to cyber threats. This includes using weak passwords, clicking on suspicious links, and downloading malicious software.

Furthermore, many people are unaware of the importance of regularly updating their software, antivirus, and firewalls. This leaves them vulnerable to attacks that exploit known vulnerabilities in outdated software and systems.

The consequences of a cyber attack on small-scale industries, general public, and unaware users can be devastating, including financial losses, reputational damage, and theft of sensitive data. It is therefore essential that everyone takes proactive steps to protect themselves against cyber attacks, such as educating themselves on basic cybersecurity practices, using strong passwords, keeping their software up to date, and being vigilant against suspicious activity.



JavaScript: JavaScript is the primary language used for implementing the frontend of the project, including the user interface, interactivity, and dynamic rendering of network information. It is widely supported by web browsers and allows for seamless client-side scripting.

HTML/CSS: HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are used for structuring and styling the project's web pages. HTML is responsible for defining the page structure and content, while CSS is used for visual presentation, including layout, colors, and typography.

Arduino (C++): Arduino, which uses a simplified version of C++, is employed for programming the ESP32 CAM module. It enables low-level control of the camera module's functions, configuration, and interactions with other components.

Python: Python is utilized for the backend server implementation, including the Blynk cloud module and handling data transmission between the network gateway and the admin control application. Python's versatility and extensive libraries make it suitable for server-side programming tasks.

Java: Java is used for developing the admin control application, which provides an interface for managing and controlling the network gateway. Java's object-oriented programming paradigm and rich ecosystem make it well-suited for building robust desktop applications.

XML: XML (eXtensible Markup Language) is used for data representation and configuration in various parts of the project. It provides a standardized format for storing and exchanging structured data, ensuring compatibility and ease of integration between different components.

The combination of these programming languages forms the core technology stack for the project, enabling the implementation of different modules and functionalities across the frontend, backend, embedded systems, and desktop application layers

Computer Vision Module

Overview:

YOLOv8, an advanced version of the YOLO (You Only Look Once) object detection model, introduces significant improvements over its predecessors, such as a new backbone network, a refined loss function, and an anchor-free detection head. Developed by Ultralytics, YOLOv8 is designed for high-speed and accurate image segmentation tasks, making it a state-of-the-art tool in computer vision.

Image Segmentation with YOLOv8:

Definition: Image segmentation partitions an image into segments, each corresponding to different objects or regions of interest.

Functionality: YOLOv8 starts by detecting objects and predicting their locations and classes. These predictions are used to generate segmentation masks, which define pixel-level boundaries of each object. Additional techniques, like contour detection, refine these masks for better accuracy.

Instance Segmentation:

YOLOv8 supports instance segmentation, which differentiates individual instances of the same object class. Pre-trained models like "yolov8n-seg.pt" are trained on the COCO128-seg dataset, enabling detailed analysis of objects in images.

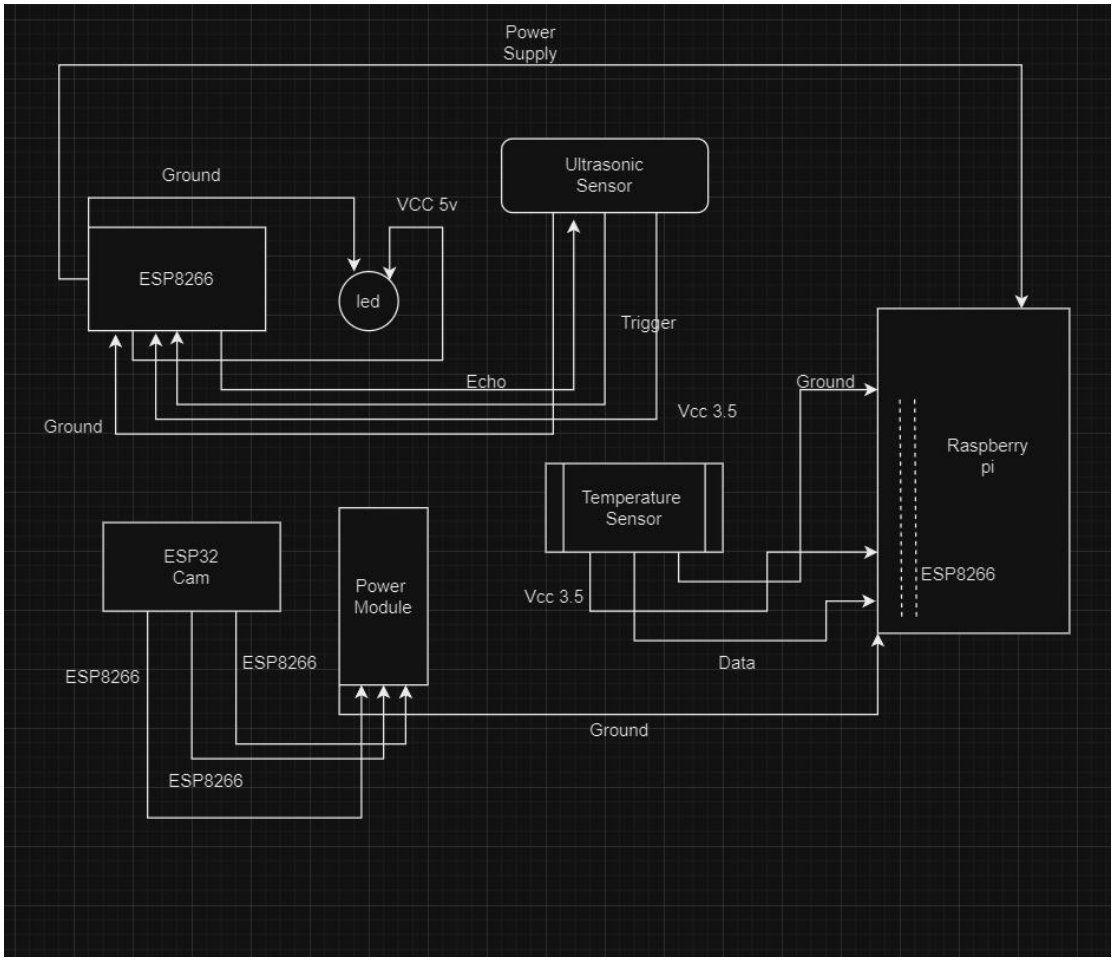
Training Details: The model is trained for 100 epochs on 640-pixel images, enhancing its ability to accurately segment objects.

Advantages of YOLOv8:

Speed: Capable of processing 81 frames per second, YOLOv8 is significantly faster than other models like Mask R-CNN, making it ideal for real-time applications such as self-driving cars and security systems.

Accuracy: Achieves high mean average precision (mAP), outperforming models like Detectron2. This is due to its advanced architecture and loss function, which minimize false positives and negatives.

Circuit Diagram



The circuit diagram in the product has been built using major components like a raspberry pi , An ESP32 cam , an ESP 8266 module and a power converter . Some other major censors like temperature sensor ultrasonic sensor and PIR sensors have also been used in order to create a functioning model of the IOT circuit .The modules have been connected in serial manner and are power dependent on only one of the power input cables .The circuit diagram here presented is developed in a manner as such the ESP8266 module Drives power from the raspberry pi usb port simultaneously powering the ultrasonic sensor and consuming hits eco and data pin .

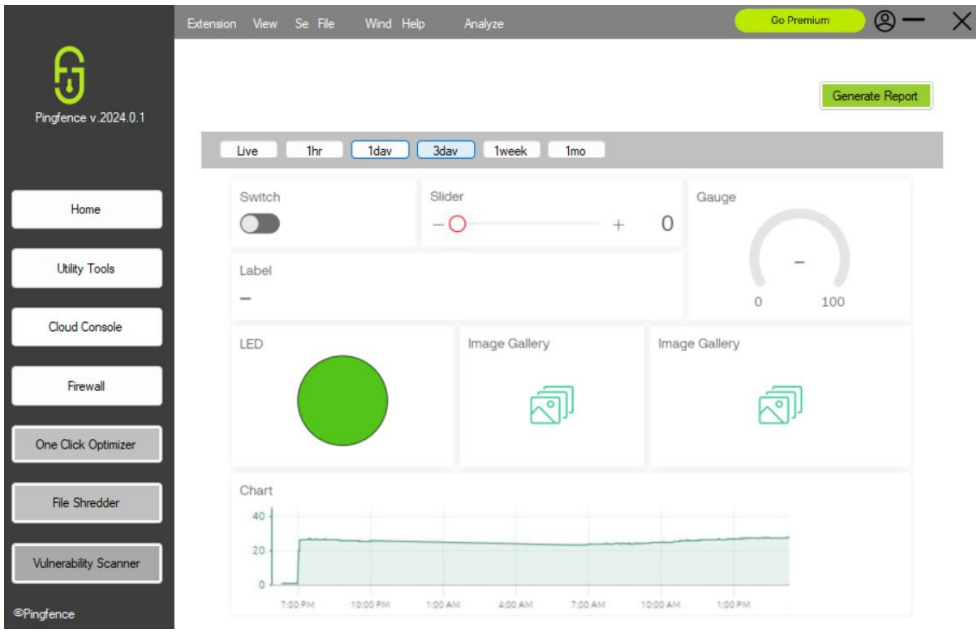
The ESP32 cam connected with the power module drives power from the raspberry pie and doesn’t need any other power adapter it also connects its surveillance module to the computer vision module of the raspberry buyer and makes artificial intelligence decisions in real time .

The components altogether consume a 3.5 voltage of electricity while generating a latency of 2000 milliseconds which was necessary for the IOT module to process data in a serial manner .

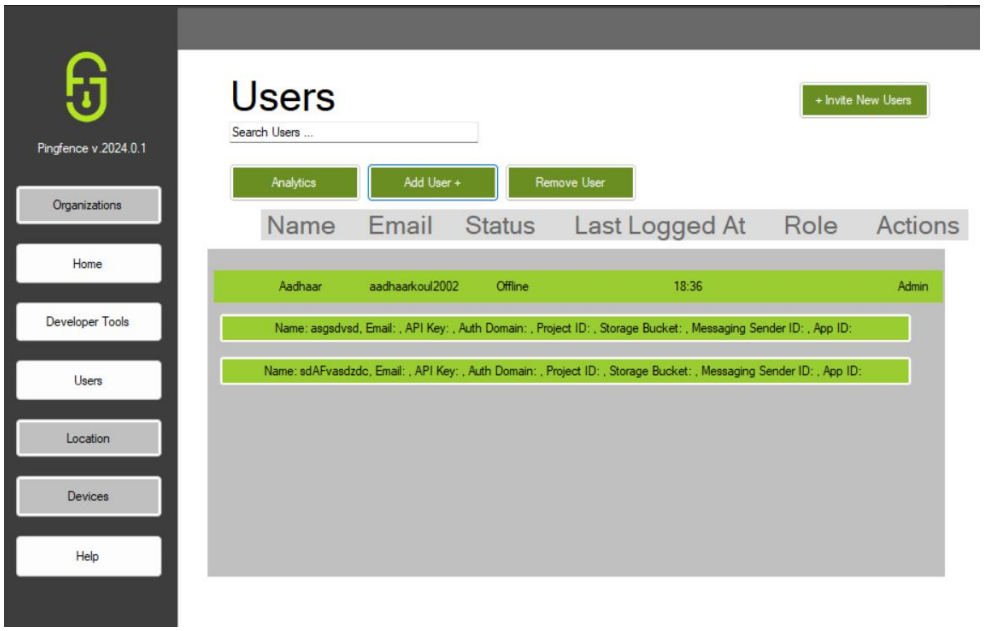
The one stop Solutions Acts as a mothership for all the IOT network and parameter components

It uses smart notification system of the IOT Network To generate smart analytics that can be viewed on the admins smartphone web platform as well as the system software .

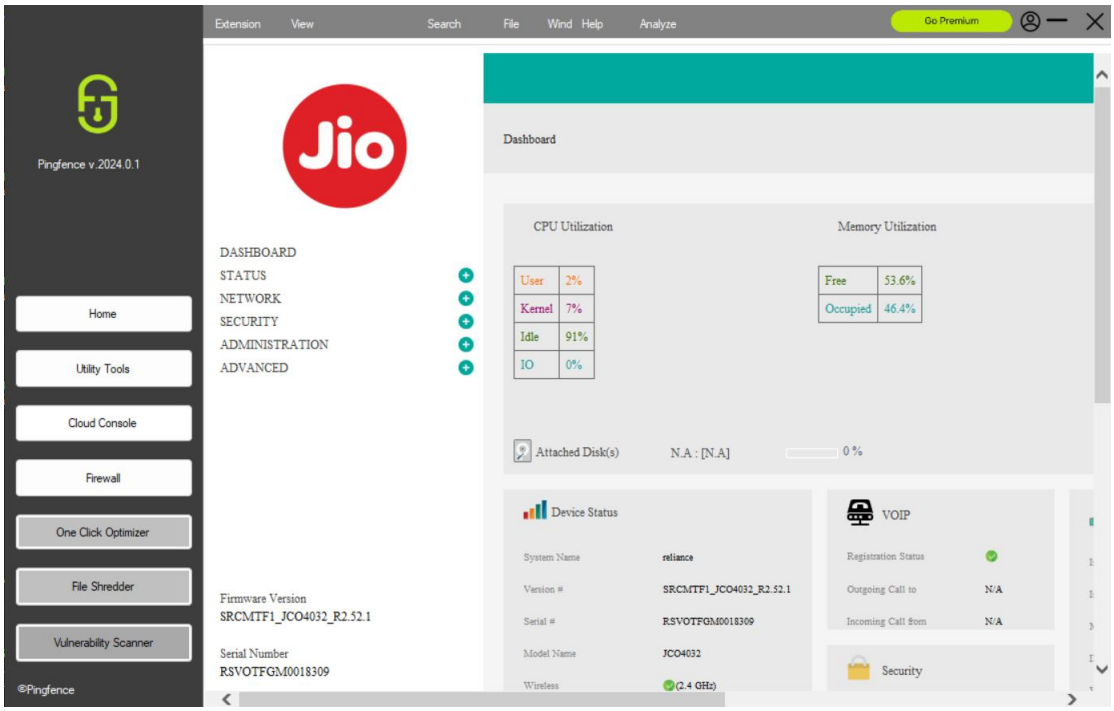
Some of the main features of the one stop solution can be seen below:



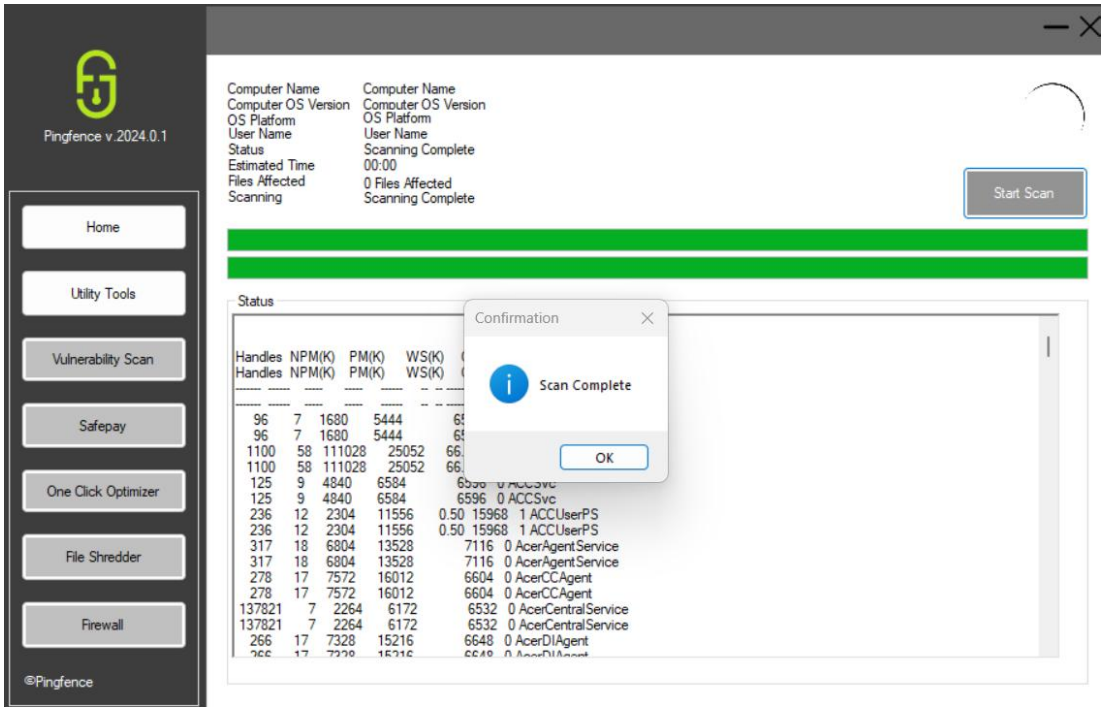
The above picture depicts the analytics dashboard that is presenting one day previous league collected data from the iot components onto the one stop solution dashboard .



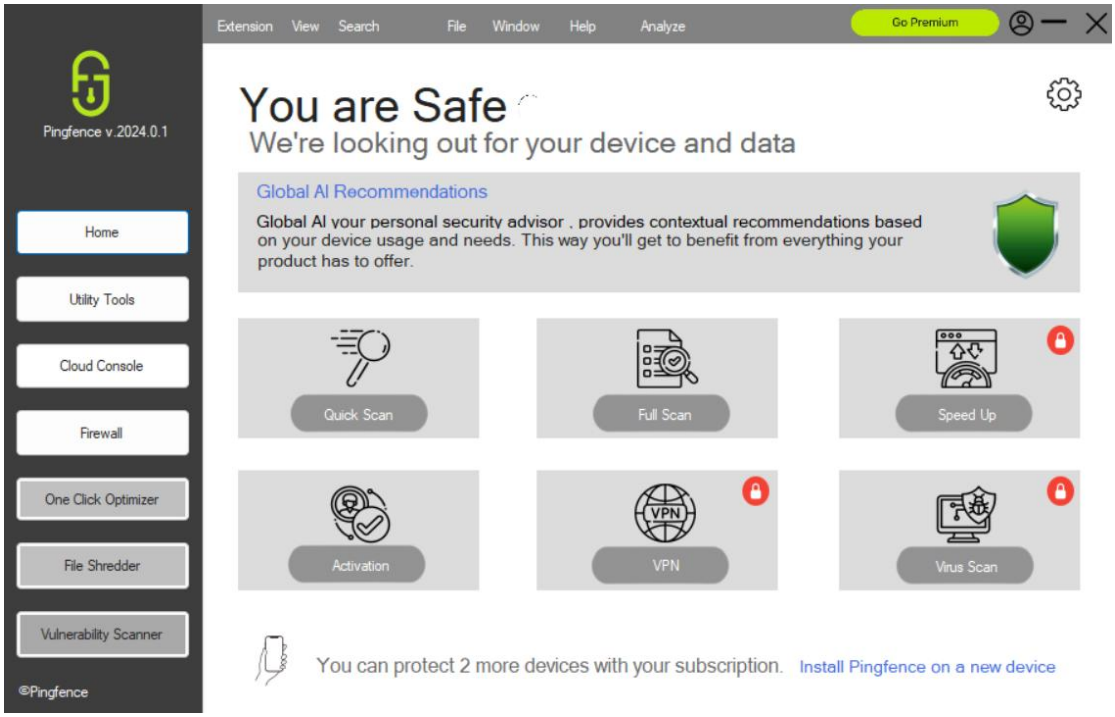
Adding users through multiple apis functionality .



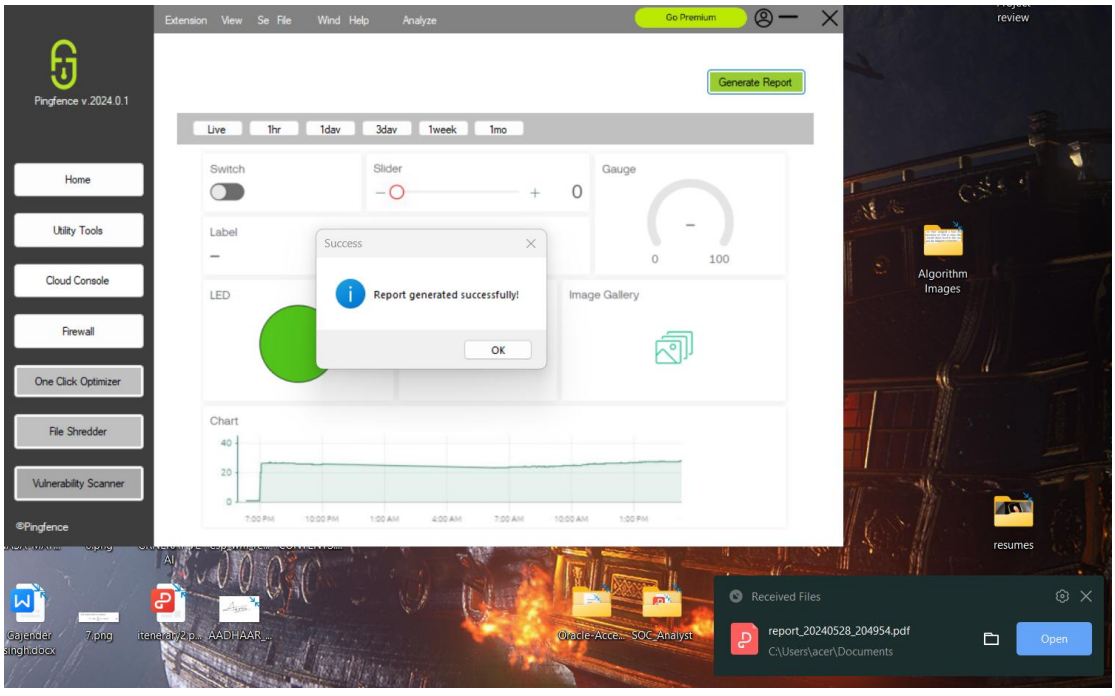
Inbuilt firewall functionality of the one stop solution platform



Complete and quick scan functionality of the oss platform .



Dashboard featuring all the contents and functions of the OSS platform .



The one stop solution was developed on Visual Studio code using C as the scripting language that uses state of the art library to integrate with the cloud consoles as well as the iot components for a seamless and an attractive interface .

Smart Notification System

The smart notification system is built upon the deep neural network concept by using libraries like Relu and Ultra Latics and tensorflow in order to deeply classify the level of intrusion in the iot components and generate smart notifications .

Below given court depicts the implementation of tensor flows deep neural network technique in order to classify whether a threat is fatalor not .

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Load the dataset
data = pd.read_csv('surveillance_data.csv', encoding='utf-8')
X = data[['sensor_reading']].values
y = data['threat_level'].values

# Normalize the feature data
X = X / np.max(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the neural network model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

# Train the model
try:
    model.fit(X_train, y_train, epochs=50, batch_size=10, validation_split=0.2)
except Exception as e:
    # Handle the exception and ensure proper encoding of error message
    error_msg = f"An error occurred during training: {str(e)}"
    print(error_msg.encode('utf-8').decode('utf-8'))  # Ensure proper encoding and decoding

# Save the model to a file
model.save('threat_classifier.h5')
print("Model trained and saved to 'threat_classifier.h5'")
```

The below code depicts the raspberry pi implementation using the GPIO pins that uses an smtp server to deliver messages to the user slash admin based on the deep neural network processing .

```python
import time
import joblib
import smtplib
from email.mime.text import MIMEText

# Setup GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# SPI setup for MCP3008
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1350000

# Load the trained model
model = joblib.load('threat_classifier.h5')

# ADC Channel where the surveillance module is connected
adc_channel = 0

# Email setup
smtp_server = 'smtp.your_email_provider.com'
smtp_port = 587
smtp_user = 'ckeve911@gmail.com'
smtp_password = 'Fallinlovewithfailure@01'
recipient_email = 'aadhaarkoul2002@gmail.com'

Comment Code
def read_adc(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    data = ((adc[1] & 3) << 8) + adc[2]
    return data

Comment Code
def predict_threat_level(sensor_value):
    prediction = model.predict([[sensor_value]])
    return prediction[0]

Comment Code
def send_notification(threat_level):
    subject = f'Intrusion Detected: Threat Level {threat_level}'
    body = f'Threat level detected: {threat_level}\nTake appropriate action.'
    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = smtp_user
    msg['To'] = recipient_email

    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
        server.login(smtp_user, smtp_password)
        server.sendmail(smtp_user, recipient_email, msg.as_string())

try:
    while True:
        adc_value = read_adc(adc_channel)
        threat_level = predict_threat_level(adc_value)
        print(f'Threat Level: {threat_level}, ADC Value: {adc_value}')

        send_notification(threat_level)

        time.sleep(10)  # Delay between checks
except KeyboardInterrupt:
    pass
```

Even below is an example of a smart notification generated by the raspberry pie module that was seen on the admins smartphone when an intrusion was detected on the ultrasonic sensor.