

REPORT
on
Malware Detection using ML

A Project Report submitted in partial fulfilment of the
requirements for the award of the degree of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
with IBM specialization in INTERNET OF THINGS AND SMART CITIES

Submitted by

Ayush Patel (Enroll. No. R164216014)
Keshav Gupta (Enroll. No. R164216026)
Prakhar Garg (Enroll. No. R164216043)
Bhavya Joshi (Enroll. No. R164216015)

Under the guidance of

Ms. Amber Hayat
Assistant Professor(SS), SoCS
Department of Systemics



SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Bidholi Campus, Energy Acres, Dehradun – 248007.
November - 2019

DECLARATION

I hereby declare that this submission is my own and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other Degree or Diploma of the University or other Institute of Higher learning, except where due acknowledgement has been made in the text.

Ayush Patel (Enroll No. R164216014) -

Bhavya Joshi (Enroll No. R164216015) -

Keshav Gupta (Enroll No. R164216026) -

Prakhar Garg (Enroll No. R164216043) -

CERTIFICATE

This is to certify that the project titled Identification and Classification of Malware submitted by Ayush Patel (Enroll. No. R164216014), Bhavya Joshi (Enroll. No. R164216015), Keshav Gupta (Enroll. No. R164216026), and Prakhar Garg (Enroll. No. R164216043) to the University of Petroleum & Energy Studies, for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING is a bonafide record of project work carried out by them under my supervision and guidance. The content of the project, in full or parts have not been submitted to any other Institute or University for the award of any other degree or diploma.

Date: _____

Name of Guide
Ms. Amber Hayat
SOCS

HOD
Dr. Neelu Jyoti Ahuja
Dept. of Systemics | SOCS

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide Ms. Amber Hayat, for all advice, encouragement and constant support she has given us throughout our project work. This work would not have been possible without her support and valuable suggestions.

We sincerely thank to our Head of the Department, Dr. Neelu Jyoti Ahuja, for her great support in doing our Malware detection using Machine Learning at SoCS.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

Name	Ayush Patel	Bhavya Joshi	Keshav Gupta	Prakhar Garg
Roll No.	R164216014	R164216015	R164216026	R164216043

ABSTRACT

There has been an exponential growth in the number of malwares in the cyber world in the last few years. Modern malwares use sophisticated techniques such as polymorphism and metamorphism to thwart the malware detection and analysis. Detecting malware on the basis of their features and behavior is critical for the computer security community. Most anti-virus depends on the signature-based detection which is relatively easy to evade and is ineffective for zero-day exploit based malwares. With this project, we propose a new approach to identify malwares using static analysis, i.e. without executing. With the help of different machine learning models, we will identify malwares and analyse the performance of different models for the same.

Keywords: Malwares, Static Analysis, Machine Learning

TABLE OF CONTENTS

Contents

1	Introduction	8
1.1	Type of Analysis	8
1.2	Static Analysis	8
1.3	PE File	9
2	Literature Review	10
3	Problem Statement	10
4	Objective	10
5	Design Methodology	10
6	Implementation	11
6.1	Pseudocode	11
7	Code and Output	18
7.1	Code	18
7.2	Output	20

LIST OF FIGURES

List of Figures

1 PE File Structure	9
2 Waterfall Model	11
3 Flow Chart	11
4 Providing Path for Unseen files	14
5 Depict Unseen file as legitimate	14
6 Random Forest	16
7 Gaussian Naïve Bayes.	17
8 Legitimate Output.	20
9 Malicious Output.	20

INTRODUCTION

Over the past few years, The Internet has become a vital part in our lives with increased usage of services like online banking, online reservation etc., and our dependence on the Internet is expected to grow. With the rise of the Internet, there has been huge growth in the amount of malware samples in the wild.

The term Malware is used when referring to a malicious software, which is designed to disrupt or gain unauthorized access to a system. There are various types of malware, brief description of each is given below:

- Viruses and Worms: These are famous for the way they spread to other systems. Virus can infect any normal file or program on the system by attaching itself to it and will infect the whole system as well when executed. On the other hand, worms spread automatically using the computer network and performs various malicious actions
- Spy-Wares: These programs are crafted to intrude on the privacy of users by leaking confidential information like screen-shots, on-line banking credentials or Internet surfing history etc. of users to the third party without user's knowledge. They get installed on the system very secretly and could be difficult to detect.
- Trojans: Often referred as Trojan Horses, are programs designed to have an appearance of serving a useful utility but actually contains some hidden malicious payload which may be to install spy-wares
- Root-kits: These are collection of tools which are used to gain administrator/root privileges on a system by exploiting a known vulnerability in the kernel or through passwords (Brute-forcing commonly used passwords or using social engineering). Once installed, they are very difficult to detect.

It is therefore required to differentiate between a malware and a clean ware once an unknown file has entered into the system in order to ensure it will not perform any malicious activity. Several ways by which a malware could enter in your system include:

- Downloading a file from Internet.
- Opening attachments in email or social networking websites such as face-book.
- Plugging external hard drives, USB and CDs to the system.
- This project aims to perform malware detection, where models will be trained on large corpus of executable using good set of discriminative predictors extracted through static analysis

1. Type of Analysis

- Static Analysis - examining the executable files without viewing the actual instructions.
- Dynamic Analysis - observing the behavior of the malware while it is actually running

2. Static Analysis

- Static analysis is the technique of getting insights into the malware functionalities by examining its static properties. Without executing the malware, we can examine various properties such as the strings embedded in the binary, PE Header, embedded resource directories, meta-data and disassembly etc.
- The Portable Executable (PE) format is a file format for executable, object code and DLLs, used in 32-bit and 64-bit versions of Windows operating systems.

3. PE File

The Portable Executable (PE) format is a file format for executable, object code and DLLs, used in 32-bit and 64-bit versions of Windows operating systems.

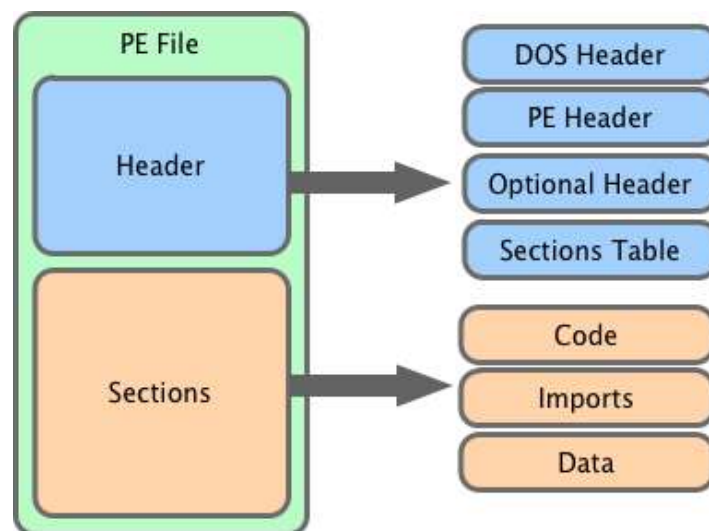


Figure 1: *PE File Structure*

- **DOS Header**

Starts with the first 64 bytes of every PE file. It's there because DOS can recognize it as a valid executable and can run it in the DOS stub mode. As we can investigate on the

Windows.inc we can see below details: Same thing can be found on the cff- explorer which is very popular malware analysis tool for PE file validation. We also have a same implementation as a picture as we can see we have a list of structure that came under DOS header. We will not discuss everything as it is beyond our scope; we will discuss important ones that are required, such as magic and if a new structure.

- **Portable Executable(PE) Header**

Like other executable files, a PE file has a collection of fields that defines what the rest of file looks like. The header contains info such as the location and size of code, as we discussed earlier. The first few hundred bytes of the typical PE file are taken up by the MS-DOS stub. There are some basic sub-sections defined in the header section itself; they are listed below: Number of Sections: This defines the size of the section table, which immediately follow the header. Size of Optional Header: This lies between top of the optional header and the start of the section table. This is the size of the optional header that is required for an executable file. This value should be zero for an object file. Characteristics: This is the characteristic flags that indicate an attribute of the object or image file.

LITERATURE REVIEW

Conventional classification methods have been relying on static feature extraction using reverse engineering, examining DLL usage from PE Headers and Strings, used list of functions inside DLL called by binary, string features and byte sequences using hex-dump to perform classification [2]. Also used sequences of API called under windows, extracted using de-compilation analysis. Another technique known as the n-gram sequence of byte code obtained after applying the hex-dump utility has been used for higher accuracy by [5]. Another frequently used approach is based on extracting the strings from the executable using strings utility from Linux. Authors of used IDA to extract strings and perform classification. Though strings feature has been successful in classification with greater accuracy, they are not helpful in case the malware is obfuscated. Packers usually render almost all the strings inside the executable unprintable [3].

PROBLEM STATEMENT

In today's world people have no idea about that their devices are infected by malwares that make their computers bots or zombie machines that can-do task directed by the attacker without users permission or knowledge and many more tasks like that. Prevention Mechanism based on Hash computation to check if the file is malware have become old and attackers have discovered new ways to avoid them.

OBJECTIVES

To Identify and classify malwares using static analysis i.e. without executing with the help of Machine learning models.

METHODOLOGY

Requirement Analysis - The programming language we used was Python so we found out what will be the system requirements, libraries requirement, and compiler/ID requirements for conducting our project successfully.

The required libraries for the project were:

Python PE file

Pandas

Hashlib

Matplotlib

NumPy

Design- After the requirements phase, we begin designing the algorithm required for our Project

Implementation- After the requirements analysis and design of algorithms of the project, we implemented the algorithm in Python language.

Testing- After the implementation was complete, we tested our code on various systems and network connections and we performed grey box testing for the same.

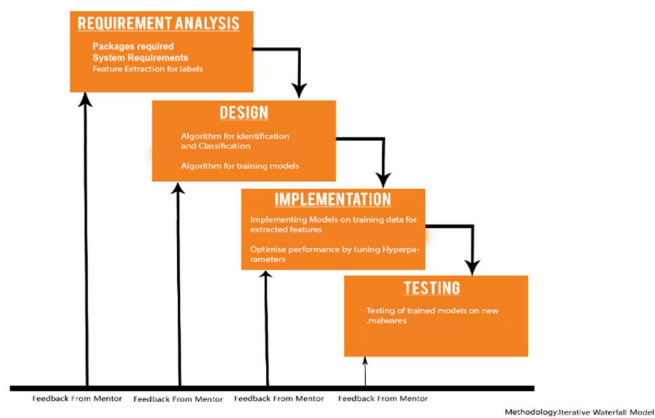


Figure 2: *Waterfall Model*

IMPLEMENTATION

- After the hash value for every file is computed.
- Header details are extracted.
- Applying the machine learning algorithms.
- We have classified Unseen Setup files as legitimate or malicious

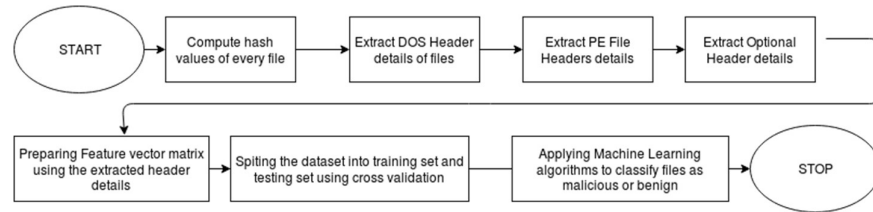


Figure 3: *Flow Chart*

6.1 Pseudocode

1. START
2. Compute hash values of every file and check whether any of the file in the corpus is duplicate and then remove the duplicate files.
3. Extract header details of the binaries with the help of PE File module functions of python for the analysis purpose.

The following header details are extracted

- DOS header
- PE File header
- Optional header

4. Prepare the feature vector matrix by selecting best features for training and testing purpose of the dataset.
5. With Cross-Validation split the dataset into training and testing set.
6. Apply ML algorithms to classify files as malwares and benign Algorithms used to evaluate:
 - K-Nearest Neighbors
 - Decision Trees
 - Random Forest
 - Logistic Regression
 - SVM (Support Vector Machines)

7. STOP

Cross Validation

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

K-Fold Cross Validation- In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is re-trained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimate- ton. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

- **Gradient Boosting-**Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Decision Tree and Random Forest- A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance- event outcomes, resource costs, and utility. Random forest algorithm is a supervised classification algorithm. As the name suggest, this algorithm creates the forest with a number of trees. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

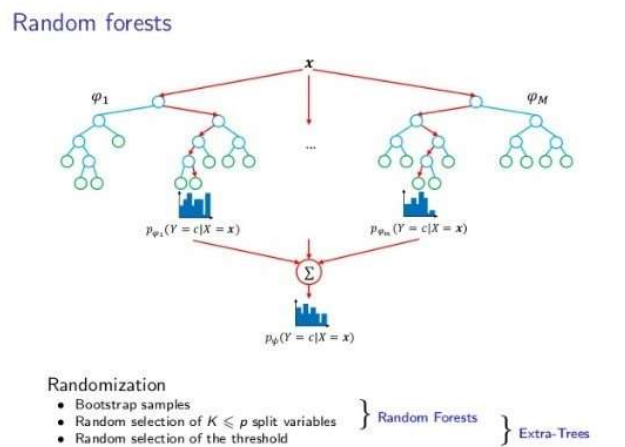


Figure 4: *Random Forest*

- **Linear regression:** An attractive model because the representation is so simple. The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric. The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

$$y = B0 + B1*x$$

- **Naïve Bayes:** This classifier is a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

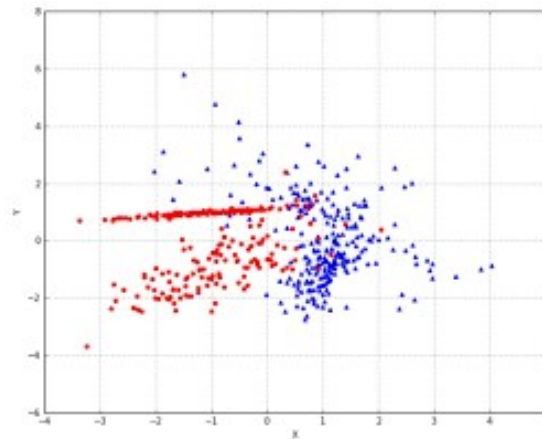


Figure 5: Naïve Bayes

CODE AND OUTPUT

7.1 CODE

```
from google.colab import drive
drive.mount('/content/drive')
!cp "/content/drive/My Drive/data.csv" "data.csv"

!pip install pefile
import os
import pandas
import numpy
import pickle
import pefile
import sklearn.ensemble as ek
import array
import math
import pickle
from sklearn.externals import joblib
import sys
import argparse
from sklearn import cross_validation, tree, linear_model
from sklearn.model_selection import KFold, cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn.externals import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn import svm
from sklearn.linear_model import LinearRegression

dataset = pandas.read_csv('data.csv', sep='|', low_memory=False)
dataset.head()
dataset.describe()
dataset.groupby(dataset['legitimate']).size()

X = dataset.drop(['Name', 'md5', 'legitimate'], axis=1).values
y = dataset['legitimate'].values
extratrees = ek.ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
print nbfeatures

X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_new, y, test_size=0.2)
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]], extratrees.feature_importances_[index[f]]))
    features.append(dataset.columns[2+f])
```

```

model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
          "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
          "LinearRegression":LinearRegression(),
          "RandomForest":ek.RandomForestClassifier(n_estimators=50),
          "Gaussian Naive Bayes":GaussianNB()
}
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score
winner = max(results, key=results.get)
open('features.pkl', 'wb').write(pickle.dumps(features))
clf = model[winner]
res = clf.predict(X_new)
mt = confusion_matrix(y, res)
print("False positive rate : %f%%" % ((mt[0][1] / float(sum(mt[0])))*100))
print('False negative rate : %f%%' % ( (mt[1][0] / float(sum(mt[1]))*100))
clf = joblib.load('classifier/classifier.pkl')
features = pickle.loads(open(os.path.join('classifier/features.pkl'),'r').read())

def get_entropy(data):
    if len(data) == 0:
        return 0.0
    occurrences = array.array('L', [0] * 256)
    for x in data:
        occurrences[x if isinstance(x, int) else ord(x)] += 1
    entropy = 0
    for x in occurrences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x * math.log(p_x, 2)
    return entropy

def get_resources(pe):
    resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in resource_id.directory.entries:
                                data = pe.get_data(resource_lang.data.struct.OffsetToData,
                                                    resource_lang.data.struct.Size)
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)
                                resources.append([entropy, size])
        except Exception as e:
            return resources
    return resources

```

```

def get_version_info(pe):
    res = {}
    for fileinfo in pe.FileInfo:
        if fileinfo.Key == 'StringFileInfo':
            for st in fileinfo.StringTable:
                for entry in st.entries.items():
                    res[entry[0]] = entry[1]
        if fileinfo.Key == 'VarFileInfo':
            for var in fileinfo.Var:
                res[var.entry.items()[0][0]] = var.entry.items()[0][1]
    if hasattr(pe, 'VS_FIXEDFILEINFO'):
        res['flags'] = pe.VS_FIXEDFILEINFO.FileFlags
        res['os'] = pe.VS_FIXEDFILEINFO.FileOS
        res['type'] = pe.VS_FIXEDFILEINFO.FileType
        res['file_version'] = pe.VS_FIXEDFILEINFO.FileVersionLS
        res['product_version'] = pe.VS_FIXEDFILEINFO.ProductVersionLS
        res['signature'] = pe.VS_FIXEDFILEINFO.Signature
        res['struct_version'] = pe.VS_FIXEDFILEINFO.StrucVersion
    return res

def extract_infos(fpath):
    res = {}
    pe = pefile.PE("E:\vlc.exe")
    res['Machine'] = pe.FILE_HEADER.Machine
    res['SizeOfOptionalHeader'] = pe.FILE_HEADER.SizeOfOptionalHeader
    res['Characteristics'] = pe.FILE_HEADER.Characteristics
    res['MajorLinkerVersion'] = pe.OPTIONAL_HEADER.MajorLinkerVersion
    res['MinorLinkerVersion'] = pe.OPTIONAL_HEADER.MinorLinkerVersion
    res['SizeOfCode'] = pe.OPTIONAL_HEADER.SizeOfCode
    res['SizeOfInitializedData'] = pe.OPTIONAL_HEADER.SizeOfInitializedData
    res['SizeOfUninitializedData'] = pe.OPTIONAL_HEADER.SizeOfUninitializedData
    res['AddressOfEntryPoint'] = pe.OPTIONAL_HEADER.AddressOfEntryPoint
    res['BaseOfCode'] = pe.OPTIONAL_HEADER.BaseOfCode
    try:
        res['BaseOfData'] = pe.OPTIONAL_HEADER.BaseOfData
    except AttributeError:
        res['BaseOfData'] = 0
    res['ImageBase'] = pe.OPTIONAL_HEADER.ImageBase
    res['SectionAlignment'] = pe.OPTIONAL_HEADER.SectionAlignment
    res['FileAlignment'] = pe.OPTIONAL_HEADER.FileAlignment
    res['MajorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MajorOperatingSystemVersion
    res['MinorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MinorOperatingSystemVersion
    res['MajorImageVersion'] = pe.OPTIONAL_HEADER.MajorImageVersion
    res['MinorImageVersion'] = pe.OPTIONAL_HEADER.MinorImageVersion
    res['MajorSubsystemVersion'] = pe.OPTIONAL_HEADER.MajorSubsystemVersion
    res['MinorSubsystemVersion'] = pe.OPTIONAL_HEADER.MinorSubsystemVersion
    res['SizeOfImage'] = pe.OPTIONAL_HEADER.SizeOfImage
    res['SizeOfHeaders'] = pe.OPTIONAL_HEADER.SizeOfHeaders
    res['Checksum'] = pe.OPTIONAL_HEADER.CheckSum
    res['Subsystem'] = pe.OPTIONAL_HEADER.Subsystem
    res['DllCharacteristics'] = pe.OPTIONAL_HEADER.DllCharacteristics
    res['SizeOfStackReserve'] = pe.OPTIONAL_HEADER.SizeOfStackReserve
    res['SizeOfStackCommit'] = pe.OPTIONAL_HEADER.SizeOfStackCommit

```

```

res['SizeOfHeapReserve'] = pe.OPTIONAL_HEADER.SizeOfHeapReserve
res['SizeOfHeapCommit'] = pe.OPTIONAL_HEADER.SizeOfHeapCommit
res['LoaderFlags'] = pe.OPTIONAL_HEADER.LoaderFlags
res['NumberOfRvaAndSizes'] = pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

res['SectionsNb'] = len(pe.sections)
entropy = list(map(lambda x: x.get_entropy(), pe.sections))
res['SectionsMeanEntropy'] = sum(entropy) / float(len(entropy))
res['SectionsMinEntropy'] = min(entropy)
res['SectionsMaxEntropy'] = max(entropy)
raw_sizes = list(map(lambda x: x.SizeOfRawData, pe.sections))
res['SectionsMeanRawsize'] = sum(raw_sizes) / float(len(raw_sizes))
res['SectionsMinRawsize'] = min(raw_sizes)
res['SectionsMaxRawsize'] = max(raw_sizes)
virtual_sizes = list(map(lambda x: x.Misc_VirtualSize, pe.sections))
res['SectionsMeanVirtualsize'] = sum(virtual_sizes) / float(len(virtual_sizes))
res['SectionsMinVirtualsize'] = min(virtual_sizes)
res['SectionMaxVirtualsize'] = max(virtual_sizes)
try:
    res['ImportsNbDLL'] = len(pe.DIRECTORY_ENTRY_IMPORT)
    imports = sum([x.imports for x in pe.DIRECTORY_ENTRY_IMPORT], [])
    res['ImportsNb'] = len(imports)
    res['ImportsNbOrdinal'] = len(list(filter(lambda x: x.name is None, imports)))
except AttributeError:
    res['ImportsNbDLL'] = 0
    res['ImportsNb'] = 0
    res['ImportsNbOrdinal'] = 0
try:
    res['ExportNb'] = len(pe.DIRECTORY_ENTRY_EXPORT.symbols)
except AttributeError:
    res['ExportNb'] = 0
resources = get_resources(pe)
res['ResourcesNb'] = len(resources)
if len(resources) > 0:
    entropy = list(map(lambda x: x[0], resources))
    res['ResourcesMeanEntropy'] = sum(entropy) / float(len(entropy))
    res['ResourcesMinEntropy'] = min(entropy)
    res['ResourcesMaxEntropy'] = max(entropy)
    sizes = list(map(lambda x: x[1], resources))
    res['ResourcesMeanSize'] = sum(sizes) / float(len(sizes))
    res['ResourcesMinSize'] = min(sizes)
    res['ResourcesMaxSize'] = max(sizes)
else:
    res['ResourcesNb'] = 0
    res['ResourcesMeanEntropy'] = 0
    res['ResourcesMinEntropy'] = 0
    res['ResourcesMaxEntropy'] = 0
    res['ResourcesMeanSize'] = 0
    res['ResourcesMinSize'] = 0
    res['ResourcesMaxSize'] = 0
try:
    res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
except AttributeError:

```

```

res['LoadConfigurationSize'] = 0

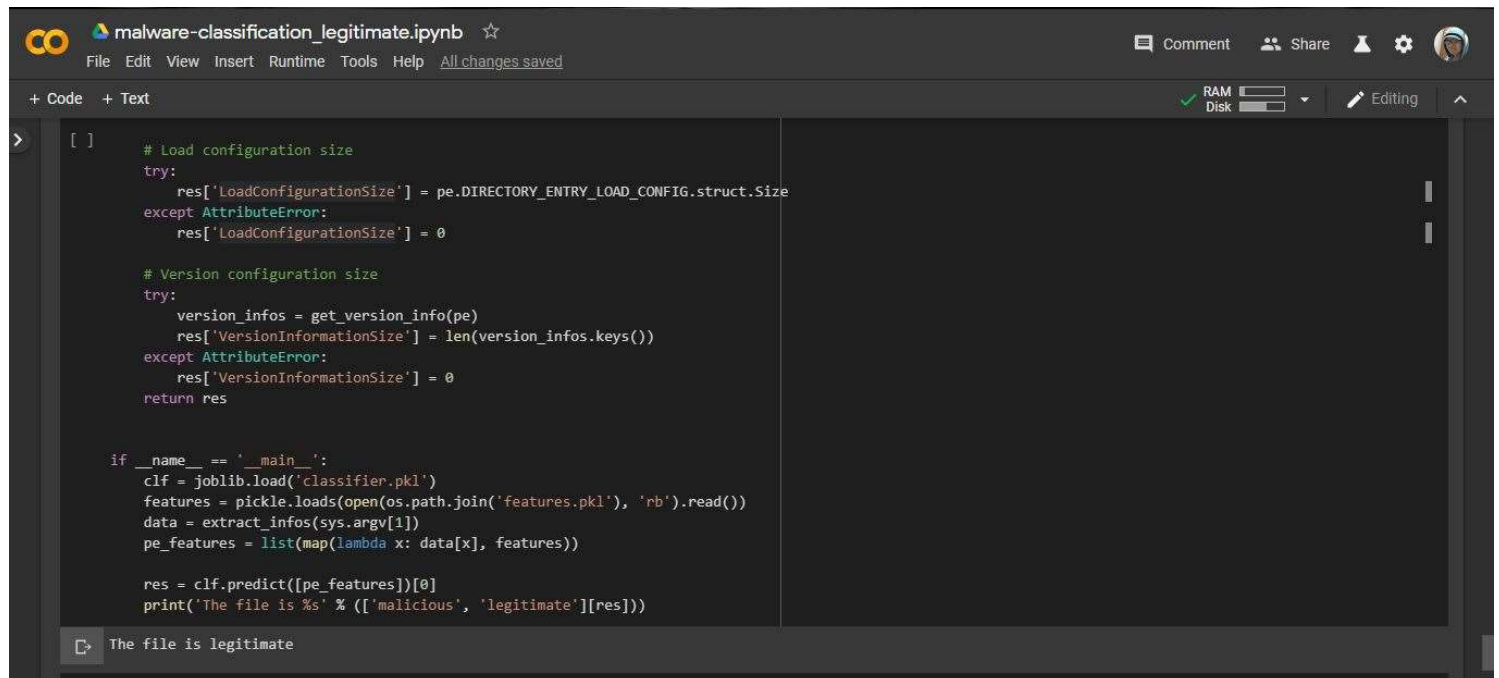
try:
    version_infos = get_version_info(pe)
    res['VersionInformationSize'] = len(version_infos.keys())
except AttributeError:
    res['VersionInformationSize'] = 0
return res

if __name__ == '__main__':
    clf = joblib.load('classifier.pkl')
    features = pickle.loads(open(os.path.join('features.pkl'), 'rb').read())
    data = extract_infos(sys.argv[1])
    pe_features = list(map(lambda x: data[x], features))
    res = clf.predict([pe_features])[0]
    print('The file is %s' % (['malicious', 'legitimate'][res]))

```

7.2 OUTPUTS

7.2.1 Legitimate



The screenshot shows a Jupyter Notebook titled 'malware-classification_legitimate.ipynb'. The code cell contains the following Python code:

```

[ ]
# Load configuration size
try:
    res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
except AttributeError:
    res['LoadConfigurationSize'] = 0

# Version configuration size
try:
    version_infos = get_version_info(pe)
    res['VersionInformationSize'] = len(version_infos.keys())
except AttributeError:
    res['VersionInformationSize'] = 0
return res

if __name__ == '__main__':
    clf = joblib.load('classifier.pkl')
    features = pickle.loads(open(os.path.join('features.pkl'), 'rb').read())
    data = extract_infos(sys.argv[1])
    pe_features = list(map(lambda x: data[x], features))

    res = clf.predict([pe_features])[0]
    print('The file is %s' % (['malicious', 'legitimate'][res]))

```

The output cell at the bottom shows the result of the execution:

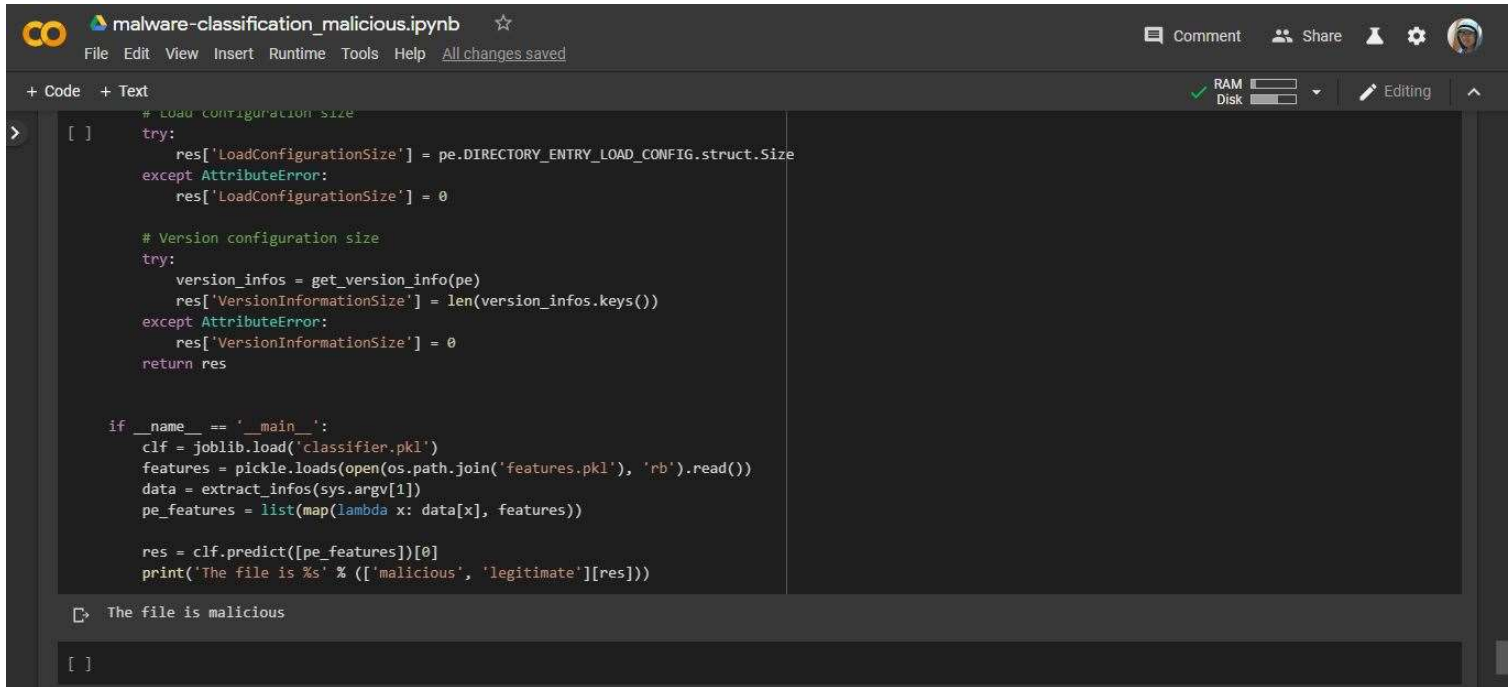
```

The file is legitimate

```

Figure 8 Output

7.2.2 Malicious



The image shows a Jupyter Notebook interface with a dark theme. The title bar at the top reads "malware-classification_malicious.ipynb" and includes a star icon for bookmarks. Below the title bar is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, and Help. A status bar on the right shows "Comment", "Share", and a user profile icon. The main area is divided into a code editor and an output area. The code editor contains a Python script for malware classification. The output area shows the result of the script's execution.

```
[ ] try:
    res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
except AttributeError:
    res['LoadConfigurationSize'] = 0

# Version configuration size
try:
    version_infos = get_version_info(pe)
    res['VersionInformationSize'] = len(version_infos.keys())
except AttributeError:
    res['VersionInformationSize'] = 0
return res

if __name__ == '__main__':
    clf = joblib.load('classifier.pkl')
    features = pickle.loads(open(os.path.join('features.pkl'), 'rb').read())
    data = extract_infos(sys.argv[1])
    pe_features = list(map(lambda x: data[x], features))

    res = clf.predict([pe_features])[0]
    print('The file is %s' % (['malicious', 'legitimate'][res]))
```

The output area shows the result of the script's execution: "The file is malicious".

Figure 9 Output

REFERENCES

- [1] C. Wagner, G. Wagener, R. State, and T. Engel, “Malware analysis with graph kernels and support vector machines,” pp. 63–68, 2009.
- [2] D. D. Lille, B. Coppens, and B. D. Sutter, “Automatically Combining Static Malware Detection Techniques,” 2015.
- [3] Z. Markel and M. Bilzor, “Building a Machine Learning Classifier for Malware Detection,”
- [4] K. Raman, T. Street, and S. Francisco, “Selecting Features to Classify Malware,” pp. 1–5, 2012.
- [5] R. Tian, R. Islam, L. Batten, and S. Versteeg, “Differentiating Malware from Clean ware Using Behavioral Analysis,” pp. 23–30, 2010.
- [6] E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification: A Survey,” no. April, pp. 56–64, 2014.