**Malware and Benign Software Classification using Machine Learning:**

**A Report of the work carried out as a Summer Intern**

**Submitted by**

**Satvik Ganesh**

B.Tech (CSE) 2nd Year Student at University of Aizu, Japan



For the work carried out during the

Internship at C3i Hub at Indian Institute of Technology Kanpur, India

(August 15 to September 15, 2023)



Mentor: Prof. Manindra Agrawal

Student Mentor: Mr. Venkata Sai Charan Putrevu

## **Objective of the work:**

Get data from various malware and benign software and predict whether a given sample is a malware or a benign software using a machine learning classifier.

## **Basics of MITRE ATT&CK:**

MITRE ATT&CK1 is an online repository which contains information about adversary tactics and procedures using real-world observations. Displayed in matrices, they show attack stages arranged from initial system access to data theft or machine control. There are matrices for popular desktop platforms such as Linux, macOS, and Windows, as well as technologies such as cloud, containers, networking, ICS, and mobile platforms.

ATT&CK stands for adversarial tactics, techniques, and common knowledge. The first "T" in ATT&CK stands for **tactics** which are the goals that the adversaries try to achieve in their attacks. They are the "why" of an attack, for instance gaining initial access, executing malicious code, or stealing data.

Each tactic comes with a set of **techniques**, which are the "how" of an attack. This is the second "T" in ATT&CK. For instance, if the tactic is Initial Access - a tactic where the adversary is trying to get into an individual's network - the techniques will be the many ways by which attackers will try to carry out the initial access.

The "CK" at the end of ATT&CK stands for **common knowledge**. This refers to adversaries' documented usage of tactics and procedures. Common knowledge is essentially a written record of **procedures**.

Studying MITRE ATT&CKs provides insight into an attacker's methods for obtaining vital information from an organization.

## **Setting up a virtual machine:**

For analysing malware, you need to either read its source code (static analysis) or see it running in real-time inside a virtual environment that does not harm the main computer system (dynamic analysis).

By using virtual machines one can run malware easily without worrying about their system getting hacked. In simple terms, a virtual machine is a program that creates an isolated environment within a computer, working as a separate computer with the same functionality as the host.

Virtual machines are operated using software called hypervisor which emulates a computer's hardware environment. Hypervisors isolate the main operating system and the computing resources from the virtual machines.

During my internship, I used Oracle VM VirtualBox2 to conduct my malware analysis. It is a free, open-source, cross-platform virtualization software. Using this virtual machine, I ran Linux inside Windows 10.

## **Cuckoo for malware diagnostics:**

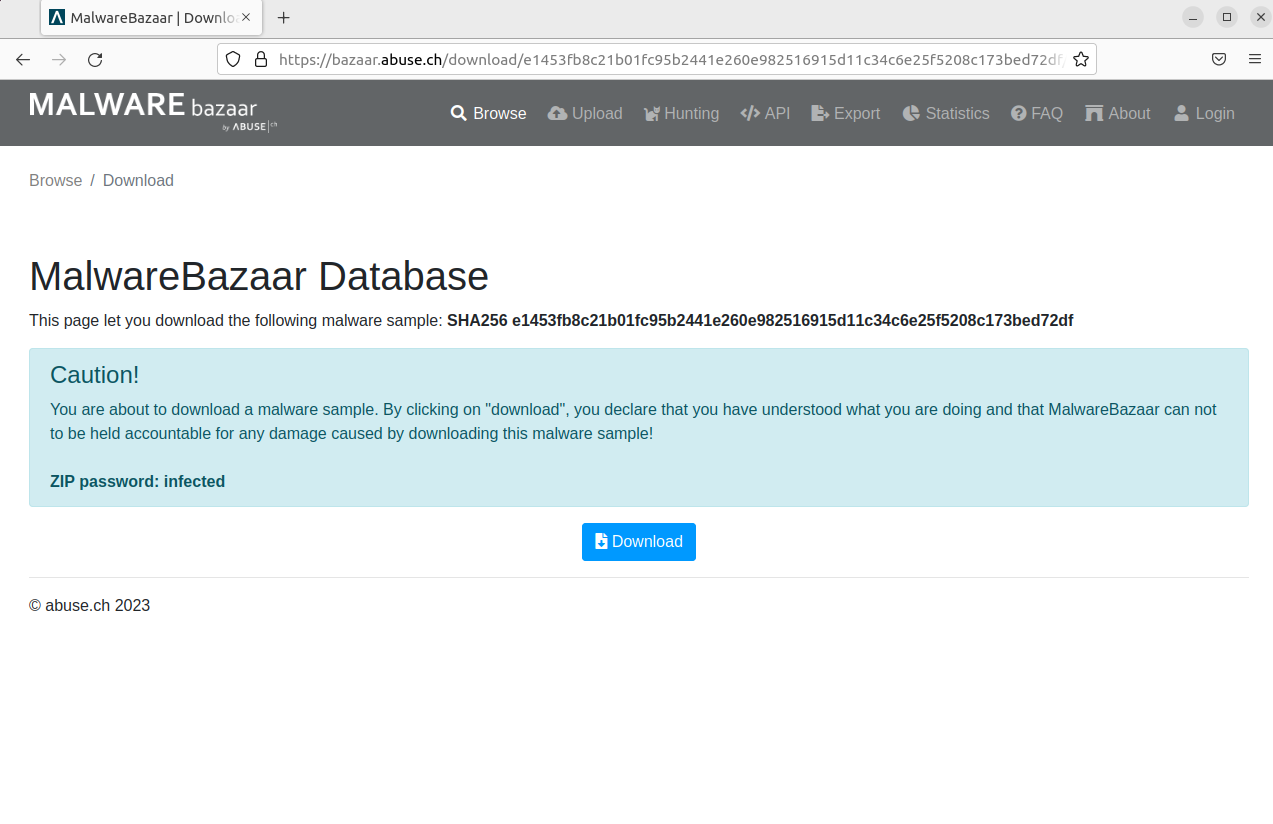
To safely analyze malware in real-time through dynamic analysis, you require software that performs the task for you.

For my case, I used Cuckoo Sandbox3 which is an open-source automated malware analysis system. Suspicious files can be quickly scanned and analysed, with a detailed report depicting file behaviour in a realistic but isolated environment.

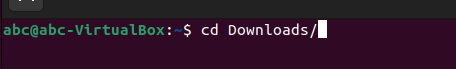
I installed this software in the Linux virtual machine to use the Terminal for control, as it runs smoothly on this OS.

Here is how to analyse the malware samples:

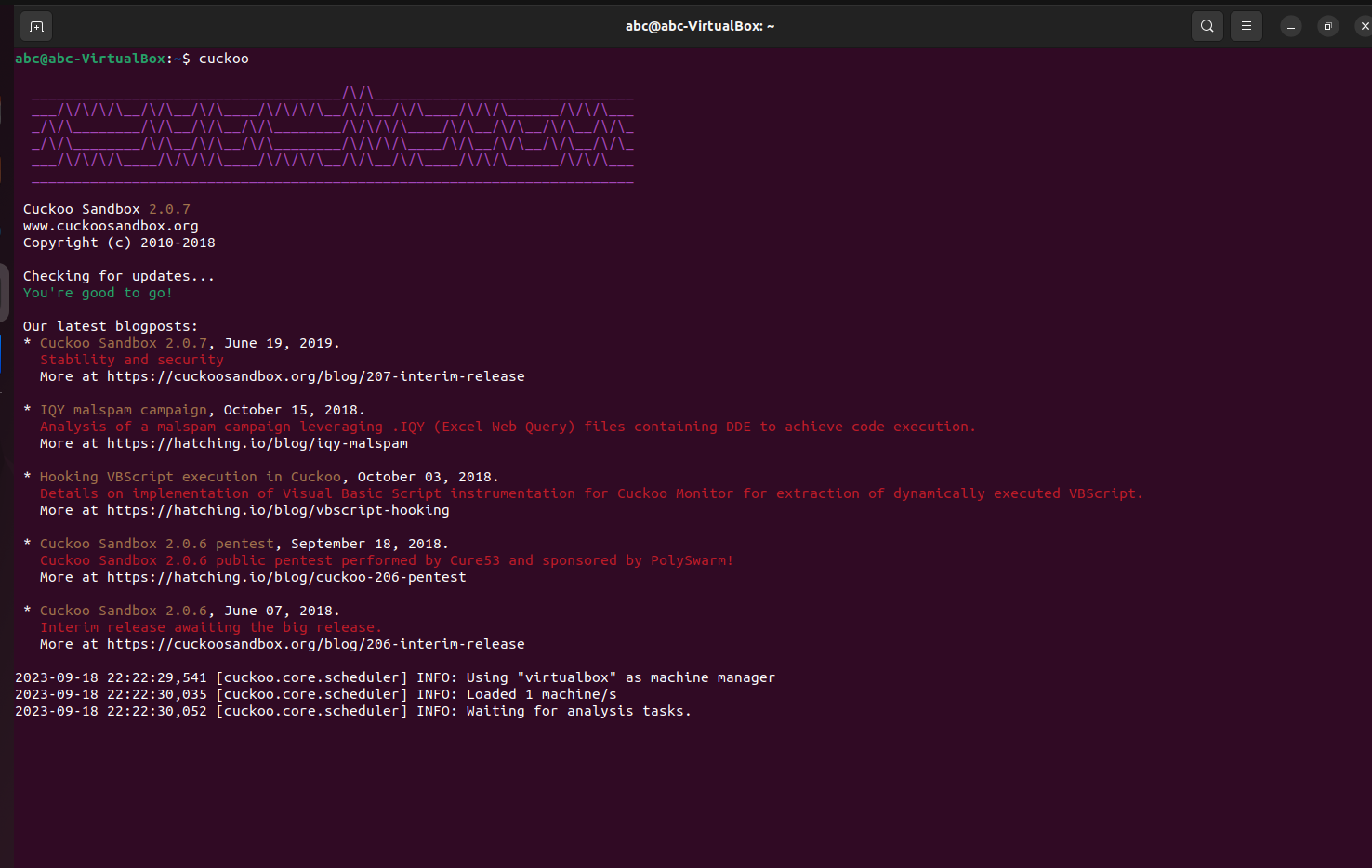
1. Inside the virtual machine, download a malware sample that you want to test. In my case, I downloaded a sample from MalwareBazaar4.



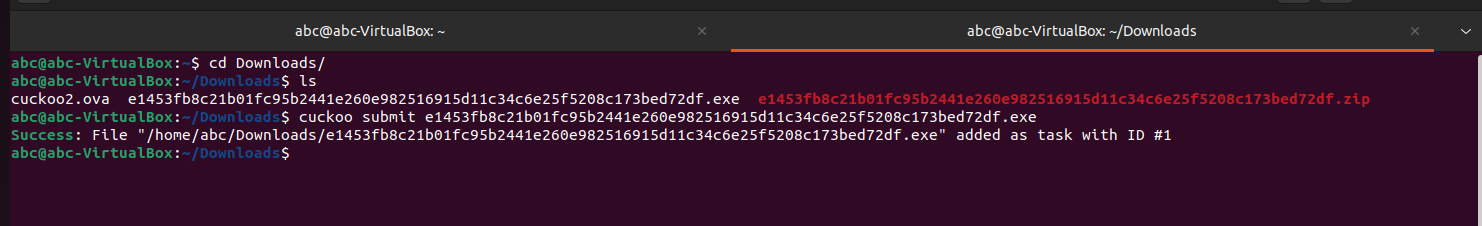
1. Using the Terminal navigate to the directory where the malware was downloaded (which is the "Downloads/" folder) and unzip the file.



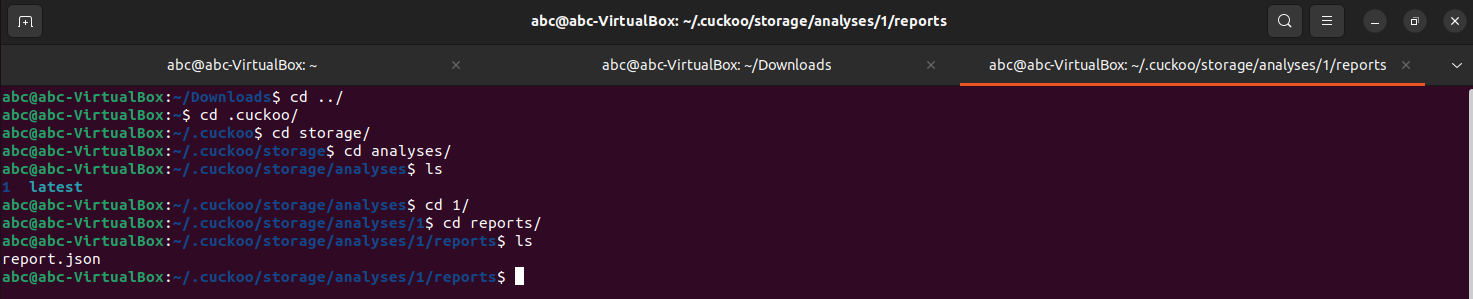
1. Then in the Terminal enter "cuckoo" to start the application.



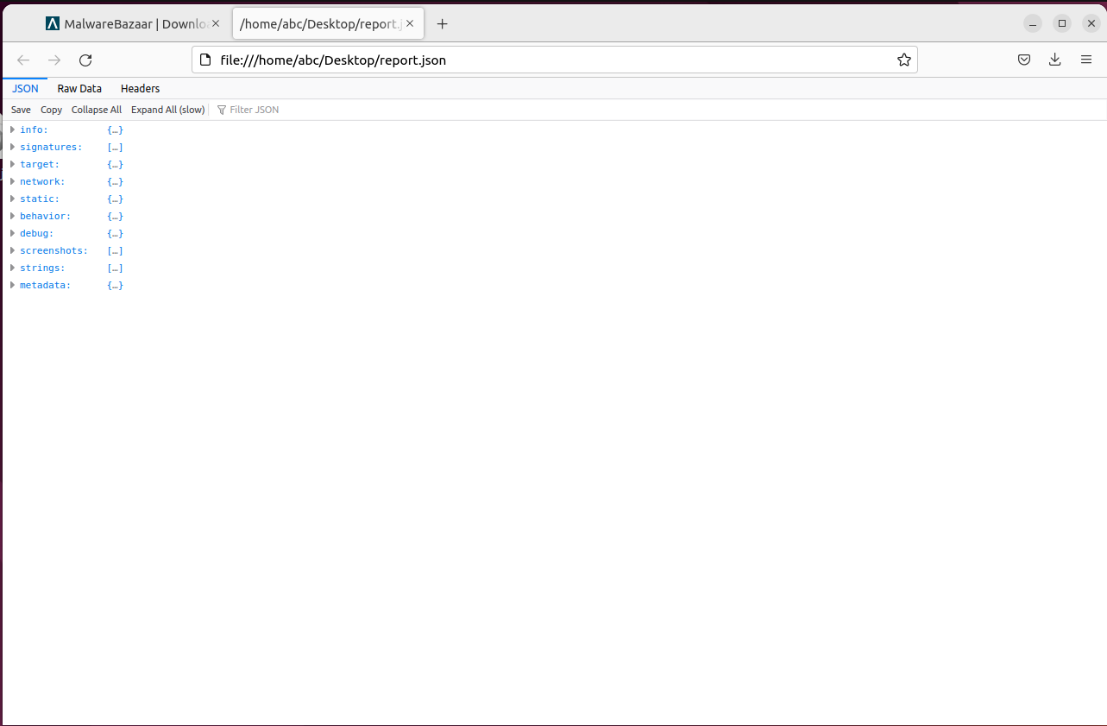
1. Then enter the command "cuckoo submit [malware].exe" to run the analysis. It generates a task ID (which is a number) of that malware and creates a folder named after that ID.



1. When the analysis is done navigate to the folder where the report of the malware is generated by typing: cd .cuckoo/ > cd storage/ > cd analyses/ > cd [task ID]/ > cd reports/. The report directory has the report.json file that contains the behavioural statistics of the submitted malware.



1. Then you can open the report.json file via Firefox to check the malware report.



## **Prompting Bing Chat to create a Python script to automate the submission of malware and benign files to Cuckoo:**

During the internship, my supervisors gave me a huge dataset of malware and benign files (500 each) in folders called "malwares" and "benigns" respectively. All those files must be run in Cuckoo. They suggested I create a code in Python that can automate the process of submitting the programs to Cuckoo. At the time, I did not know how to code in Python, so I turned to Bing Chat for creating the required code.

I used the following prompt: "Write a Python script that iterates over all the files in a directory called "malwares" and submit each file to Cuckoo for analysis. Do error handling to avoid errors."

It gave the following code:

import os

import subprocess

# Path where the malware samples are located

malware\_path = "/home/abc/Documents/satvikdataset/malwares"

# Check if Cuckoo is running

try:

    subprocess.check\_call(["pgrep", "-f", "cuckoo"])

except subprocess.CalledProcessError:

    print("Cuckoo is not running. Please start Cuckoo before running this script.")

    exit()

# List all files in the directory

malware\_files = os.listdir(malware\_path)

# Counter for submitted files

submitted\_files = 0

# Loop through each file and submit it to Cuckoo for analysis

for malware in malware\_files:

    # Construct the full path of the file

    file\_path = os.path.join(malware\_path, malware)

    # Check if the file exists and is a file

    if os.path.isfile(file\_path):

        try:

            # Submit the file to Cuckoo for analysis

            subprocess.check\_call(["cuckoo", "submit", file\_path])

            submitted\_files += 1

        except subprocess.CalledProcessError as e:

            print(f"An error occurred while submitting {file\_path} to Cuckoo: {str(e)}")

    else:

        print(f"{file\_path} is not a valid file.")

print(f"Submitted {submitted\_files} out of {len(malware\_files)} files to Cuckoo.")

Here is an explanation of what the code does:

1. **Import necessary modules**: The os and subprocess modules are imported. The os module provides functions for interacting with the operating system, while the subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.
2. **Define the malware path**: The path where the malware samples are located is defined.
3. **Check if Cuckoo is running**: The script checks if Cuckoo Sandbox is currently running on the system. If it’s not running, a message is printed, and the script exits.
4. **List all files in the directory**: All files in the specified directory are listed and stored in ‘malware\_files’.
5. **Initialize a counter**: A counter for submitted files is initialized to 0.
6. **Submit files to Cuckoo for analysis**: The script loops through each file in ‘malware\_files’. For each file, it constructs the full path of the file and checks if it’s a valid file. If it is, it tries to submit the file to Cuckoo Sandbox for analysis using the cuckoo submit command. If successful, the counter for submitted files is incremented by 1. If an error occurs during submission, an error message is printed.
7. **Print submission summary**: After all files have been processed, a summary of how many files were successfully submitted to Cuckoo Sandbox out of the total number of files is printed.

For the benign files the word "malwares" was changed to "benigns" in the malware\_path variable in the source code.

The whole analysis took about 2 days to complete.

After analysing the benign and malware files, 500 analysis files (named 1-500) from each were stored in BenignReport and MalReport respectively.

## **Python basics for machine learning:**

The next step of the internship involved using the analysed data to predict whether a sample is malware or a benign application. This task can be accomplished with machine learning, which offers various classifiers to distinguish between different sets of data. Python is a popular programming language for using these tools thus, learning Python is essential if you want to do machine learning.

Before this internship, I only knew how to code in C. I decided to learn Python from w3schools5 (from Python Syntax to Python String Formatting), which gave me a good foundation in this language.

## **Machine Learning in Colab:**

Now that I understood how to code in Python, I decided to learn about ML.

Machine learning, in a nutshell, is the ability of a machine to use data and algorithms to imitate the "human" way of learning, gradually increasing accuracy.

I watched freeCodeCamp.org's "Machine Learning for Everyone - Full Course6" on YouTube, up until "SVM Implementation" to build a binary classifier for identifying benign or malicious files.

The video was informative and helped me understand various ML algorithms. The coding part for these was done on Google Colab which is a hosted Jupyter notebook that can run on any system and requires no setup to use. The best part is that it provides access to computing resources through the cloud free of charge.

Learning about machine learning and its algorithms helped me understand how machines can "learn" from large datasets.

## **Python script to automate the reading of report.json data and convert it into a .csv file for ML analysis:**

After learning about ML, it was time to create a dataset that could be fed to a model for analysis. For that first, I need to extract all the 1000 report.json files from malware and benign files. Cuckoo gives the analysis of a malicious file in a folder with a given ID number. This folder contains many subfolders and binaries that can be unsafe to be extracted in the main system. Only the "reports" subfolder is required.

Thus, I had to create a Python script (with the help of Bing Chat) that deletes any folder/ file that does not have the name "reports".

Here is the code:

import os

import shutil

# Set the username and path to the BenignReport folder

username = 'abc'

path = f'/home/{username}/Documents/BenignReport'

# Iterate over the subfolders

for i in range(1, 501):

  subfolder\_path = os.path.join(path, str(i))

  # Check if the subfolder exists

  if os.path.exists(subfolder\_path):

    print(f'Processing subfolder {i}...')

    # Iterate over the files and folders in the subfolder

    for item in os.listdir(subfolder\_path):

      item\_path = os.path.join(subfolder\_path, item)

      # Check if the item is not named "reports"

      if item != 'reports':

        # Delete the item

        if os.path.isfile(item\_path):

          print(f'Deleting file {item}...')

          os.remove(item\_path)

        elif os.path.isdir(item\_path):

          print(f'Deleting folder {item}...')

          shutil.rmtree(item\_path)

print('Done!')

Here is how it works:

1. **Set the username and path**: The username and path to the ‘BenignReport’ folder are set.
2. **Iterate over the subfolders**: The script iterates over 500 subfolders in the ‘BenignReport’ folder.
3. **Check if the subfolder exists**: For each subfolder, it checks if the subfolder exists.
4. **Iterate over the items in the subfolder**: If the subfolder exists, it iterates over all files and folders in the subfolder.
5. **Check if the item is not named “reports”**: For each item, it checks if the item is not named “reports”.
6. **Delete the item**: If the item is not named “reports”, it deletes the item. If the item is a file, it uses os.remove to delete it. If the item is a folder, it uses shutil.rmtree to delete it.
7. **Print completion message**: After all subfolders have been processed, it prints a completion message.

To do the same in BenignReport I replaced "MalReport" with "BenignReport" in the path variable.

However, when this code originally ran in the MalReport folder it failed to remove the binary file. This file could be deleted manually but it would take a long time hence, I had to create a code (again with the help of Bing Chat) that transfers the reports subfolder from the MalReport folder to another folder called the MalReport2 folder.

Here is the code:

import os

import shutil

# Set the path to the MalReport and MalReport2 folders

malreport\_path = '/home/abc/Documents/MalReport'

malreport2\_path = '/home/abc/Documents/MalReport2'

# Create the MalReport2 folder if it doesn't exist

if not os.path.exists(malreport2\_path):

    os.makedirs(malreport2\_path)

# Iterate over the subfolders in MalReport

for i in range(1, 501):

    malreport\_subfolder\_path = os.path.join(malreport\_path, str(i))

    # Check if the subfolder exists

    if os.path.exists(malreport\_subfolder\_path):

        # Check if the reports folder exists in the subfolder

        reports\_path = os.path.join(malreport\_subfolder\_path, 'reports')

        if os.path.exists(reports\_path):

            # Create the corresponding subfolder in MalReport2

            malreport2\_subfolder\_path = os.path.join(malreport2\_path, str(i))

            if not os.path.exists(malreport2\_subfolder\_path):

                os.makedirs(malreport2\_subfolder\_path)

            # Move the reports folder to the corresponding subfolder in MalReport2

            shutil.move(reports\_path, malreport2\_subfolder\_path)

Here is how it works:

1. **Set the paths**: The paths to the ‘MalReport’ and ‘MalReport2’ folders are set.
2. **Create the ‘MalReport2’ folder**: If the ‘MalReport2’ folder doesn’t exist, it is created.
3. **Iterate over the subfolders in ‘MalReport’**: The script iterates over 500 subfolders in the ‘MalReport’ folder.
4. **Check if the subfolder exists**: For each subfolder, it checks if the subfolder exists.
5. **Check if the ‘reports’ folder exists in the subfolder**: If the subfolder exists, it checks if a ‘reports’ folder exists within it.
6. **Create corresponding subfolder in ‘MalReport2’**: If a ‘reports’ folder exists, it creates a corresponding subfolder in the ‘MalReport2’ folder.
7. **Move the ‘reports’ folder**: Finally, it moves the ‘reports’ folder from the current subfolder in ‘MalReport’ to the corresponding subfolder in ‘MalReport2’.

The folders BenignReport and MalReport2 (renamed as MalReport) were put inside a folder called MLDataSet which was compressed and brought to the main system via Google Drive.

Now, I had to do the most important task which is to get data from the JSON files. My supervisors suggested me to check on what API calls are called out (which are present under the "behaviour" section) in the JSON files concerning the ones mentioned in MalApi.io7. This website is a valuable resource that maps Windows APIs to the typical tactics used by malware. It contains around 429 WindowsAPI calls that the malwares can call when they are running.

So, I decided to write a Python script that does exactly that and prints the output in a .csv file which is a file format used for feeding data for ML analysis.

Here is the code:

# Import necessary modules

import json

import csv

import os

# Define the path to the main folder

path = 'C:\\Users\\parih\\Desktop\\MalReport'

# Define a function to process the data in a JSON file

def converterFunc(jsonFilePath, csv\_writer):

    try:

        # Open the JSON file and load its data

        print(f'Trying to open JSON data in {jsonFilePath}')

        with open(jsonFilePath, 'r') as f:

            data = json.load(f)

    except json.JSONDecodeError:

        # Handle the error

        print(f'Error: Invalid JSON data in {jsonFilePath}')

        return []

    # Initializing "keys" list for storing WindowsAPI calls

    keys = ['CreateToolhelp32Snapshot', 'EnumDeviceDrivers', 'EnumProcesses', 'EnumProcessModules', 'EnumProcessModulesEx', 'FindFirstFileA', 'FindNextFileA', 'GetLogicalProcessorInformation', 'GetLogicalProcessorInformationEx', 'GetModuleBaseNameA', 'GetSystemDefaultLangId', 'GetVersionExA', 'GetWindowsDirectoryA', 'IsWoW64Process', 'Module32First', 'Module32Next', 'Process32First', 'Process32Next', 'ReadProcessMemory', 'Thread32First', 'Thread32Next', 'GetSystemDirectoryA', 'GetSystemTime', 'ReadFile', 'GetComputerNameA', 'VirtualQueryEx', 'GetProcessIdOfThread', 'GetProcessId', 'GetCurrentThread', 'GetCurrentThreadId', 'GetThreadId', 'GetThreadInformation', 'GetCurrentProcess', 'GetCurrentProcessId', 'SearchPathA', 'GetFileTime', 'GetFileAttributesA', 'LookupPrivilegeValueA', 'LookupAccountNameA', 'GetCurrentHwProfileA', 'GetUserNameA', 'RegEnumKeyExA', 'RegEnumValueA', 'RegQueryInfoKeyA', 'RegQueryMultipleValuesA', 'RegQueryValueExA', 'NtQueryDirectoryFile', 'NtQueryInformationProcess', 'NtQuerySystemEnvironmentValueEx', 'EnumDesktopWindows', 'EnumWindows', 'NetShareEnum', 'NetShareGetInfo', 'NetShareCheck', 'GetAdaptersInfo', 'PathFileExistsA', 'GetNativeSystemInfo', 'RtlGetVersion', 'GetIpNetTable', 'GetLogicalDrives', 'GetDriveTypeA', 'RegEnumKeyA', 'WNetEnumResourceA', 'WNetCloseEnum', 'FindFirstUrlCacheEntryA', 'FindNextUrlCacheEntryA', 'WNetAddConnection2A', 'WNetAddConnectionA', 'EnumResourceTypesA', 'EnumResourceTypesExA', 'GetSystemTimeAsFileTime', 'GetThreadLocale', 'EnumSystemLocalesA', 'CreateFileMappingA', 'CreateProcessA', 'CreateRemoteThread', 'CreateRemoteThreadEx', 'GetModuleHandleA', 'GetProcAddress', 'GetThreadContext', 'HeapCreate', 'LoadLibraryA', 'LoadLibraryExA', 'LocalAlloc', 'MapViewOfFile', 'MapViewOfFile2', 'MapViewOfFile3', 'MapViewOfFileEx', 'OpenThread', 'Process32First', 'Process32Next', 'QueueUserAPC', 'ReadProcessMemory', 'ResumeThread', 'SetProcessDEPPolicy', 'SetThreadContext', 'SuspendThread', 'Thread32First', 'Thread32Next', 'Toolhelp32ReadProcessMemory', 'VirtualAlloc', 'VirtualAllocEx', 'VirtualProtect', 'VirtualProtectEx', 'WriteProcessMemory', 'VirtualAllocExNuma', 'VirtualAlloc2', 'VirtualAlloc2FromApp', 'VirtualAllocFromApp', 'VirtualProtectFromApp', 'CreateThread', 'WaitForSingleObject', 'OpenProcess', 'OpenFileMappingA', 'GetProcessHeap', 'GetProcessHeaps', 'HeapAlloc', 'HeapReAlloc', 'GlobalAlloc', 'AdjustTokenPrivileges', 'CreateProcessAsUserA', 'OpenProcessToken', 'CreateProcessWithTokenW', 'NtAdjustPrivilegesToken', 'NtAllocateVirtualMemory', 'NtContinue', 'NtCreateProcess', 'NtCreateProcessEx', 'NtCreateSection', 'NtCreateThread', 'NtCreateThreadEx', 'NtCreateUserProcess', 'NtDuplicateObject', 'NtMapViewOfSection', 'NtOpenProcess', 'NtOpenThread', 'NtProtectVirtualMemory', 'NtQueueApcThread', 'NtQueueApcThreadEx', 'NtQueueApcThreadEx2', 'NtReadVirtualMemory', 'NtResumeThread', 'NtUnmapViewOfSection', 'NtWaitForMultipleObjects', 'NtWaitForSingleObject', 'NtWriteVirtualMemory', 'RtlCreateHeap', 'LdrLoadDll', 'RtlMoveMemory', 'RtlCopyMemory', 'SetPropA', 'WaitForSingleObjectEx', 'WaitForMultipleObjects', 'WaitForMultipleObjectsEx', 'KeInsertQueueApc', 'Wow64SetThreadContext', 'NtSuspendProcess', 'NtResumeProcess', 'DuplicateToken', 'NtReadVirtualMemoryEx', 'CreateProcessInternal', 'EnumSystemLocalesA', 'UuidFromStringA', 'CreateFileMappingA', 'DeleteFileA', 'GetModuleHandleA', 'GetProcAddress', 'LoadLibraryA', 'LoadLibraryExA', 'LoadResource', 'SetEnvironmentVariableA', 'SetFileTime', 'Sleep', 'WaitForSingleObject', 'SetFileAttributesA', 'SleepEx', 'NtDelayExecution', 'NtWaitForMultipleObjects', 'NtWaitForSingleObject', 'CreateWindowExA', 'RegisterHotKey', 'timeSetEvent', 'IcmpSendEcho', 'WaitForSingleObjectEx', 'WaitForMultipleObjects', 'WaitForMultipleObjectsEx', 'SetWaitableTimer', 'CreateTimerQueueTimer', 'CreateWaitableTimer', 'SetWaitableTimer', 'SetTimer', 'Select', 'ImpersonateLoggedOnUser', 'SetThreadToken', 'DuplicateToken', 'SizeOfResource', 'LockResource', 'CreateProcessInternal', 'TimeGetTime', 'EnumSystemLocalesA', 'UuidFromStringA', 'AttachThreadInput', 'CallNextHookEx', 'GetAsyncKeyState', 'GetClipboardData', 'GetDC', 'GetDCEx', 'GetForegroundWindow', 'GetKeyboardState', 'GetKeyState', 'GetMessageA', 'GetRawInputData', 'GetWindowDC', 'MapVirtualKeyA', 'MapVirtualKeyExA', 'PeekMessageA', 'PostMessageA', 'PostThreadMessageA', 'RegisterHotKey', 'RegisterRawInputDevices', 'SendMessageA', 'SendMessageCallbackA', 'SendMessageTimeoutA', 'SendNotifyMessageA', 'SetWindowsHookExA', 'SetWinEventHook', 'UnhookWindowsHookEx', 'BitBlt', 'StretchBlt', 'GetKeynameTextA', 'WinExec', 'FtpPutFileA', 'HttpOpenRequestA', 'HttpSendRequestA', 'HttpSendRequestExA', 'InternetCloseHandle', 'InternetOpenA', 'InternetOpenUrlA', 'InternetReadFile', 'InternetReadFileExA', 'InternetWriteFile', 'URLDownloadToFile', 'URLDownloadToCacheFile', 'URLOpenBlockingStream', 'URLOpenStream', 'Accept', 'Bind', 'Connect', 'Gethostbyname', 'Inet\_addr', 'Recv', 'Send', 'WSAStartup', 'Gethostname', 'Socket', 'WSACleanup', 'Listen', 'ShellExecuteA', 'ShellExecuteExA', 'DnsQuery\_A', 'DnsQueryEx', 'WNetOpenEnumA', 'FindFirstUrlCacheEntryA', 'FindNextUrlCacheEntryA', 'InternetConnectA', 'InternetSetOptionA', 'WSASocketA', 'Closesocket', 'WSAIoctl', 'ioctlsocket', 'HttpAddRequestHeaders', 'CreateToolhelp32Snapshot', 'GetLogicalProcessorInformation', 'GetLogicalProcessorInformationEx', 'GetTickCount', 'OutputDebugStringA', 'CheckRemoteDebuggerPresent', 'Sleep', 'GetSystemTime', 'GetComputerNameA', 'SleepEx', 'IsDebuggerPresent', 'GetUserNameA', 'NtQueryInformationProcess', 'ExitWindowsEx', 'FindWindowA', 'FindWindowExA', 'GetForegroundWindow', 'GetTickCount64', 'QueryPerformanceFrequency', 'QueryPerformanceCounter', 'GetNativeSystemInfo', 'RtlGetVersion', 'GetSystemTimeAsFileTime', 'CountClipboardFormats', 'CryptAcquireContextA', 'EncryptFileA', 'CryptEncrypt', 'CryptDecrypt', 'CryptCreateHash', 'CryptHashData', 'CryptDeriveKey', 'CryptSetKeyParam', 'CryptGetHashParam', 'CryptSetKeyParam', 'CryptDestroyKey', 'CryptGenRandom', 'DecryptFileA', 'FlushEfsCache', 'GetLogicalDrives', 'GetDriveTypeA', 'CryptStringToBinary', 'CryptBinaryToString', 'CryptReleaseContext', 'CryptDestroyHash', 'EnumSystemLocalesA', 'ConnectNamedPipe', 'CopyFileA', 'CreateFileA', 'CreateMutexA', 'CreateMutexExA', 'DeviceIoControl', 'FindResourceA', 'FindResourceExA', 'GetModuleBaseNameA', 'GetModuleFileNameA', 'GetModuleFileNameExA', 'GetTempPathA', 'IsWoW64Process', 'MoveFileA', 'MoveFileExA', 'PeekNamedPipe', 'WriteFile', 'TerminateThread', 'CopyFile2', 'CopyFileExA', 'CreateFile2', 'GetTempFileNameA', 'TerminateProcess', 'SetCurrentDirectory', 'FindClose', 'SetThreadPriority', 'UnmapViewOfFile', 'ControlService', 'ControlServiceExA', 'CreateServiceA', 'DeleteService', 'OpenSCManagerA', 'OpenServiceA', 'RegOpenKeyA', 'RegOpenKeyExA', 'StartServiceA', 'StartServiceCtrlDispatcherA', 'RegCreateKeyExA', 'RegCreateKeyA', 'RegSetValueExA', 'RegSetKeyValueA', 'RegDeleteValueA', 'RegOpenKeyExA', 'RegEnumKeyExA', 'RegEnumValueA', 'RegGetValueA', 'RegFlushKey', 'RegGetKeySecurity', 'RegLoadKeyA', 'RegLoadMUIStringA', 'RegOpenCurrentUser', 'RegOpenKeyTransactedA', 'RegOpenUserClassesRoot', 'RegOverridePredefKey', 'RegReplaceKeyA', 'RegRestoreKeyA', 'RegSaveKeyA', 'RegSaveKeyExA', 'RegSetKeySecurity', 'RegUnLoadKeyA', 'RegConnectRegistryA', 'RegCopyTreeA', 'RegCreateKeyTransactedA', 'RegDeleteKeyA', 'RegDeleteKeyExA', 'RegDeleteKeyTransactedA', 'RegDeleteKeyValueA', 'RegDeleteTreeA', 'RegDeleteValueA', 'RegCloseKey', 'NtClose', 'NtCreateFile', 'NtDeleteKey', 'NtDeleteValueKey', 'NtMakeTemporaryObject', 'NtSetContextThread', 'NtSetInformationProcess', 'NtSetInformationThread', 'NtSetSystemEnvironmentValueEx', 'NtSetValueKey', 'NtShutdownSystem', 'NtTerminateProcess', 'NtTerminateThread', 'RtlSetProcessIsCritical', 'DrawTextExA', 'GetDesktopWindow', 'SetClipboardData', 'SetWindowLongA', 'SetWindowLongPtrA', 'OpenClipboard', 'SetForegroundWindow', 'BringWindowToTop', 'SetFocus', 'ShowWindow', 'NetShareSetInfo', 'NetShareAdd', 'NtQueryTimer', 'GetIpNetTable', 'GetLogicalDrives', 'GetDriveTypeA', 'CreatePipe', 'RegEnumKeyA', 'WNetOpenEnumA', 'WNetEnumResourceA', 'WNetAddConnection2A', 'CallWindowProcA', 'NtResumeProcess', 'lstrcatA', 'ImpersonateLoggedOnUser', 'SetThreadToken', 'SizeOfResource', 'LockResource', 'UuidFromStringA']

    # Initializing "keysIndex" for storing the position of the API calls

    keysIndex = [x for x in range(len(keys))]

    # Initializing "keysDict" dictionary for merging the "keysIndex" and "keys" lists in the keys() and values() position respectively

    keysDict = dict(zip(keysIndex, keys))

    # Pre-initializing the "values" list with 0s and lenght similar to "keys" list

    value = [0]\*len(keys)

    print("JSON data opened sucessfully")

    # Check if the "behavior" key is present in the data dictionary

    if "behavior" in data:

        print("Processing JSON data...")

        # Iterate over the elements in the data["behavior"] dictionary

        for i in data["behavior"]:

            # Check if the key is "apistats"

            if i == "apistats":

                # Iterate over the elements in the data["behavior"]["apistats"] dictionary

                for j in data["behavior"]["apistats"]:

                    for keyNumbers, keyValues in keysDict.items():

                        if keyValues in data["behavior"]["apistats"][j]:

                            # Set its value to 1 in the dictionary

                            value[keyNumbers] = 1

        print("JSON data processed sucessfully")

    else:

        # Write an error message to the CSV file

        print(f'Error: "behavior" key not found in {jsonFilePath}')

        return []

    # Return the list of extracted values

    return value

# Define a function to count the number of subfolders in a directory

def countSubFolders():

    # Count the number of items in the directory that are directories themselves

    ctr = len([name for name in os.listdir(path) if os.path.isdir(os.path.join(path, name))])

    # Return the count

    return ctr

# Call the countSubFolders() function and store its return value in a variable named maxm

maxm = countSubFolders()

# Open a new CSV file named testdata.csv in write mode

with open('Maldata.csv', 'w', newline='') as f1:

    # Create a CSV writer object

    print(f'Opening {f1.name}')

    write = csv.writer(f1)

    print(f'Scanning through {path}')

    print(f'Total number of items:{maxm}')

    # Iterate over the range from 1 to maxm + 1 (inclusive)

    for i in range(1, maxm+1):

        # Construct the path to a subfolder by joining the path variable with the current loop index converted to a string

        subfolderPath = os.path.join(path, str(i))

        # Check if this subfolder exists

        if os.path.exists(subfolderPath):

            # Construct the path to a subfolder named reports inside this subfolder

            reportFolderPath = os.path.join(subfolderPath, 'reports')

            # Construct the path to a file named report.json inside this subfolder

            reportFilePath = os.path.join(reportFolderPath, 'report.json')

            # Check if this file exists

            if os.path.exists(reportFilePath):

                # Call the converterFunc() function with its path and csv\_writer as arguments and store its return value in a variable named values

                print("Going to: %s" % reportFolderPath)

                values = converterFunc(reportFilePath, write)

                # Write these values to a row in the CSV file using the CSV writer object

                print(f'Copying data to {f1.name}\n')

                write.writerow(values)

    print("Task completed sucessfully!")

Here is the explanation:

1. **Import necessary modules**: The json, csv, and os modules are imported.
2. **Define a function to count subfolders**: A function countSubFolders() is defined to count the number of subfolders in a specified directory. This function is then called, and its return value is stored in a variable maxm.
3. **Open CSV file**: A new CSV file named ‘Maldata.csv’ is opened in write mode.
4. **Iterate over subfolders**: The script iterates over the range from 1 to maxm + 1. For each index, it constructs the path to a subfolder and checks if this subfolder exists.
5. **Check for JSON file**: If the subfolder exists, it constructs the path to a ‘reports’ folder inside this subfolder and then to a ‘report.json’ file inside the ‘reports’ folder. It checks if this JSON file exists. If it does not exist, then an empty row is inserted in the CSV file.
6. **Extract data and write to CSV**: If the JSON file exists, it calls a function converterFunc() with the path to this file and the CSV writer object as arguments. This function first checks whether “behaviour” section is present in the JSON file. If it is not, then an empty row is inserted in the CSV file. Else, the return value of this function (“1” if the API calls exists or else “0”) is written to a row in the CSV file.
7. **Print completion message**: After all subfolders have been processed, it prints a completion message.

To do the same for benign reports, I replaced “MalReport” with “BenignReport” under the “path” variable.

Unfortunately, some files were unable to be analyzed due to malware utilizing the VM detection feature, preventing their execution in the sandbox. As a result, the "behaviour" section for some report.json files could not be obtained. Additionally, several benign files had corrupted report files, rendering them unreadable.

Despite all of this, we got ample data to work on.

## **Preprocessing of data in Python for smooth ML analysis:**

The CSV files obtained still need to be cleaned and organized since there are many spaces in the file due to some report.json files being corrupted or being of no use. Thus, I had to create a new Python script that notes down which of the file's report.json could not be used and the ones which worked had to be transferred to a new CSV file which makes the data look cleaner.

Here are the codes:

import csv

prog = list()

with open("Benigndata.csv", 'r') as fileIn:

    readIn = csv.reader(fileIn)

    with open("Benigndata2.csv", 'w', newline='') as fileOut:

        writeOut = csv.writer(fileOut)

        for idx, row in enumerate(readIn):

            if row == []:

                prog.append(idx)

            else:

                writeOut.writerow(row)

with open("data\_for\_Benigndata.txt", 'w') as Data:

    Data.write("Number of files removed: %s\n" % len(prog))

    one = [1]\*len(prog)

    real = [x+y for x, y in zip(prog, one)]

    Data.write("The removed files are: %s" % real)

import csv

prog = list()

with open("Maldata.csv", 'r') as fileIn:

    readIn = csv.reader(fileIn)

    with open("Maldata2.csv", 'w', newline='') as fileOut:

        writeOut = csv.writer(fileOut)

        for idx, row in enumerate(readIn):

            if row == []:

                prog.append(idx)

            else:

                writeOut.writerow(row)

with open("data\_for\_Maldata.txt", 'w') as Data:

    Data.write("Number of files removed: %s\n" % len(prog))

    one = [1]\*len(prog)

    real = [x+y for x, y in zip(prog, one)]

    Data.write("The removed files are: %s" % real)

Here is their explanation:

1. **Open the input CSV file**: Each script opens an input CSV file (“Maldata.csv” or “Benigndata.csv”) for reading.
2. **Open the output CSV file**: Each script also opens a corresponding output CSV file (“Maldata2.csv” or “Benigndata2.csv”) for writing.
3. **Process rows**: Each script then reads the input CSV file row by row. For each row, it checks if the row is empty. If the row is empty, it records the index of this row in a list prog. If the row is not empty, it writes this row to the output CSV file.
4. **Write summary to text file**: After all rows have been processed, each script writes a summary to a text file (“data\_for\_Maldata.txt” or “data\_for\_Benigndata.txt”). The summary includes the number of files removed (i.e., the number of empty rows in the input CSV file) and the indices of these files.

After this was done for both malware and benign files, I had to add a new column called the "class" column since this represents the output category of data in ML. In my case, malware is set to class "1" and benign is set to class "0".

Here are the codes:

import csv

Class = "0"

with open("Benigndata2.csv", 'r') as fileIn:

    readIn = csv.reader(fileIn)

    with open("Benigndata3.csv", 'w', newline = '') as fileOut:

        writeOut = csv.writer(fileOut)

        for row in readIn:

            row.append(Class)

            writeOut.writerow(row)

import csv

Class = "1"

with open("Maldata2.csv", 'r') as fileIn:

    readIn = csv.reader(fileIn)

    with open("Maldata3.csv", 'w', newline = '') as fileOut:

        writeOut = csv.writer(fileOut)

        for row in readIn:

            row.append(Class)

            writeOut.writerow(row)

Here is their explanation:

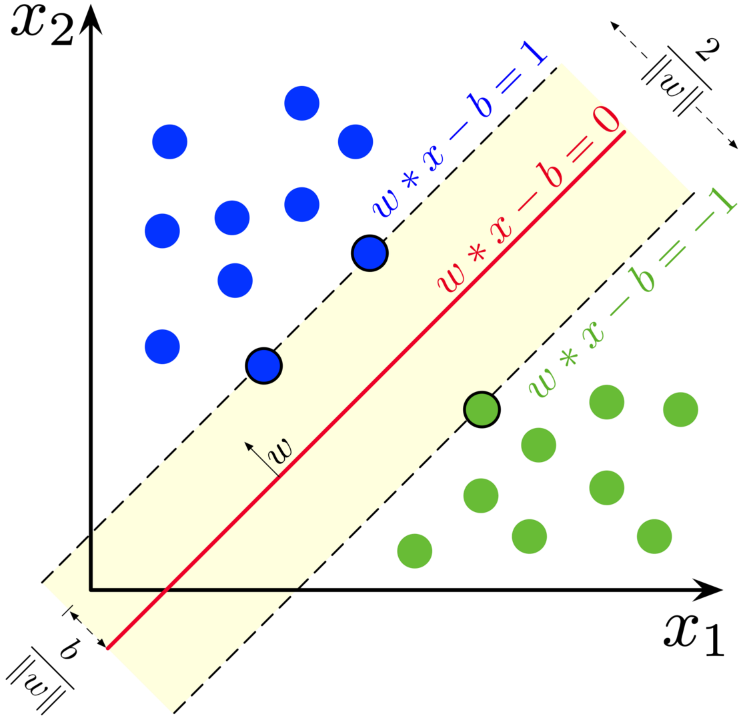
1. **Open the input CSV file**: Each script opens an input CSV file (“Benigndata2.csv” or “Maldata2.csv”) for reading.
2. **Open the output CSV file**: Each script also opens a corresponding output CSV file (“Benigndata3.csv” or “Maldata3.csv”) for writing.
3. **Process rows and append class**: Each script then reads the input CSV file row by row. For each row, it appends a class label (“0” for benign file and “1” for malicious file) to the row and writes this updated row to the output CSV file.

Finally, I merged the two data into one file “MalBenData.csv” which would be used for ML analysis.

## **Analysing Malware and Benign Reports with SVM Classifier:**

Now that my dataset is prepared, it's time to use a machine learning classifier to sort the data easily. Since my class only has 0s and 1s, a binary classifier would be ideal.

The **Support Vector Machine (SVM)8** is one of the most effective binary classifiers. It's a supervised algorithm used for classification problems. SVM determines the optimal hyperplane in the N-dimensional space to separate the data points into different classes in the feature space. The hyperplane acts as the decision boundary that separates the data points of different classes in the feature space while maximizing the margin between the closest points of the different classes.



*Source:* https://en.wikipedia.org/wiki/Support\_vector\_machine

Here is the code with explanations:

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import RandomOverSampler

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

This code imports libraries for:

1. **numpy & pandas:** Data manipulation and analysis.
2. **StandardScale**r**:** Feature standardization.
3. **RandomOverSampler:** Handling imbalanced datasets.
4. **SVC:** Support Vector Machine for classification.
5. **classification\_report:** Generating classification metrics.

col = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '70', '71', '72', '73', '74', '75', '76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89', '90', '91', '92', '93', '94', '95', '96', '97', '98', '99', '100', '101', '102', '103', '104', '105', '106', '107', '108', '109', '110', '111', '112', '113', '114', '115', '116', '117', '118', '119', '120', '121', '122', '123', '124', '125', '126', '127', '128', '129', '130', '131', '132', '133', '134', '135', '136', '137', '138', '139', '140', '141', '142', '143', '144', '145', '146', '147', '148', '149', '150', '151', '152', '153', '154', '155', '156', '157', '158', '159', '160', '161', '162', '163', '164', '165', '166', '167', '168', '169', '170', '171', '172', '173', '174', '175', '176', '177', '178', '179', '180', '181', '182', '183', '184', '185', '186', '187', '188', '189', '190', '191', '192', '193', '194', '195', '196', '197', '198', '199', '200', '201', '202', '203', '204', '205', '206', '207', '208', '209', '210', '211', '212', '213', '214', '215', '216', '217', '218', '219', '220', '221', '222', '223', '224', '225', '226', '227', '228', '229', '230', '231', '232', '233', '234', '235', '236', '237', '238', '239', '240', '241', '242', '243', '244', '245', '246', '247', '248', '249', '250', '251', '252', '253', '254', '255', '256', '257', '258', '259', '260', '261', '262', '263', '264', '265', '266', '267', '268', '269', '270', '271', '272', '273', '274', '275', '276', '277', '278', '279', '280', '281', '282', '283', '284', '285', '286', '287', '288', '289', '290', '291', '292', '293', '294', '295', '296', '297', '298', '299', '300', '301', '302', '303', '304', '305', '306', '307', '308', '309', '310', '311', '312', '313', '314', '315', '316', '317', '318', '319', '320', '321', '322', '323', '324', '325', '326', '327', '328', '329', '330', '331', '332', '333', '334', '335', '336', '337', '338', '339', '340', '341', '342', '343', '344', '345', '346', '347', '348', '349', '350', '351', '352', '353', '354', '355', '356', '357', '358', '359', '360', '361', '362', '363', '364', '365', '366', '367', '368', '369', '370', '371', '372', '373', '374', '375', '376', '377', '378', '379', '380', '381', '382', '383', '384', '385', '386', '387', '388', '389', '390', '391', '392', '393', '394', '395', '396', '397', '398', '399', '400', '401', '402', '403', '404', '405', '406', '407', '408', '409', '410', '411', '412', '413', '414', '415', '416', '417', '418', '419', '420', '421', '422', '423', '424', '425', '426', '427', '428', '429', 'Class']

df = pd.read\_csv("MalBenData.csv", names = col)

df.head()

This code reads a CSV file named “MalBenData.csv” into a pandas DataFrame ‘df’. The column names are specified as ‘1’, ‘2’, ‘3’, ‘4’…, ‘429’ and ‘Class’. It then displays the first 5 rows of the DataFrame using the head() function.

def scaleDataSet(dataframe, oversample = False):

  X = dataframe[dataframe.columns[:-1]].values

  Y = dataframe[dataframe.columns[-1]].values

  scaler = StandardScaler()

  X = scaler.fit\_transform(X)

  if oversample:

    ros = RandomOverSampler()

    X, Y = ros.fit\_resample(X, Y)

  data = np.hstack((X,np.reshape(Y, (-1,1))))

  return data, X, Y

This function, scaleDataSet(), performs several operations on a given dataframe:

1. **Separate Features and Labels**: It separates the features (X) and labels (Y) from the dataframe. The features are all columns except the last one, and the label is the last column.
2. **Feature Scaling**: It standardizes the features (X) using StandardScaler from sklearn, which standardizes features by removing the mean and scaling to unit variance.
3. **Oversampling (Optional)**: If the oversample parameter is set to True, it performs oversampling on the dataset using RandomOverSampler from imblearn. This can help improve the performance of the model if the dataset is imbalanced.
4. **Combine Features and Labels**: It then combines the standardized features and labels back into one dataset.

The function returns three items: the combined dataset, the standardized features (X), and the labels (Y).

train, valid, test = np.split(df.sample(frac=1), [int(0.6\*len(df)), int(0.8\*len(df))])

This code splits a dataframe df into three datasets: train, valid, and test. The dataframe is first shuffled using sample(frac=1). It’s then split at the 60% mark into train and the remaining 40%.

The remaining 40% is further split at the 80% mark of the original dataframe (or equivalently, the 50% mark of the remaining 40%) into valid and test. This results in a 60%/20%/20% split of the original dataframe into train, valid, and test respectively.

train = pd.DataFrame(train)

valid = pd.DataFrame(valid)

test = pd.DataFrame(test)

train, xTrain, yTrain = scaleDataSet(train, oversample = True)

valid, xValid, yValid = scaleDataSet(valid, oversample = False)

test, xTest, yTest = scaleDataSet(test, oversample = False)

This code performs several operations on the train, valid, and test datasets:

1. **Convert to DataFrame**: It first converts train, valid, and test datasets into pandas DataFrames.
2. **Scale and Oversample**: It then calls the previously defined scaleDataSet() function on each of these datasets. This function standardizes the features, optionally performs oversampling (if oversample=True), and combines the features and labels back into one dataset. For the train dataset, oversampling is performed (oversample=True), while for the valid and test datasets, oversampling is not performed (oversample=False).

The function returns three items for each dataset: the combined dataset, the standardized features, and the labels. These are stored in variables like train, xTrain, and yTrain for the training dataset.

SVMModel = SVC()

SVMModel = SVMModel.fit(xTrain, yTrain)

This code creates a Support Vector Machine (SVM) model using the SVC() function from the sklearn.svm library. It then fits this model to the training data (xTrain and yTrain) using the fit() method. The fitted model is stored in the variable SVMModel.

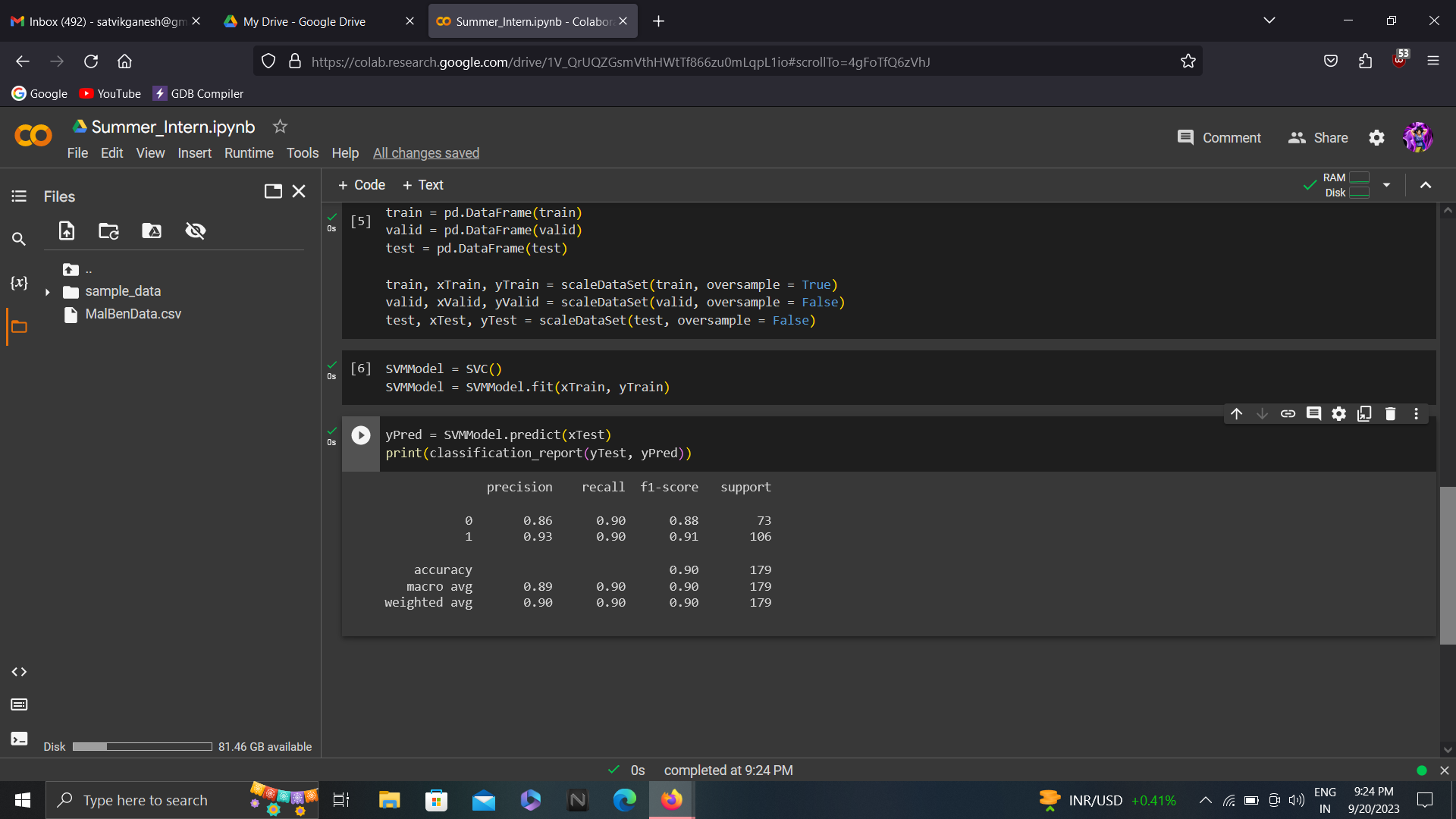
yPred = SVMModel.predict(xTest)

print(classification\_report(yTest, yPred))

This code uses the trained Support Vector Machine (SVM) model SVMModel to predict the labels for the test data xTest. The predictions are stored in yPred. It then prints a classification report using the classification\_report() function from sklearn.metrics, which shows the main classification metrics like precision, recall, f1-score, etc. for the true test labels yTest and the predicted labels yPred.

**NOTE:** The above code was implemented in Google Colab.

Here is the result:



The results indicate that the model has an overall accuracy of 90%, which is pretty good for first try.

## **Conclusion and future prospects:**

In conclusion, my internship at C3i Hub was invaluable in terms of learning new topics and developing new skills. Specifically, I gained knowledge about MITRE techniques and malware analysis, which helped me understand how hackers infiltrate systems and exploit resources to steal sensitive information. Additionally, I honed my Python skills, allowing me to create automated scripts for recursive tasks, and the gained knowledge about machine learning helped me test various models and implement them in real-world scenarios.

I believe that this experience will benefit me greatly in my future studies or the industry. In particular, the ability to accurately classify malware and benign software using machine learning has the potential to significantly improve the detection and prevention of cyber-attacks.

## **Acknowledgments:**

I want to express my sincere gratitude to Prof. Manindra Agrawal for accepting me as a summer intern at C3i Hub and thus giving me an opportunity to work with his team members. My special thanks to Mr. Sai Charan for helping me with the project, answering my queries, and caring doubts. I would also like to thank Mr. Ravi Poonia, Ms. Monika Bisht, Mr. Anand Handa, and Ms. Gargi Sarkar for the discussion time and advise.

## **Sources/References:**

1. <https://www.exabeam.com/explainers/mitre-attck/what-is-mitre-attck-an-explainer/>

<https://attack.mitre.org/techniques/enterprise/>

2. <https://www.virtualbox.org/>

3. <https://cuckoo.readthedocs.io/en/latest/introduction/what/>

<https://cuckoosandbox.org/>

4. <https://bazaar.abuse.ch/>

5. <https://www.w3schools.com/python/>

6. <https://www.youtube.com/watch?v=i_LwzRVP7bg>

7. <https://malapi.io/>

8. <https://www.geeksforgeeks.org/support-vector-machine-algorithm/>