Feasibility of Malware Packer Detection

Using Hardware Performance Counters

Final Report

Authors:

Erika Leal
Abhishek Shinde

This report highlights all the documentation, project plan, foreseeable risks, achieved goals, competitors to our application and risk mitigation.

Table of Content

LZMA & UCL Hardware Event Classifier Tool

Project Plan	**1**
List of Features	**2**
Key Data Structures	**3**
List of 5 Biggest Risks	**3**
New Risks	**3**
Plans to Deal with Risk	**4**
Specification and Design	**5**
Competing Applications	**5**
Graphical User Interface	**5**
Using our App	**6**
Control Flow Diagram	**6**
Use Case Diagram	**7**
Screenshot of Flow Graph	**7**
Test Cases	**10**
Code and Tests	**10**
Customer & User Feedback	**16**
About the Customer & User	**17**
Github Repository Link	**17**
References	**17**
Presentation Slides	**17**

Project Plan

Iterations	**Goals**	**Achieved Goals**
1	• Roll out version 1 of GUI • Decide how many test cases we would like.	• GUI Version 1.0 designed and ready • Decided on 50 Test Cases for now.
2	• Collect hardware performance counters of programs packed with LZMA • Train Machine learning classifiers for LZMA • Implement LZMA into front end • Improve User Experience v1.1	• Collected Hardware events for UPX • Trained classifier for LZMA
3	• Collect hardware performance counters of programs packed with UPX • Train Machine learning classifiers for UPX • Implement UPX into front end • Improve User Experience v1.2	• Collected more events for LZMA • Collected events for UPX • Improved User Experience • Backend connection to front end
4	• Train Machine learning classifiers for neither • Roll out version 2 of GUI	• Collected more events for LZMA • Collected more events for UPX • Collected more events for the case of “Neither” • Trained new model with 3 classes • Connected new model to back end

List of Features

Feature	Qualitative value to Customer
Can accurately identify UPX	High
Can accurately identify LZMA	High
Can accurately identify when its 'Neither' of the two algorithms	High
Can accurately identify between UPX & LZMA & when its neither.	High
Easy to use GUI	Medium
Uses Arrays & Lists to filter data in program called Extractor	Medium
Users can learn about the application and about the different concepts used in the application by accessing the ReadMe File	Medium
Uses a Python front end and back end	Low

Key Data Structures

1. Arrays -- used in Extractor & Classifier code which is available on the Github
2. Lists -- used in Extractor & Classifier code which is available on the Github
3. Dataframe -- used in Classifier code which is available on the Github

List of 5 Biggest Risks

1. Chance of exposing computer to malware
    a. P = 40% and E=8, so 3.2 extra hours
2. Machine learning classifier cannot accurately decipher algorithms of itself or between others
    a. P = 30% and E = 20, 6 extra hours
3. Not achieving iteration goals
    a. P=30% and E = 15, so extra 4.5 hours
4. Process filtering is not turned on leading to inaccurate results
    a. P = 30% and E= 10, so 3 extra hours
5. Final program is not portable enough
    a. P = 10% and E = 5, so 0.5 extra hours
6. Risk of using a tool which has been recently updated, forcing us to create a new script for the retrieval of values of HPCs.

New Risks

1. Computer Failure
    a. P = 40% and E=10, so 4 extra hours
2. Unexpected Software Update
    a. P = 70% and E =24, 16.8 extra hours
3. Working Executable malfunction
    a. P = 10% and E = 4, 0.4 extra hours
4. Automation Script Failure
    a. P=40% and E = 20, so extra 8 hours

Plans to Deal with Risk

1. **Dealing with Risk 1:** In order to prevent our systems being corrupted by malware, we will first test classification on clean programs before we move to malicious ones. When handling the malicious programs, we will be sure to use a black box environment as well as not being connected to the network and having DeepFreeze on the system which will rebuild the OS every day.

2. **Dealing with Risk 2:** A crucial high risk is the risk that our machine learning classifiers will find no relation between hardware performance counters and not correctly classify the packing algorithms. This risk can be alleviated by having enough test cases when classifying an algorithm on itself. We will first measure the HPCs of a pure algorithm and compare those to a packed program and collect data

for many many programs to ensure that our machine classifier gets trained well enough.

3. **Dealing with Risk 3:** One possible risk is not achieving our iteration goals. The plan to reach these goals is to have a meeting after each class period where we will either work together or divide the work equally and separately.

4. **Dealing with Risk 4:** Another important risk is collecting hardware performance counters from other processes but this can be filtered using the IntelVtunes tool and also perf which are the softwares being used to measure HPCs.

5. **Dealing with Risk 5:**The risk of our program not being portable enough can be helped by ensuring clear documentation in download and compiling as well as testing it on different platforms.

6. **Dealing with Risk 6:** This risk can be mitigated by creating a new python script which will work by itself to retrieve the values from the intel vtunes tool.

7. **Dealing with New Risk 1:** In order to mitigate this risk, we will have another testing environment set up and regular backups will be implemented.

8. **Dealing with New Risk 2:** There is no way to tell when a software will be updated. We will have to stay up to date with IntelVtunes and how often the software comes out with a new version.
9. **Dealing with New Risk 3**: We now need to have multiple working executables.

10. **Dealing with New Risk 4**: We encountered an issue in our test automation script which we use to gather data for multiple test runs. We did not anticipate the updation of the hardware performance collector application which caused our existing scripts to break. For this iteration, we overcome this risk by manually running test cases and gathering data.

Specification and Design

Our Project is based on analysing the Hardware Performance Counters of popular packing algorithms in order to classify families of packed algorithms in obfuscated programs. By doing so, we hope to improve malware analysis techniques. We first retrieve these values by using a tool that measures hardware performance counters like **perf** or **intelVtunes**. We will measure all hardware performance counters of a packed program. We will then process and filter out the data so that we are only accurately measuring data from the program itself. Then we will use our tool to compare the packed program's Hardware Performance Counter

values with the pure algorithm's values and identify only those events that have values between all test runs. After that, using our machine learning classifier with the best accuracy, we will be able to tell what the program is packed with. LZMA or UPX or if the packed malware is using neither of these two algorithms.

Competing Applications

1. PEiD was a tool that could detect most common packers but the tool became discontinued [1]. It worked by examining software signatures.
2. RDG Packer Detector [2] is a detector for packers, cryptors, compilers, scramblers, joiners, installers. It is based on software signatures and entropy analysis.
3. Windows Executable Packer Detection[3] is also software signature based and we could not get this one to work.

Graphical User Interface

The objective that our User Interface will accomplish is identifying what Packing Algorithm is being used on a program by uploading an excel file and at a click of the button, know what packing algorithm (either UPX or LZMA or neither of the two for now) is used.

Expected Input for Application: An Excel sheet generated from the tool IntelVtunes or Perf , or any other Excel file containing all the Hardware Performance Counter values for the events triggered during the unpacking process.

Expected Output: Output displayed in the output window of our application identifying accurately, the packing algorithm used..

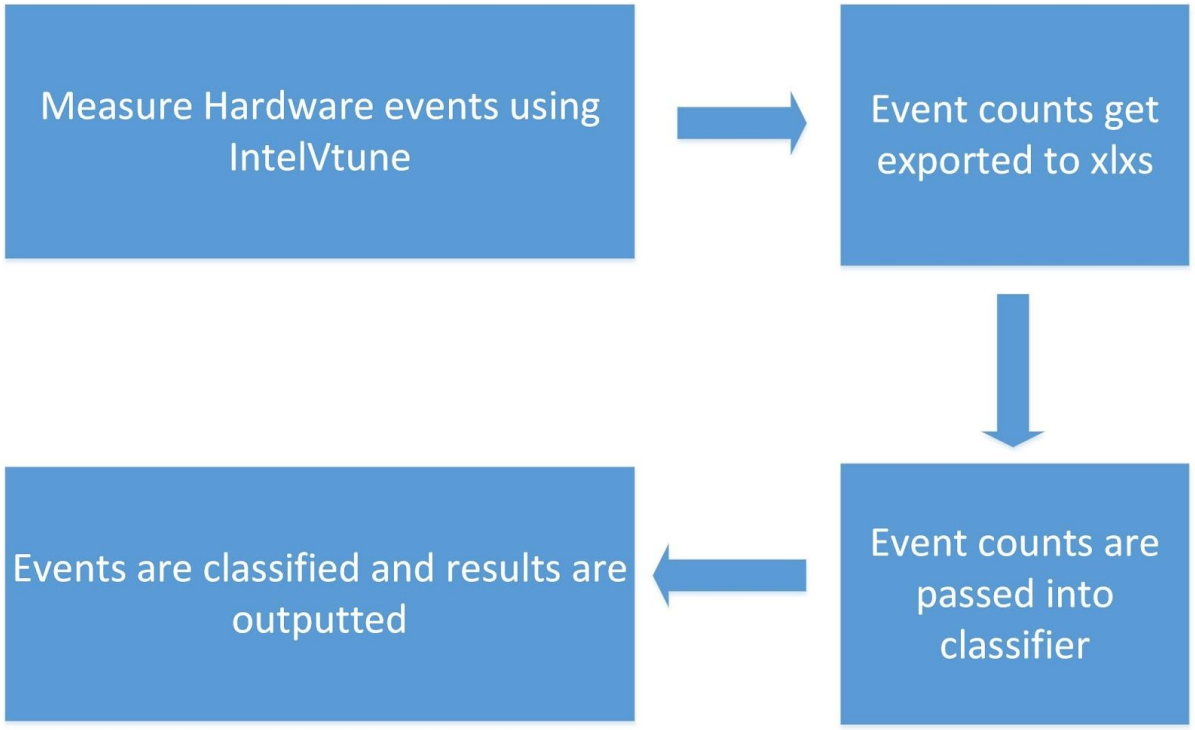
If the user uploads a file which is not an excel file, the program will not continue forward, and a message will display the user that the file is not in XLSX format. The user will be prompted to Upload an excel file for using our application to identify the algorithm.

Using our App

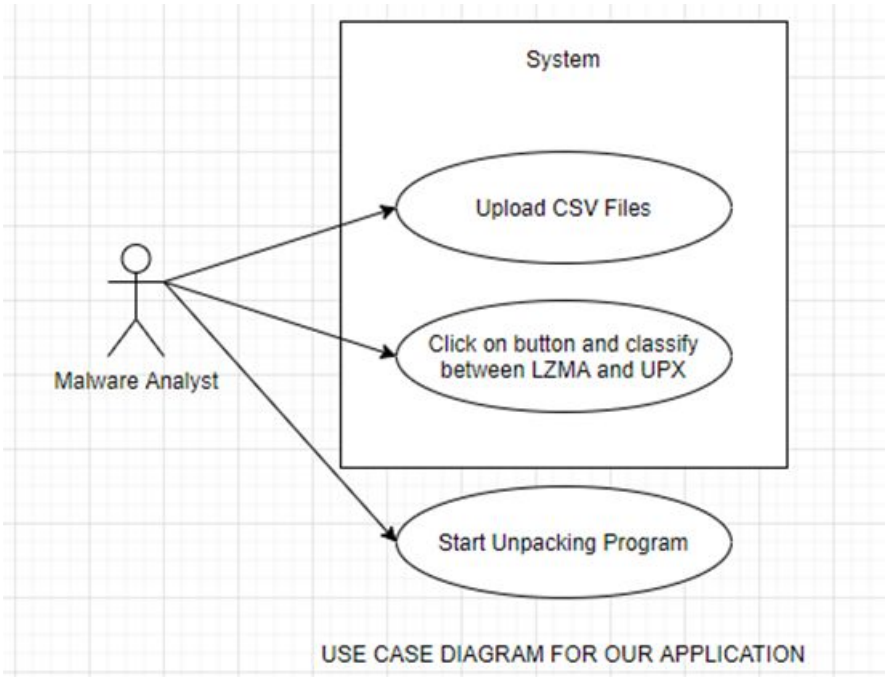
To use our application, the user must first run the program, which will display the UI of the program to the user using Python.
We assume here that the user has already used a tool for generating the Hardware Performance Counter values for the packed malware affecting their system. This generated data will be in an excel file.

Once the user has the generated data ready, they must click on the “Select Excel File” button which will open a file dialog box prompting them to select a file. They should select the Excel file which they have obtained. On selecting the Excel file, the program will check if the file is of accurate type and format, for safety measure. If the file is not in XLSX format, the program will not be able to run the classification model. When the file is identified to be of the accurate format, the user must click on “Run Classifier” which will use the Excel file selected as input and run its classification algorithm. When the program finishes execution, it will return the Packing Algorithm used as output in a text message displayed to the user in the UI.

Control Flow Diagram



Use Case Diagram



Screenshot of Flow Graph:

Fig. 1 - Shows the Program UI which the user will see when the user opens the application.

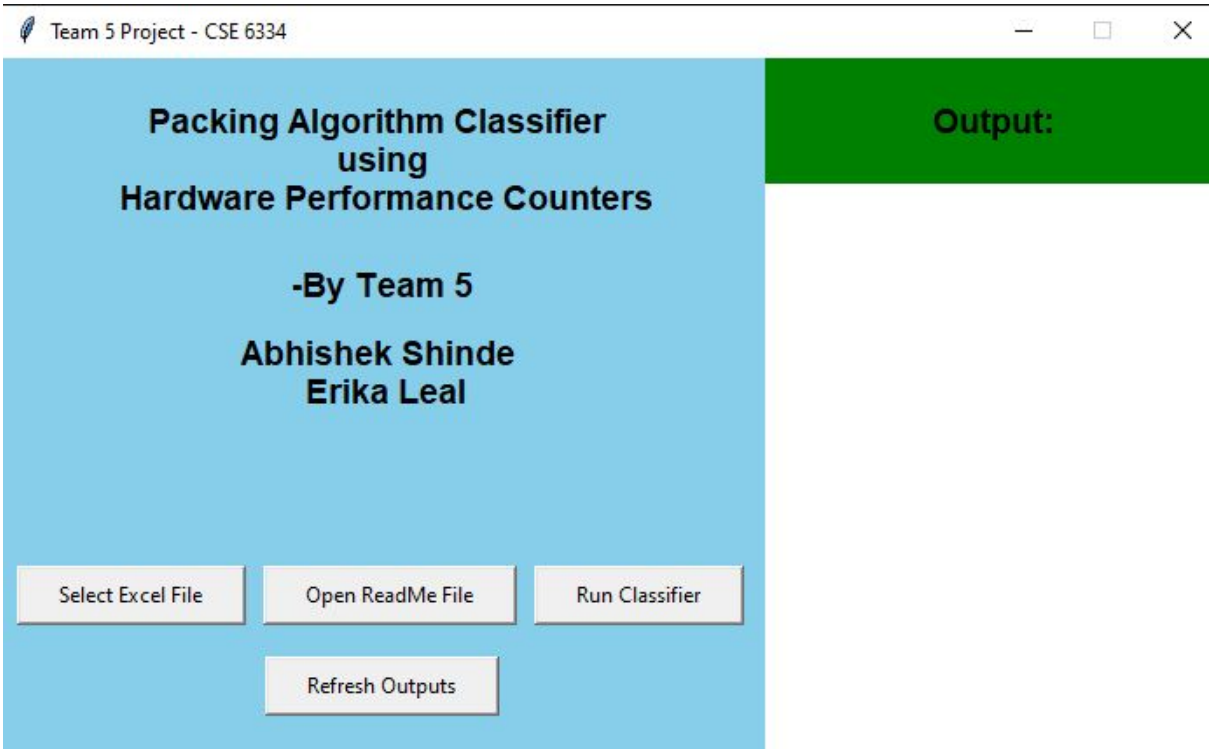


Fig. 2 - When the User Clicks the "Select Excel File" Button, the select file dialog box will open prompting the user to select the required file.

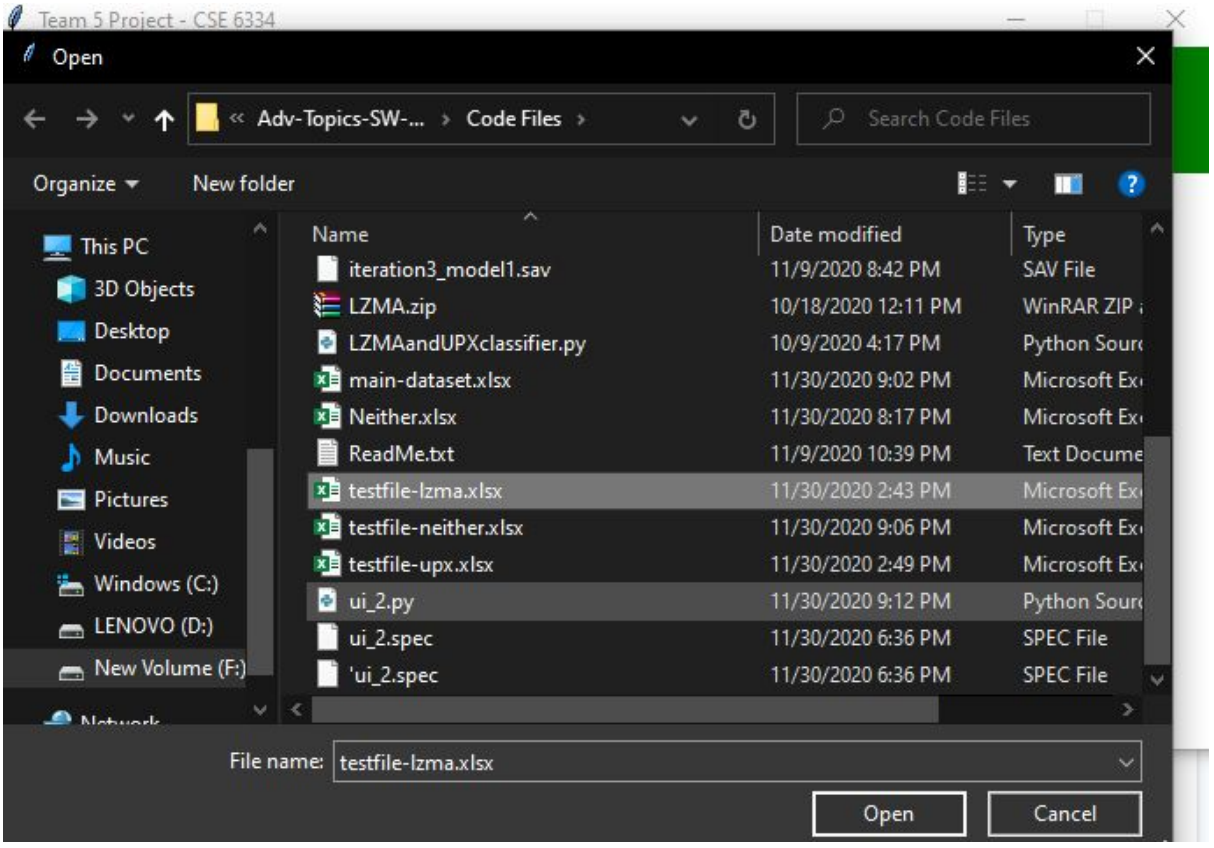


Fig. 3 - When the User selects the correct file with the .xlsx extension, the application will check if the file is of the correct and desired format. After checking, it will display a message for the user which will say "Entered File is of the correct format.".

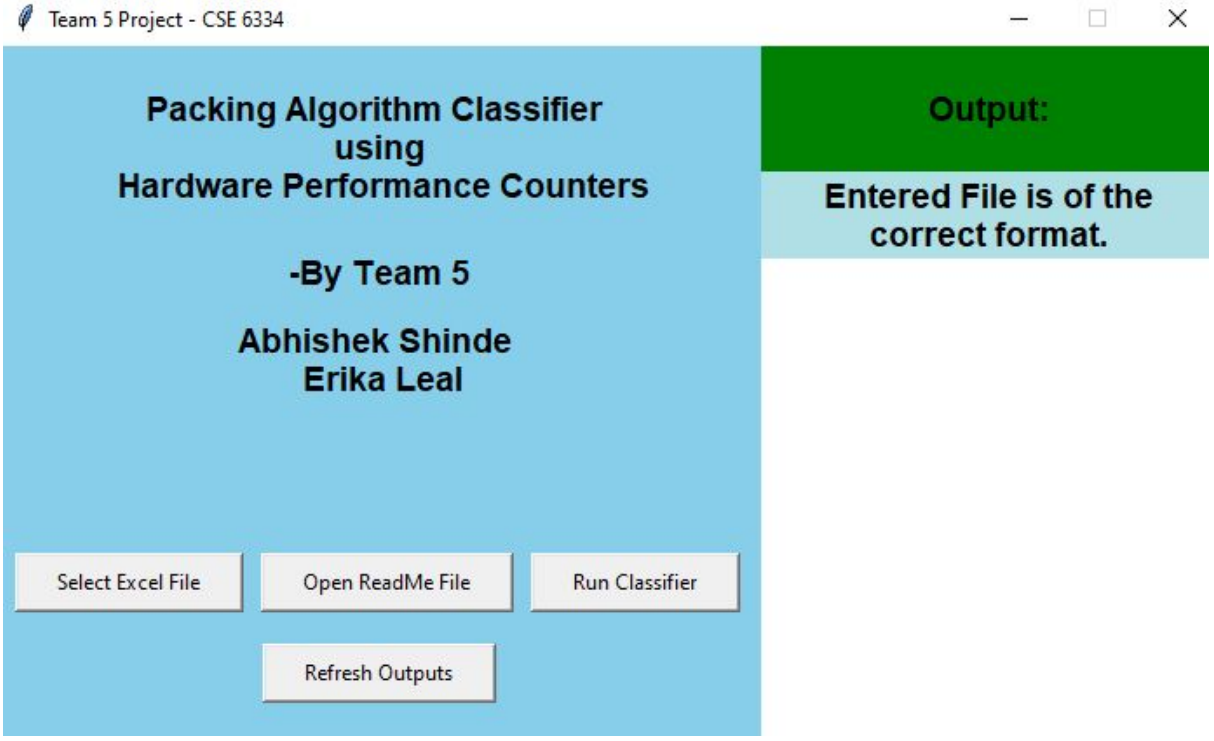


Fig. 4 - After verifying that the file is correct and is accepted, the user must click on the "Run Classifier" button which will start the classification process to identify which packing algorithm is being used. After the processing, the program will display the output with a message which says "The identified packing algorithm is : ####" where #### will be the output which is either LZMA or UPX or Neither.

Fig. 5 - If there is no file selected or if the file is not of proper format, and if the user still clicks on the "Run Classifier" button, the program will display a "Please upload a file" message.

Test Cases

Test Case	**Application Function**
Case 1: Incorrect File Format	The Application Displays an “Incorrect File Format” message.
Case 2: Clicking on Run Classifier Button without uploading a File.	The Application will ask the user to upload a file.

Code and Tests:

Extractor - It is the code responsible for helping identify HPC values that are similar between the pure compression algorithm and the packed program.

Extractor Code:

```
# in case you run into any errors try doing this first:
# pip install xlrd

from pathlib import Path

import pandas as pd
from pandas import DataFrame

#main function that checks if a folder output exists or not and then begins execution
#finds all excel files in the specified folder
def _main( folder_path: str, output_file: str ) :
        folder = Path( folder_path )
        if not folder.is_dir() :
        raise Exception( f'{folder_path} does not exist' )

        files = [x for x in folder.iterdir() if x.is_file() and x.suffix == '.xlsx']
        data = {}

        for file in files :
        df = pd.read_excel( file )
        start = _get_start_row( df )
```

```
        file_data = _extract_data( df, start )
        _merge_data( data, file_data )

        _dump_data( output_file, data )

#dump the name of the value and value
def _dump_data( filename: str, data: dict ) :
        with open( filename, 'w' ) as fp :
        for k, v in data.items() :
        row = f'{k}, {", ".join( [str( x ) for x in v] )}'
        print( row, file=fp )

#merge all dictionaries into a single dictionary
def _merge_data( data: dict, file_data: dict ) :
        for k, v in file_data.items() :
        if v == 0 :
        continue
        if k not in data.keys() :
        data[k] = []
        data[k].append( v )

#extract data from rows using pandas and return it
def _extract_data( df: DataFrame, start: int ) -> dict :
        data = {}
        required = df.iloc[start :]
        for r in range( required.shape[0] ) :
        row = required.iloc[r]
        data[row[0].strip()] = row[1]
        return data


#returns dimensions
#gets hardware event type and hardware event count
def _get_start_row( df: DataFrame ) -> int :
        for row in range( df.shape[0] ) :
        if _cell_equals( df.iat[row, 0], 'Hardware Event type' ) and _cell_equals( df.iat[row, 1],
'Hardware Event count' ) :
        return row + 1
        return -1


def _cell_equals( cell, value: str ) -> bool :
        if not isinstance( cell, str ) :
        return False
        if cell.lower().startswith( value.lower() ) :
        return True
        return False
```

```
if __name__ == '__main__' :
        _main( r'path to files', r'parth to where the file will output\output.csv' )
```

UI Code :

```
#Importing Tkinter Standard GUI Package
from tkinter import *
import tkinter as tk
#Importing the Tkinter Fonts Module for allowing the use of Multiple
Font Styles
import tkinter.font as tkFont
#Importing the FileDialog Module allowing for open and save operations
of Files in Tkinter
from tkinter import filedialog
import joblib
import pandas


#We create a new window instance of Tkinter using the statement below.
It creates the main window (or box) of our GUI
window = tk.Tk()
#We create the title for our GUI Window using the .title() function
window.title("Team 5 Project - CSE 6334")
#Using the .geometry() function, we can define the dimensions of our
GUI Window.
window.geometry("700x400")
window.resizable(0,0)

global filepath
filepath = ""
global filetype
filetype = 0


lframe = Frame(window, bg="skyblue", width="450")
lframe.pack(anchor=W, fill=Y, expand=False, side="left")

rframe = Frame(window, bg="orange",width="250")
rframe.pack(anchor=N, fill=BOTH, expand=True, side="left")

#Here we have defined a function which is called when the user clicks
on the "Select CSV File" Button.
```

```
#We also check if the file selected by the user is a CSV file or not.
#If the file is not of CSV Format, the program will display a "File is
not a CSV File" message on the UI.
#filedialog.askopenfilename() allows us to interact with File
Operations using the FileDialog Module making it easier and simpler to
code with.



def SelectCSV(event=None):
    global filepath
    global filetype
    optext = ""
    selectfile = filedialog.askopenfilename()
    filepath = selectfile
    if(selectfile.endswith('.xlsx')):
            optext="Entered File is of the correct format."
            filetype = 1 #if 1 - file is a csv file
    else:
            optext="File format not supported. Please Enter a .xlsx
file."
            filetype = 0 #if - file is not a csv file

    fileoutput = tk.Label(outframe,text=optext,
    background="powder blue",
    font=('comic sans',14,'bold'),
    wraplength=250,
    width=36)
    fileoutput.pack(fill=Y)


#NEW FUNCITON
def buttonClick():
        global filepath
        global filetype

        if ((filetype == 0) or (filepath==("")):
                classifierOP = tk.Label(outframe,text="Please Upload a
File",background="pink",font=('comic sans',12,'bold'),
                wraplength=500,width=36)
                classifierOP.pack(fill=Y)
        else :
                filename = 'final_deliverable_model.sav'
```

```
                loaded_model = joblib.load(filename)
                outputfile = pandas.read_excel(r'%s'
%filepath,header=None,sheet_name="Sheet1")
                result = loaded_model.predict(outputfile)
                classifierOP = tk.Label(outframe,text="The identified
packing algorithm is :"+result[0],background="pink",font=('comic
sans',10,'bold'),
                wraplength=250,width=36)
                classifierOP.pack(fill=Y)


def ReadMe():
    from os import startfile
    startfile("ReadMe.txt")

def refresh():
        global filepath
        filepath =""
        for widget in outframe.winfo_children():
                widget.destroy()

#Using the .Label() function, we are adding the title of our project as
Text
# and displaying it on the window , by setting different attributes
along with it.
greeting = tk.Label(lframe,text="Packing Algorithm Classifier \n using
\n Hardware Performance Counters",
        background="skyblue",
        font=('comic sans',14,'bold'),
        height=5,
        wraplength=500,
        width=36)
greeting.pack()

team5 = tk.Label(lframe,text="-By Team 5",background="skyblue",
        font=('comic sans',14,'bold'),
        height=0,
        wraplength=500,
        width=36)
team5.pack()

members = tk.Label(lframe,text="Abhishek Shinde \n Erika
Leal",background="skyblue",
```

```
        font=('comic sans',14,"bold"),
        height=3,
        wraplength=500,
        width=36)
members.pack()



selectbutton = tk.Button(lframe, text='Select Excel
File',command=SelectCSV,padx="20",pady="5")
selectbutton.pack(padx=(10,0),side="left")

howto = tk.Button(lframe, text='Open ReadMe File',
command=ReadMe,padx="20",pady="5")
howto.pack(padx=(10,0),side="left")

classifybutton = tk.Button(lframe,text='Run
Classifier',command=buttonClick, padx="20",pady="5")
classifybutton.pack(padx=(10,0),side="left")

refresh = tk.Button(lframe,text='Refresh Outputs',command=refresh,
padx="20",pady="5")
refresh.place(relx=0.5, rely=0.9, anchor=CENTER)



outwindow = tk.Label(rframe,text="Output:",
        background="green",
        font=('comic sans',14,'bold'),
        wraplength=500,
        width=36,
        height=3)
outwindow.pack()

outframe = Frame(rframe,bg="white",width="250")
outframe.pack(fill=BOTH , expand=True)



window.mainloop()
```

Customer & User Feedback

Members of the cyber security club were shown screenshots of the GUI as well as a small explanation. They thought it was a very neat idea and would like to test it when we have more to show them.

- Cyber Security Club Feedback:
  - Excited to see the product, liked the “Read Me” section
  - Asked about helping for the future

- Classmates Feedback:
  - Liked the updated UI and the “ReadMe” section

About the Customer & User

Dr. Jiang Ming focuses on malware analysis and developing security analysis techniques for finding vulnerabilities in software.

The Cybersecurity club is a non-profit organization based at UTA that focuses on all elements of cybersecurity and its members have varying levels of expertise.

Github Repository Link:

https://github.com/AbhishekShindeUTA/CSE-6324--Malware-Packer-Detection-using-Hardware-Performance-Counters

References

[1] https://www.aldeid.com/wiki/PEiD
[2]http://www.rdgsoft.net/
[3]https://www.guidancesoftware.com/app/Windows-Executable-Packer-Detection

Presentation Slides

https://docs.google.com/presentation/d/1bJtL7iE3sJpw2ZLS_BEoTwe1H1DFRDdqxc5bRhGGzm4/edit?usp=sharing