National College of Ireland

# Adversarial Resilience in Malware Detection: A Two-Stage Structural Analysis Approach for Robust Cybersecurity

## Configuration Manual

MSc Research Project
Msc in Artificial Intelligence

Pavithrasri Udayakumar
Student ID: x22182730

School of Computing
National College of Ireland

Supervisor: Anh Duong Trinh

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Pavithrasri Udayakumar |
| **Student ID:** | x22182730 |
| **Programme:** | MSc Artificial Intelligence |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Anh Duong Trinh |
| **Submission Due Date:** | 05/01/2023 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 1110 |
| **Page Count:** | 12 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

| | |
|---|---|
| **Signature:** | Pavithrasri Udayakumar |
| **Date:** | 05th January 2024 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

| | |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | ☐ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Manoj Kumar Periyasamy
x22153209

## 1 Introduction

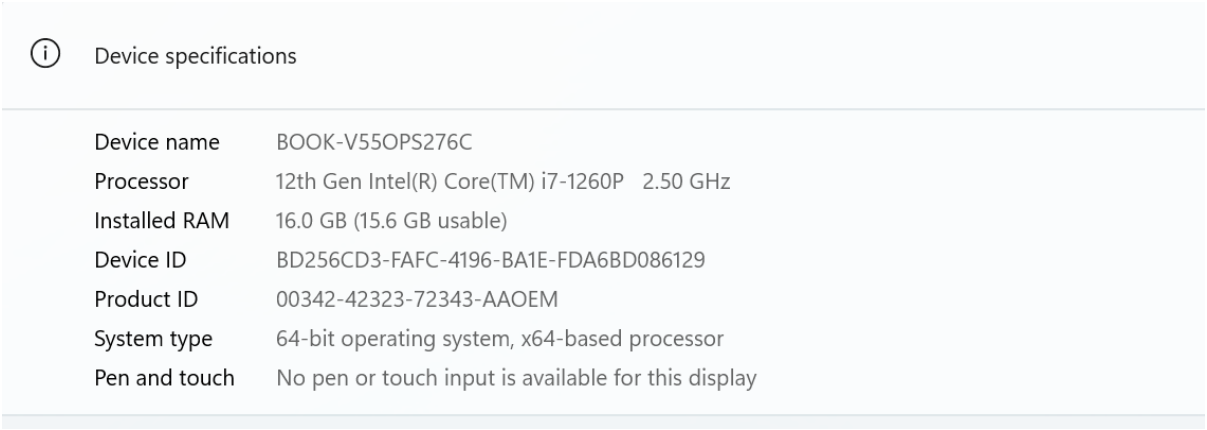
A Configuration Manual tells you in simple words how to perfectly set up and change the settings of a certain system, program or hardware. Usually it involves what hardware you need, how to put in software and the right setup. It also helps with problems and finding answers for them when they come up. Setting up a manual is good for making a system work the way you want it to by having its setup right. Moreover, these computer records are vital helpers that can aid users who want to fix problems or change how the system works.

## 2 System Configuration

The models that spot when someone is tired work largely because of how the system inside it operates. This includes physical parts and program bits. A strong system makes sure that data is handled well and studied correctly. This helps to identify quickly if someone is tired or not, in the right way with no mistakes. In this section, we outline the key elements of the system configuration employed in our evaluation:

### 2.1 Hardware Requirments:

The parts of the computer setup include what makes it go and deal with lots of information. This handles running machine learning methods on data load too. In our study, we utilized a system with the following specifications:

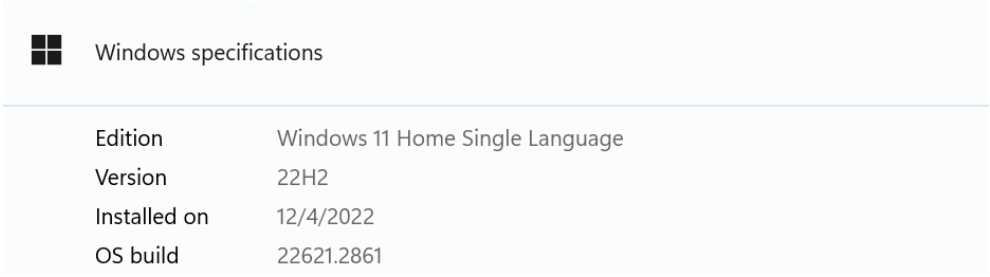



These hardware details were picked to have enough power. This makes sure the model can be trained and tested. But higher version than this hardware specification would work more efficient.

## 2.2 Software Requirments:

The software setting is also very important because it decides what tools, libraries and frameworks are used to put machine learning methods into action. The software configuration in our study included:

- Windows 11



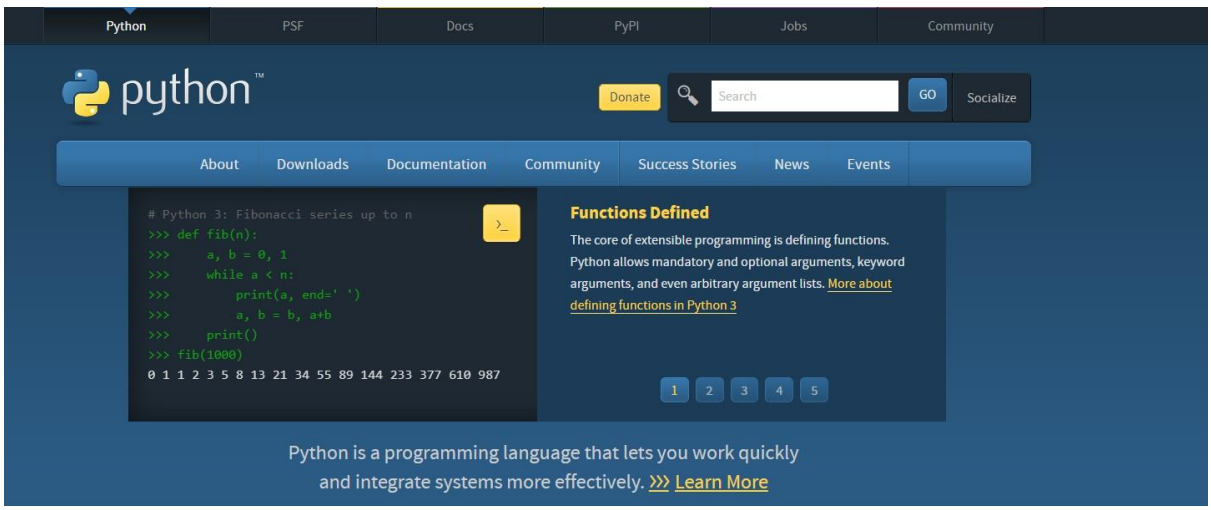

- VS Code or Pycharm
- Python (Version 3.10)

These parts of the software were chosen very carefully to make a complete and helpful place for creating fatigue detection models.

# 3 Installation and Environment Setup

- **Python**

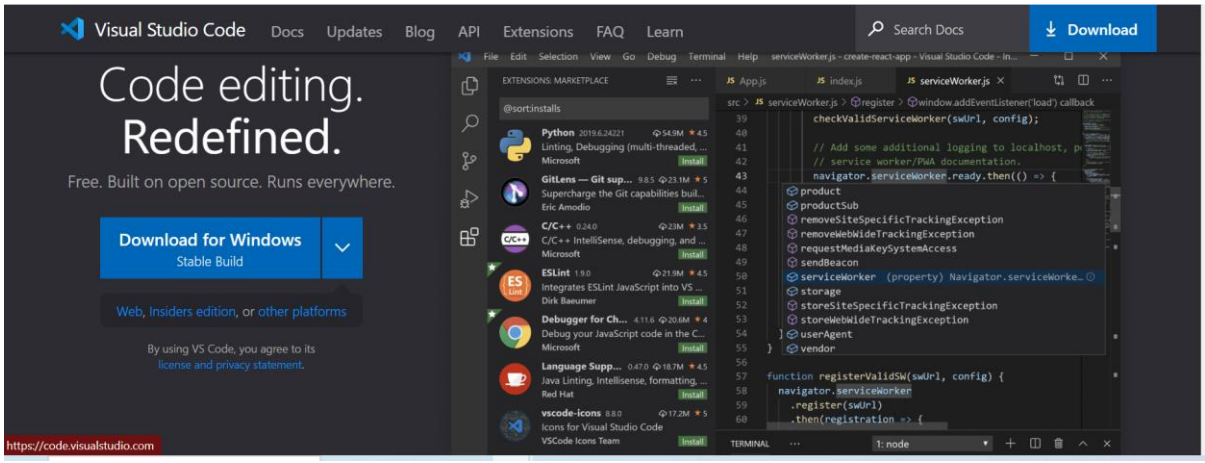
For this project, we used a python package. It has many libraries that give plenty of help for most deep learning and machine learning jobs. With many plans to pick from, it helps build and later study the models. First of all, make sure you have the latest version of python installed in your system. The package installer is capable of being downloaded through a web browser from the website reference https: Python: depending on OS; http://www.python.org/downloads/ . The picture below shows that by typing 'python -version' on the command line, you can check if python came from this website.




- **Visual Studio Code (VSCode**

Visual Studio Code, available for download at https://code.visualstudio.com/ is a helpful tool that makes it easy to code and works well with all kinds of programming languages, so you can write code smoothly wherever the website takes you. It helps a lot with different types of code and add-ons, which makes it the best pick for creating Python programs. VSCode has a

simple interface with features like coloring of codes, help for finishing them and built-in use of Git. This makes it easy to add extra features without much trouble. It lets us put in important Python libraries such as transformers, Scikit-Learn, nltk (a natural language toolkit), Numpy and Pandas for numbers stuffs. Other tools included are tensorflow used with deep learning models like artificial brains but not limited to anyone just get them from Google's platform.




To enhance Python environment in VSCode, you can use the integrated terminal to execute pip commands for library installations, such as:

**Command**: pip install 'LibraryName'

# 4 Dataset Details:

BODMAS is a collection of data about different harmful software called malware, which helps researchers find new ways to spot and understand these internet dangers. With a big group of 57,293 bad and 77,142 good files in it, this shows how malware threats keep changing all the time. Starting from August 2019 to September 2020, the dataset covers a time period of almost one year. It involves removing malware - or harmful software for short - which numbers around at least five hundred and eighty-one different kinds. The CSV file is really big because it has 2,381 finely tuned feature vectors that help figure Scientists use these qualities to watch how malware actions change over time. This improves their knowledge of growing virus families. The BODMAS list is a key tool we use for studying and improving ways to spot computer viruses. This helps us stay safe in an always changing threat world.

But we picked up only 100 benign samples and 100 malicious sample for this research.

**Dataset Link:** This Dataset was shared by the CeADAR team (Not supposed to be shared).

# 5 Implementation

## 5.1 Importing Libraries

The section about putting the project into action tells us how it was made using Python. Please do what is said step by step. Before we start using the given data, we need to prepare it first. The libraries needed for starting up are shown in the picture below.

```
import torch
import argparse
import os
import sys
from tqdm import tqdm
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
import json
import numpy as np
from src.ml.ablation_schemes.dynamic_chunk_ablations_generator import SequentialDynamicChunkAblatedEnd2End
from src.ml.classifiers.smoothed_classifier import SmoothedClassifier
from src.ml.end2end_builders.malconv_builder import MalConvBuilder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
import torch.nn as nn
import torch.optim as optim
```

## 5.2 Data Collection:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Predict Maliciousness Score')
    parser.add_argument("model_checkpoint", type=str, help="Model weights checkpoint file")
    parser.add_argument("hyperparameters_filepath", type=str, help="Path to the hyperparameters file")
    parser.add_argument("input_directory", type=str, help="Directory containing input executable files")
    parser.add_argument("output_directory", type=str, help="Directory to save the predicted scores")
    parser.add_argument("--max_len", type=int, default=1000000, help="Maximum length of input bytes")
    parser.add_argument("--padding_value", type=float, default=256.0, help="Padding value. Either 0.0 or 2
    args = parser.parse_args()
```

Reading Inputs

## 5.3 Base detector Model calling:

```
builder = MalConvBuilder()
device = torch.device("cpu")
hyperparameters = load_hyperparameters(args.hyperparameters_filepath)
model, hyperparameters = builder.build(
    args.hyperparameters_filepath,
    model_checkpoint=args.model_checkpoint,
    pretrained_emb=None,
    device=device,
    padding_idx=int(args.padding_value)
)

dynamic_chunk_generator = SequentialDynamicChunkAblatedEnd2EndGenerator(
    train_mode=False,
    file_portion=0.05,
    padding_value=args.padding_value
)

smoothed_model = SmoothedClassifier(model, dynamic_chunk_generator)
```

Calling Malconv model and SmoothedClassifier

## 5.4 Model building:

The build model part of the user guide shows how to make and change models in products or systems. This part is very important for people who want to use the setup process and change it as per their special needs.

### 5.4.1 Logistic Regression

```
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train, y_train)

# Evaluate the model on the validation set
predictions_val = logistic_regression_model.predict(X_val)
accuracy_val = accuracy_score(y_val, predictions_val)
precision_val, recall_val, f1_val, _ = precision_recall_fscore_support(y_val, predictions_val, average
print(f"Logistic Regression Model Validation Accuracy: {accuracy_val:.4f}")
print(f"Validation Precision: {precision_val:.4f}")
print(f"Validation Recall: {recall_val:.4f}")
print(f"Validation F1-Score: {f1_val:.4f}")
```

### 5.4.2 Random Forest

```
# Train Random Forest Model
random_forest_model = RandomForestClassifier()
random_forest_model.fit(X_train, y_train)

# Evaluate the Random Forest model on the validation set
predictions_val_rf = random_forest_model.predict(X_val)
accuracy_val_rf = accuracy_score(y_val, predictions_val_rf)
precision_val_rf, recall_val_rf, f1_val_rf, _ = precision_recall_fscore_support(y_val, predictions_val
print(f"Random Forest Model Validation Accuracy: {accuracy_val_rf:.4f}")
print(f"Validation Precision: {precision_val_rf:.4f}")
print(f"Validation Recall: {recall_val_rf:.4f}")
print(f"Validation F1-Score: {f1_val_rf:.4f}")
```

### 5.4.3 Decision Tress

```
# Train Decision Tree Model
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)

# Evaluate the Decision Tree model on the validation set
predictions_val_dt = decision_tree_model.predict(X_val)
accuracy_val_dt = accuracy_score(y_val, predictions_val_dt)
precision_val_dt, recall_val_dt, f1_val_dt, _ = precision_recall_fscore_support(y_val, predictions_val
print(f"Decision Tree Model Validation Accuracy: {accuracy_val_dt:.4f}")
print(f"Validation Precision: {precision_val_dt:.4f}")
print(f"Validation Recall: {recall_val_dt:.4f}")
print(f"Validation F1-Score: {f1_val_dt:.4f}")
```

### 5.4.4 LSTM

```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
```

Lstm model Intalization

```python
if classify_maliciousness(outputs) == "Goodware":
    # Perform LSTM process
    input_tensor = input_tensor.unsqueeze(0).unsqueeze(0).float()
    lstm_model = LSTMModel(input_size=input_tensor.size(-1), hidden_size=128, output_size=1)
    lstm_model.to(device)
    lstm_model.train()

    criterion = nn.MSELoss()
    optimizer = optim.Adam(lstm_model.parameters(), lr=0.001)

    # Train the LSTM model
    num_epochs=1
    for _ in range(num_epochs):
        optimizer.zero_grad()

        outputs = lstm_model(input_tensor)
        loss = criterion(outputs, torch.tensor([[0.0]], dtype=torch.float32, device=device))

        loss.backward()
        optimizer.step()

    # Get the final output from the LSTM model
    outputs = lstm_model(input_tensor).squeeze().detach().cpu().numpy()
```

LSTM Model calling and Traning

# 6 Evaluation

## 6.1 Machine Learning:

```
Processing Malicious Test Examples: 100%| 10/10 [00:15<00:00,  1.53s/it]
Logistic Regression Model Validation Accuracy: 0.7000
Validation Precision: 0.6429
Validation Recall: 0.9000
Validation F1-Score: 0.7500
Logistic Regression Model Test Accuracy: 0.7000
Test Precision: 0.6429
Test Recall: 0.9000
Test F1-Score: 0.7500
Random Forest Model Validation Accuracy: 0.7000
Validation Precision: 0.6429
Validation Recall: 0.9000
Validation F1-Score: 0.7500
Random Forest Model Test Accuracy: 0.6500
Test Precision: 0.6154
Test Recall: 0.8000
Test F1-Score: 0.6957
Decision Tree Model Validation Accuracy: 0.7000
Validation Precision: 0.6429
Validation Recall: 0.9000
Validation F1-Score: 0.7500
Decision Tree Model Test Accuracy: 0.6000
Test Precision: 0.5833
Test Recall: 0.7000
Test F1-Score: 0.6364
```