



# VIT<sup>®</sup>

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

### INFORMATION SECURITY ANALYSIS AND AUDIT

#### PROJECT REPORT

#### **FINAL REPORT**

### **MALWARE ANALYSIS USING MACHINE LERANING**

#### DETAILS OF PROJECT MEMBERS:

- 1). PARTH MAHESHWARI – 19BCT0221 – [parth.maheshwari2019@vitstudent.ac.in](mailto:parth.maheshwari2019@vitstudent.ac.in) – 8401291012
- 2). PRAMIT GUPTA – 19BCT0204 – [pramit.gupta2019@vitstudent.ac.in](mailto:pramit.gupta2019@vitstudent.ac.in) – 7982363745
- 3). AGRIM NAGRANI – 19BCT0217 – [agrim.nagrani2019@vitstudent.ac.in](mailto:agrim.nagrani2019@vitstudent.ac.in) – 9406549900

#### General Course Details:

**Course Title:** Information Security Analysis and Audit

**Course Code:** CSE3501

**Slot:** F2

**Faculty:** Anil Kumar K.

#### Contents:

- 1). Abstract.
- 2). Required Software and Hardware.
- 3). Introduction.
- 4). General Literature Survey.

## **Abstract:**

“People’s computers are not getting more secure. They are getting more infected with viruses. They are getting more under the control of malware.”

-Avi Rubin

Malware is a superset term for several malicious software variants which includes ransomwares, spywares, and viruses. Malware is a fancy word for malicious software. Malware discovery is typically finished with the assistance of hostile to infection programming which thinks about each program in the framework to known malwares. Another way we could distinguish malware is with the assistance of Machine Learning calculations. We could utilize the known highlights of malwares and train a model to anticipate if a program is a malware.

Our purpose for this project is to work on and explore techniques that are used to efficiently perform Malware analysis and detection on enterprise systems to reduce the damage of malware attacks on the systems of public and private organizations.

We will figure different classes of Malware and their propagation techniques. We also aim to highlight different detection vectors for malware along with different tools to analyse and study them.

## **Required Software and Hardware:**

Software:

- Python – (General Python Compiler or Google Collab).
- Data Set – Public Source Data for Experimenting.

Hardware:

- ✦ A general Computer with Internet Connection.

## **Introduction:**

Machine Learning is an appealing apparatus for an essential discovering capacity and advantageous location heuristics. Malware Analysis is the study or process of determining the functionality, origin and potential impact of a given malware sample and extracting as much information from it. The

information that is extracted helps to understand the functionality and scope of malware, how the system was infected and how to defend against similar attacks in future. A simple classification of malware consists of file infectors and stand-alone malware. The problem to be examined involves the high spreading rate of computer malware (viruses, worms, Trojan horses, rootkits, botnets, backdoors, and other malicious software) and conventional signature matching based antivirus systems fail to detect polymorphic and new, previously unseen malicious executables. Thus, ML and AI investigation offers a potential answer for the issues of examination.

## **General Literature Survey:**

### **[1]. Malware Detection using Machine Learning Based Analysis of Virtual Memory Access Patterns:**

*[Xu, Zhixing, et al. "Malware detection using machine learning based analysis of virtual memory access patterns." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017.]*

Malware continues to grow in sophistication. Past proposals for malware detection have primarily focused on software-based detectors which are vulnerable to being compromised. Thus, recent work has proposed hardware-assisted malware detection. In this paper, they introduce a new framework for hardware-assisted malware detection based on monitoring and classifying memory access patterns using machine learning. This provides for increased automation and coverage through reducing user input on specific malware signatures. The key insight underlying the work is that malware must change control flow and/or data structures, which leaves fingerprints on program memory accesses. Building on this, they proposed an online framework for detecting malware that uses machine learning to classify malicious behaviour based on virtual memory access patterns. Although we envision memory accesses will be collected using specialized hardware, this initial evaluation gathers memory accesses using an instrumented version of QEMU 2.2.0.

### **[2]. Analysis of Machine Learning Techniques Used in Behaviour-Based Malware Detection:**

*(Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. "Analysis of machine learning techniques used in behavior-based malware detection." 2010 second international conference on advances in computing, control, and telecommunication technologies. IEEE, 2010. )*

The problem to be examined involves the high spreading rate of computer malware (viruses, worms, Trojan horses, rootkits, botnets, backdoors, and other malicious software) and conventional signature matching-based antivirus systems fail to detect polymorphic and new, previously unseen malicious executables. The data set consists of malware data set and benign instance data set. Both malware and benign instance data sets are in the format of Windows Portable Executable (PE) file binaries. A total of 220 unique malware (specifically 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies Indonesian malware) samples were acquired. The next step is conducting dynamic analysis (behaviour monitoring) of both the malware and benign instance data sets. This process is done by submitting each sample to a free-

online automatic dynamic analysis service: Anubis [6]. Binary submission and execution of Anubis result in the generation of a report file. In this research, all the generated report files were downloaded in XML format. The next step is to conduct learning and classification based on the ARFF files. Machine learning techniques were applied for the learning and classification of the ARFF files.

### **[3]. Malware Analysis and Detection in Enterprise Systems:**

*(Mokoena, Tebogo, and Tranos Zuva. "Malware analysis and detection in enterprise systems." 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). IEEE, 2017.)*

This research investigate techniques that are used to effectively perform Malware analysis and detection on enterprise systems to reduce the damage of malware attacks on the operation of organizations. Two techniques of malware analysis which are Dynamic and Static analysis on two different malware samples. The results showed that Dynamic analysis is more effective than Static analysis. Static analysis of malware is defined as the process of extracting information from malware while it is not running by analysing the code of the malware to determine its true intention. Extraction of information from malware includes the examination of disassembly listings, extracted strings, obtaining signatures of a virus, determining the architecture of the target and compiler that is used, as well as many other characteristics of malware. Dynamic analysis is defined as the process of extracting information from malware when it is executed. This process entails executing the malware artifact in a secure isolated environment, unlike the static analysis process which provides only a view of the malware that is being analysed.

### **[4]. Evaluation on Malware Analysis:**

*(Agrawal, Monika, et al. "Evaluation on malware analysis." International Journal of Computer Science and Information Technologies 5.3 (2014): 3381-3383.)*

Sharing information, communication, socializing, shopping, running businesses and many more are now easily achievable by internet. Despite its vitality, the Internet experiences significant inconveniences, for example, clients' privacy, robbery, fraud, and spamming. Viruses are the Internet's number one opponent and today's viruses are more complex than old era. Malware, a malicious code seeks financial benefit instead of physical harms and some are handled by expert criminal associations. The purpose behind this review paper is to distinguish the techniques and tools utilized by anti-virus to secure Internet's clients from the dangers of malware. Seeing how the malware is constructed and how it is utilized by the attackers is getting essential for system administrators, programming designers and IT security field master. In this evaluation paper they have measured distinctive key issues of malware. We have discussed about malicious code, their effects and detection techniques. Users can be secured by following the protective methods which have discussed above. User must be aware about the malware and their dangerous effects, and they should make their system fully protected so the malware can't inject into their system. Conceivable methods are characterized for the protection of the system.

**[5]. Analysis of features selection and machine learning classifier in android malware detection:**

(Mas' ud, Mohd Zaki, et al. "Analysis of features selection and machine learning classifier in android malware detection." *2014 International Conference on Information Science & Applications (ICISA)*. IEEE, 2014.)

With user are now able to use mobile devices for various purposes such as web browsing, ubiquitous services, online banking, social networking, MMS and etc, more credential information is expose to exploitation. Applying a similar security solution that work in Desktop environment to mobile devices may not be proper as mobile devices have a limited storage, memory, CPU, and power consumption. Hence, there is a need to develop a mobile malware detection that can provide an effective solution to defence the mobile user from any malicious threat and at the same time address the limitation of mobile devices environment. This paper proposes the suitable system call features to be used in machine learning classifying algorithm for classifying the benign and malicious android application. The study has investigated 30 normal android's applications acquired from the google play and 30 infected android's applications acquired from the MalGenome Project. In the Data collection phase (Phase I), each application is running on a real device in an experimental testbed environment. The system call generated by each of the application is captured in a log file using a tool call 'strace'.

**[6]. A Novel Malware Analysis Framework for Malware Detection and Classification using Machine Learning Approach:**

(Sethi, Kamalakanta, et al. "A novel malware analysis framework for malware detection and classification using machine learning approach." *Proceedings of the 19th International Conference on Distributed Computing and Networking*. 2018.)

Digitization of the world is under a serious threat due to the emergence of various new and complex malware every day. Due to this, the traditional signature-based methods for detection of malware effectively become an obsolete method. The efficiency of the machine learning techniques in context to the detection of malwares has been proved by state-of-the-art research works. In this paper, we have proposed a framework to detect and classify different files (e.g., exe, pdf, php, etc.) as benign and malicious using two level classifiers namely, Macro (for detection of malware) and Micro (for classification of malware files as a Trojan, Spyware, Adware, etc.). Our solution uses Cuckoo Sandbox for generating static and dynamic analysis report by executing the sample files in the virtual environment. In addition, a novel feature extraction module has been developed which functions based on static, behavioural and network analysis using the reports generated by the Cuckoo Sandbox. Weka Framework is used to develop machine learning models by using training datasets. The experimental results using the proposed framework shows high detection rate and high classification rate using different machine learning algorithms.

**[7]. Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection:**

(Ahmed, Faraz, et al. "Using spatio-temporal information in API calls with machine learning algorithms for malware detection." *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*. 2009.)

Run-time monitoring of program execution behaviour is widely used to discriminate between benign and malicious processes running on an end-host. Towards this end, most of the existing runtime intrusion or malware detection techniques utilize information available in Windows Application Programming Interface (API) call arguments or sequences. In comparison, the key novelty of our proposed tool is the use of statistical features which are extracted from both spatial (arguments) and temporal (sequences) information available in Windows API calls. We provide this composite feature set as an input to standard machine learning algorithms to raise the final alarm. The results of our experiments show that the concurrent analysis of spatio-temporal features improves the detection accuracy of all classifiers. We also perform the scalability analysis to identify a minimal subset of API categories to be monitored whilst maintaining high detection accuracy.

#### **[8]. Combining File Content and File Relations for Cloud Based Malware Detection:**

(Ye, Yanfang, et al. "Combining file content and file relations for cloud-based malware detection." *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011.)

Resting on the analysis of file contents extracted from the file samples, like Application Programming Interface (API) calls, instruction sequences, and binary strings, data mining methods such as Naive Bayes and Support Vector Machines have been used for malware detection. However, besides file contents, relations among file samples, is always associated with many Trojans, can provide invaluable information about the properties of file samples. In this paper, we study how file relations can be used to improve malware detection results and develop a file verdict system building on a semi-parametric classifier model to combine file content and file relations together for malware detection. To the best of our knowledge, this is the first work of using both file content and file relations for malware detection. Promising experimental results demonstrate that the accuracy and efficiency of our Valkyrie system outperform other popular anti-malware software tools such as Kaspersky Antivirus and McAfee VirusScan, as well as other alternative data mining-based detection systems.

#### **[9]. Automated Microsoft office macro malware detection using machine learning:**

(Bearden, Ruth, and Dan Chai-Tien Lo. "Automated Microsoft office macro malware detection using machine learning." *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017.)

Macro malware in Microsoft (MS) Office files has long persisted as a cybersecurity threat. Though it ebbed after its initial rampages around the turn of the century, it has remerged as threat. Attackers are taking a persuasive approach and using document engineering, aided by improved data mining methods, to make MS Office file malware appear legitimate. Recent attacks have targeted specific corporations with malicious documents containing unusually relevant information. This development undermines the ability of users to distinguish between malicious and legitimate MS Office files and

intensifies the need for automating macro malware detection. This study proposes a method of classifying MS Office files containing macros as malicious or benign using the K Nearest Neighbours machine learning algorithm, feature selection, and TFIDF where p-code opcode n-grams (translated VBA macro code) compose the file features.. Finally, it discusses the challenges automated macro malware detection faces and possible solutions.

#### **[10]. Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection:**

(Sewak, Mohit, Sanjay K. Sahay, and Hemant Rathore. "Comparison of deep learning and the classical machine learning algorithm for the malware detection." *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2018.).

Deep Learning has been showing promising results in various Artificial Intelligence applications like image recognition, natural language processing, language modelling, neural machine translation, etc. Although, in general, it is computationally more expensive as compared to classical machine learning techniques, their results are found to be more effective in some cases. Therefore, in this paper, we investigated and compared one of the Deep Learning Architecture called Deep Neural Network (DNN) with the classical Random Forest (RF) machine learning algorithm for the malware classification.

---

#### **OBJECTIVE:**

The goal of this task is to distinguish malware present statically utilizing AI calculations with the assistance of compact executable (PE). Information is commonly characterized as assortment of data changed over into double structure for handling. Huge information is generally characterized as colossal volume of information that can be utilized for dissecting and growing new advances and dynamic ML models, yet it relies upon different components like volume. Huge information can likewise be utilized for malware recognition utilizing profound learning and neural organizations.

---

#### **PE FILE FORMAT**

The PE record design portrays the prevalent executable arrangement for Microsoft Windows working frameworks, and incorporates executables, powerfully connected libraries (DLLs), and FON textual style documents. The arrangement is at present upheld on Intel, AMD and variations of ARM guidance set structures. The record design is organized with various standard headers followed by at least one segments. Headers incorporate the Common Object File Format (COFF) document header that contains significant data, for example, the sort of machine for which the record is expected, the idea of the record (DLL, EXE, OBJ), the quantity of segments, the quantity of images, and so on The discretionary header recognizes the linker adaptation, the size of the code, the size of instated and

uninitialized information, the location of the passage point, and so on. Information registries inside the discretionary header give pointers to the areas that follow it. This incorporates tables for trades, imports, assets, exemptions, troubleshoot data, declaration data, and movement tables. Thusly, it gives a helpful synopsis of the substance of an executable. At last, the part table diagrams the name, counterbalanced and size of each segment in the PE record. PE segments contain code and introduced information that the Windows loader is to plan into executable or coherent/writeable memory pages, separately, just as imports, fares and assets characterized by the document. Each segment contains a header that determines the size and address. An import address table educates the loader which capacities to statically import.

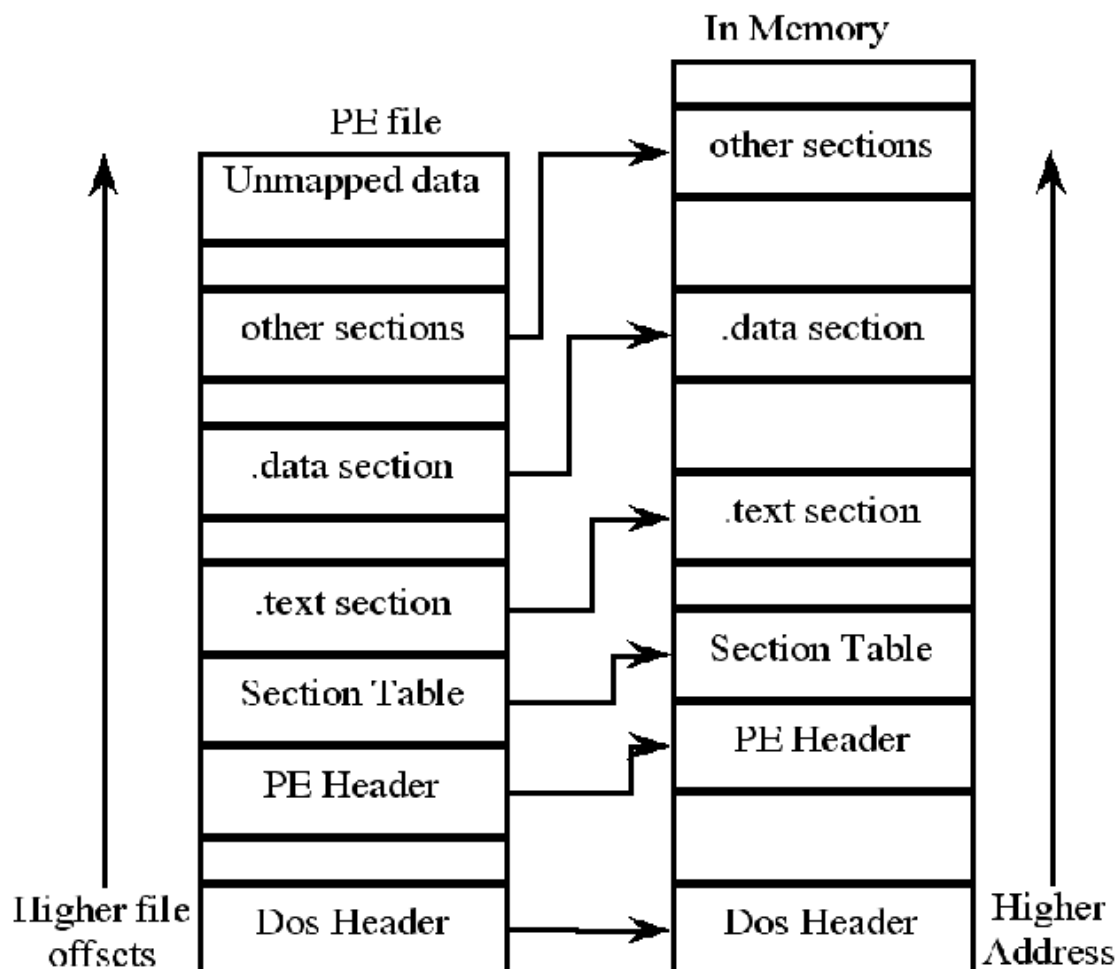


Fig: PE-file structure



## **DATASET:**

We gathered 51,000 special kind Windows PE records from endpoints and removed highlights from them. We utilized our reaping framework to gather 17,000 one of a kind pernicious Windows PE documents from different sources, these PE were confirmed as noxious utilizing Virus Total. We put away all the malware test documents in a spotless climate and removed from the vindictive

PE records similar arrangement of highlights extricated from the kind PE records.

---

## **FEATURES:**

### **1.1 FEATURE EXTRACTION:**

Windows PE records can be either executable documents or documents that contain paired code utilized by other executable records. In this examination, we parse the PE document as per the PE structure and concentrates highlights from the parsed PE record. The removed highlights can isolate into four classes:

### **1.2 PE HEADER INFORMATION:**

PE files have a header that contains information about the PE file and how the OS should execute it. The header can be used to extract a known set of features regardless of the PE size. All Pes have the same header and structure which enables us to collect the same set of features for a wide variety of Pes.

### **1.3 SECTION NAMES AND CHARACTERISTICS:**

Each PE usually has one or more sections that contain the actual body of the PE. Sections can include binary code, strings, configuration, pictures and so on. Standard Pes usually have similar section structures in terms of names, order, and characteristics. Malicious Pes sometimes have different structures due to evasion techniques that are used to avoid detection and classification. The difference in section structure and names is due to several factors including the compiler used to generate the PE. Malicious PE files are often generated by custom compilers which enables them to perform operations on the PE files that are not supported by regular compilers. Examples of such operations are: packing, encryption, manipulation of strings, obfuscation, anti-debugging features, stripping of identifying information, etc. Some section names that are associated with known builders, compilers and packing infrastructure include:

### **1.4 SECTION ENTROPY:**

The PE sections contain data which usually has a known entropy. Higher entropy can indicate packed data. Malicious files are commonly packed to avoid static analysis since the actual code is usually stored encrypted in one of the sections and will only be extracted at runtime. Many malwares use a known packing utility such as: Themida, VMProtect, UPX, MPress, etc. There are

also numerous custom packers that high-end malicious actors use to implement their own custom encryption algorithms. Usually, an entropy level of above 6.7 is considered a good indication that a section is packed. However, there are some custom packers that use padding as part of the encryption process to generate less entropy in the compressed section. Luckily these packers are not very prevalent.

## 1.5 PE IMPORTS:

A PE can import code from another Pes. To do so, it specifies the PE file name and the functions to import. It is important to analyse the imports to get a coherent image of what the PE is doing. Some of the imported functions are indicative of potential malicious operations such as crypto APIs used for unpacking/encryption or APIs used for anti-debugging. Some examples of potential malicious imports:

Import Names	Potential Malicious Usage
KERNEL32.DLL!MapViewOfFile	Code Injection
KERNEL32.DLL!IsDebuggerPresent	Anti-Debugging
KERNEL32.DLL!GetThreadContext	Code Injection
KERNEL32.DLL!ReadProcessMemory	Code Injection
KERNEL32.DLL!ResumeThread	Code Injection
KERNEL32.DLL!ResumeThread	Code Injection
KERNEL32.DLL!WriteProcessMemory	Code Injection
KERNEL32.DLL!SetFileTime	Stealth

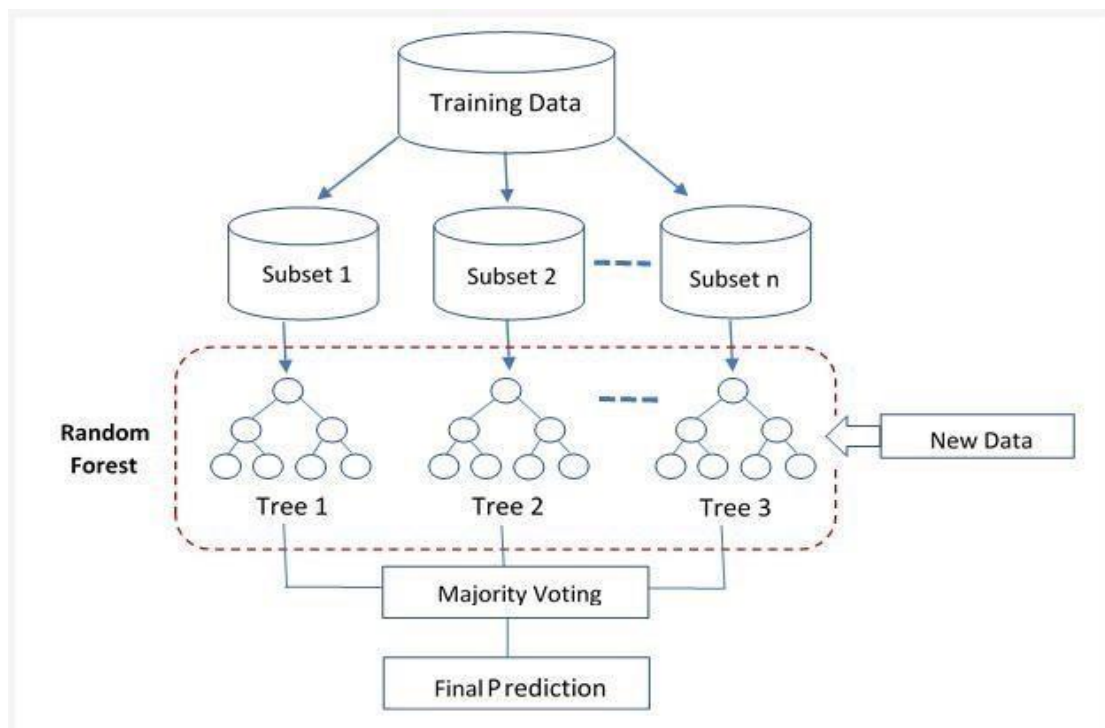
USER32.DLL!SetWindows	API Hooking
KERNEL32.DLL!MapView	Code Injection
ADVAPI32.DLL!CryptGen	Encryption
ADVAPI32.DLL!CryptAcqu	Encryption
KERNEL32.DLL!CreateTod	Process Enumeration
ADVAPI32.DLL!OpenThre	Token Manipulation
ADVAPI32.DLL!Duplicate	Token Manipulation
CRYPT32.DLL!CertDupli cateCertificateContext	Encryption

## FEATURE SELECTION:

Feature selection is a pre-processing technique used in machine learning to obtain an optimal subset of relevant and non-redundant features to increase learning accuracy. Feature selection is necessary because the high dimensionality and vast amount of data pose a challenge to the learning process. In the presence of many irrelevant features, learning models tend to become computationally complex, over fit, become less comprehensible and decrease learning accuracy. Feature selection is one effective way to identify relevant features for dimensionality reduction. However, the advantages of feature selection come with the extra effort of trying to get an optimal subset that will be a true representation of the original dataset. In our study, we extracted 191 features from each PE file.

## ALGORITHMS:

In our investigation, we utilized the Random woodland AI calculation for a few reasons. Irregular woodland permits us to remove the significance of highlights and increase some understanding into the discovery cycle. Irregular timberland has techniques to deal with uneven datasets. Since the quantity of considerate PE records isn't equivalent to the quantity of vindictive PE documents, we can utilize these techniques to conquer the irregularity in our informational index. Moreover, irregular timberland is an outfit learning calculation and along these lines can forestall overfitting. In this segment, we quickly portrayed the arbitrary woods calculation that was applied. Irregular backwoods is a gathering learning calculation, that comprises of various choice trees. The trees are developed as follows: Each tree develops by utilizing subsets of tests that are chosen arbitrarily from the first preparing information.



## ALGORITHM SETUP:

---

As mentioned, in this study we incorporated feedback from human analysts to verify files identified by the classifier were indeed malicious to adjust the learning process. In each cycle, we train the random forest on features extracted from one time-period and malicious PE files. Then the algorithm generation classifier is applied during the following period to classify the testing PE files. The training data set include features extracted from ~12,000 different benign PE files and from 12,750 malicious PE files. The testing PE files include ~5000 benign PE files and 4,250 malicious PE files.

## MODELS:

### 1.6 RANDOM FOREST

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. In our case Random Forest produces a score of around 0.994 and was the best model.

### 1.7 ADABOOST

AdaBoost, short for Adaptive Boosting, is a machine learning metaalgorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. In our case, AdaBoost produces a score of 0.9860 and was one of the better performers.

## 1.8 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. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. It produces the score of 0.9890 in our case.

## 1.9 SVM

In machine learning, support-vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. It produces a score of around 0.698 in our case.

## 1.10 LINEAR REGRESSION

Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x). When there is a single input variable (x), the method is referred to as simple linear regression. When there are multiple input variables, literature from statistics often refers to the method as multiple linear regression. In our case it produces a score of around 0.58935.

```
results = {}
for i in model:
    clf = model[i]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(i, score))
    results[i] = score
```

```
RandomForest : 0.9942412169503803
Adaboost : 0.9860557768924303
GradientBoosting : 0.9890257153205361
SVM : 0.6985512495472654
LinearRegression : 0.5893568800313873
```

Fig.3 Accuracy Comparison of all models

## EXPERIMENTATION AND RESULTS:

### 1.11 CODE

- First, we run leaning.py file using command python learning.py and checker.py files

```
Learner.py
import pandas as pd #used for DATAFrames and DataFrames can hold different
types data of multidimensional arrays.
import numpy as np#Numpy provides robust data structures for efficient computa
tion of multi-dimensional arrays & matrices.
import pickle import sklearn.ensemble as ske from
sklearn import model_selection, tree, linear_model
from sklearn.feature_selection import SelectFromModel
import joblib from sklearn.naive_bayes import
GaussianNB from sklearn.metrics import
confusion_matrix

data = pd.read_csv('data.csv', sep='|') #generate df as data
X = data.drop(['Name', 'md5', 'legitimate'], axis=1).values #now dropping
some columns as axis 1(mean column) and will show the values in the rows y
= data['legitimate'].values #values of legitimate data
```

strong learner (usually a decision tree) and iteratively create a weak learner that is added to the strong learner. They differ on how they create the weak learners during the iterative process.

```
"GNB": GaussianNB()
#Bayes theorem is based on conditional probability. The conditional probability helps us calculating the probability that something will happen
} results = {} print("\nNow testing algorithms") for
algo in algorithms: clf = algorithms[algo]
clf.fit(X_train, y_train)#fit may be called as 'trained'
score = clf.score(X_test, y_test) print("%s : %f %" %
(algo, score*100)) results[algo] = score
winner = max(results, key=results.get) print('\nWinner algorithm is %s with
a %f %% success' % (winner, results[winner]*100))

# Save the algorithm and the feature list for later predictions print('Saving
algorithm and feature list in classifier directory...')
joblib.dump(algorithms[winner], 'classifier/classifier.pkl')#Persist an
arbitrary Python object into one file.
open('classifier/features.pkl', 'wb+').write(pickle.dumps(features))
#joblib works especially well with NumPy arrays which are used by sklearn so
depending on the classifier type you use you might have performance and size
benefits using joblib.Otherwise pickle does work correctly so saving a
trained classifier and loading it again will produce the same results no
matter which of the serialization libraries you use print('Saved')

# Identify false and true positive
rates clf = algorithms[winner] res =
clf.predict(X_test)
mt = confusion_matrix(y_test, res)
#A confusion matrix, also known as an error matrix,[4] is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning.
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)))
```

---



Checker.py

```
#!/usr/bin/python2
#pefile is a Python module to read and work with PE (Portable Executable) file
s.
import pefile
import os import
array import math
import pickle
import joblib
import sys import
argparse import
shutil, time
import re
import pandas as pd
    from flask import Flask, request, jsonify,
render_template,abort,redirect,url_ for from werkzeug.utils import
secure_filename
from sklearn.ensemble import RandomForestClassifier
def
cutit(s,n):
return s[n:]
    app =
Flask(__name__)

@app.route('/') def
home():
    return render_template('index.html')

@app.route('/uploader', methods = ['GET',
'POST']) def upload_file():    if request.method
== 'POST':        f = request.files['file']
        f.save(secure_filename(f.filename))
        #####
        # Load classifier        clf =
joblib.load(os.path.join(os.path.dirname(os.path.realpath(__file__))
,'classifier/classifier.pkl'))
        features =pickle.loads(open(os.path.join(os.path.dirname(os.path.realpath(
__file__))
                                ,'classifier/features.pkl'),'rb+').read
())
        #####
```

```

resource_lang.data.struct.S size)
resource_lang.data.struct.Size
get_entropy(data)

resources.append([entropy,
size])
except Exception as e:
return
resources
return resources

```

```

def get_version_info(pe):
    """Return version
    infos"""
    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(fpath)
    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.SizeOfUninitializedDat a
    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
    # Sections      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)

    # Imports
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,
im ports)))    except AttributeError:    res['ImportsNbDLL'] = 0
res['ImportsNb'] = 0    res['ImportsNbOrdinal'] = 0
    # Exports
    try:
        res['ExportNb'] =
len(pe.DIRECTORY_ENTRY_EXPORT.symbols)    except
AttributeError:    # No export
        res['ExportNb'] = 0
    # Resources    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

    # 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__':
    app.run(debug = True)

```

## Checker.py

### 1.12 READING THE DATASET

```
[ ] dataset = pandas.read_csv("data.csv",sep='|', low_memory=False)
```

```
[ ] dataset.head()
```

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLink
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	
2	setup.exe	4d92f518527353c0db88a70fdcdcf390	332	224	3330	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	

### 1.13 FEATURE EXTRACTION

```
[ ] dataset.groupby(dataset['legitimate']).size()
```

```

legitimate
0    96724
1    41323
dtype: int64

```

```

▶ X = dataset.drop(['Name','md5','legitimate'],axis=1).values
  y = dataset['legitimate'].values

```

```

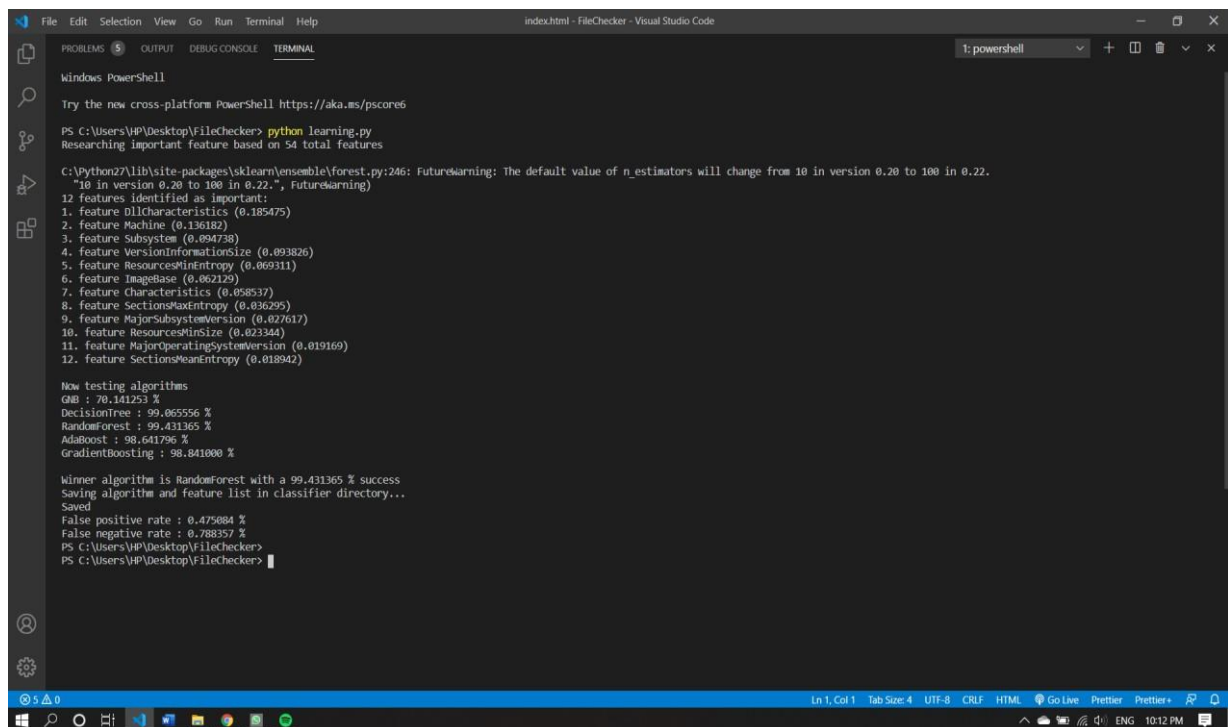
[ ] extratrees = ek.ExtraTreesClassifier().fit(X,y)
  model = SelectFromModel(extratrees, prefit=True)
  X_new = model.transform(X)
  nbfeatures = X_new.shape[1]

```

```
▶ nbfeatures
```

```
13
```

## 1.14 FEATURE SELECTION



```
index.html - FileChecker - Visual Studio Code
1: powershell

Windows PowerShell

Try the new cross-platform PowerShell https://aka.ms/powershell

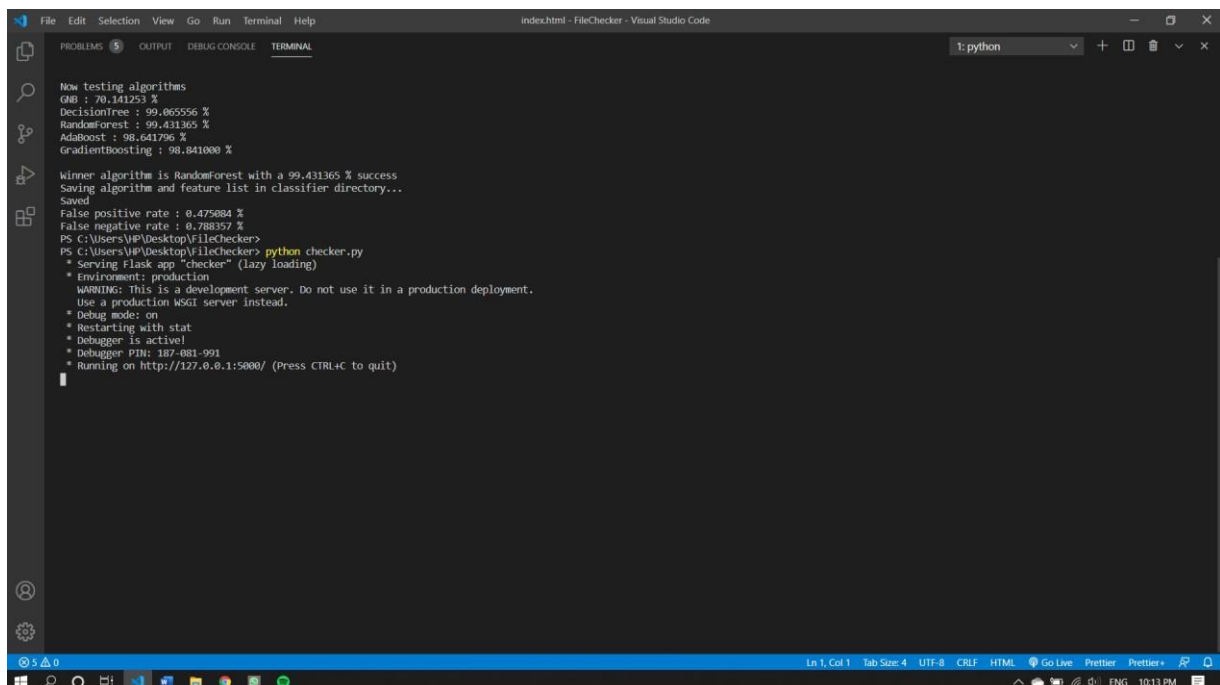
PS C:\Users\HP\Desktop\FileChecker> python learning.py
Researching important feature based on 54 total features

C:\Python27\Lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
12 features identified as important:
1. feature DLLCharacteristics (0.185475)
2. feature Machine (0.136182)
3. feature Subsystem (0.094738)
4. feature VersionInformationSize (0.093826)
5. feature ResourceMinEntropy (0.069311)
6. feature ImageBase (0.062129)
7. feature Characteristics (0.058537)
8. feature SectionsMaxEntropy (0.036295)
9. feature MajorSubsystemVersion (0.027617)
10. feature ResourceSize (0.023344)
11. feature MajorOperatingSystemVersion (0.019169)
12. feature SectionsMeanEntropy (0.018942)

Now testing algorithms
GB : 70.141253 %
DecisionTree : 99.065556 %
RandomForest : 99.431365 %
AdaBoost : 98.641796 %
GradientBoosting : 98.841000 %

Winner algorithm is RandomForest with a 99.431365 % success
Saving algorithm and feature list in classifier directory...
Saved
False positive rate : 0.475084 %
False negative rate : 0.788357 %
PS C:\Users\HP\Desktop\FileChecker>
PS C:\Users\HP\Desktop\FileChecker>
```

## 1.15 MODEL DEVELOPMENT



```
index.html - FileChecker - Visual Studio Code
1: python

Now testing algorithms
GB : 70.141253 %
DecisionTree : 99.065556 %
RandomForest : 99.431365 %
AdaBoost : 98.641796 %
GradientBoosting : 98.841000 %

Winner algorithm is RandomForest with a 99.431365 % success
Saving algorithm and feature list in classifier directory...
Saved
False positive rate : 0.475084 %
False negative rate : 0.788357 %
PS C:\Users\HP\Desktop\FileChecker>
PS C:\Users\HP\Desktop\FileChecker> python checker.py
* Serving Flask app "checker" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 187-081-991
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

## 1.16 FLASK APP FOR MALWARE DETECTION

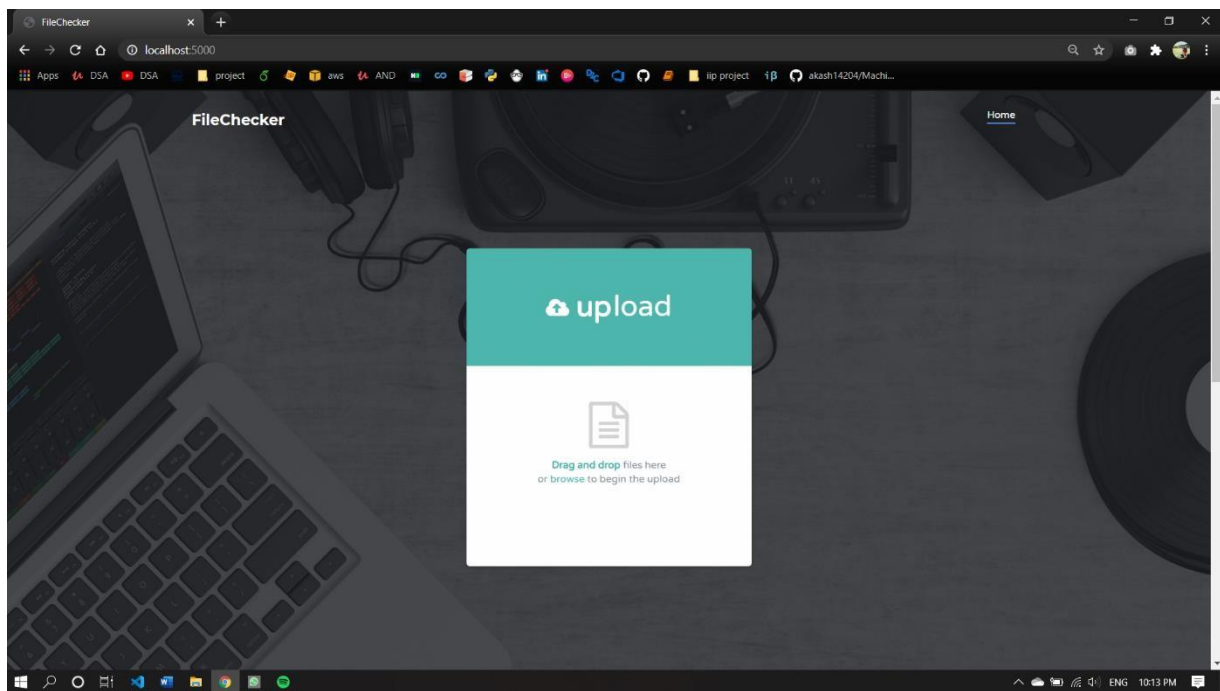
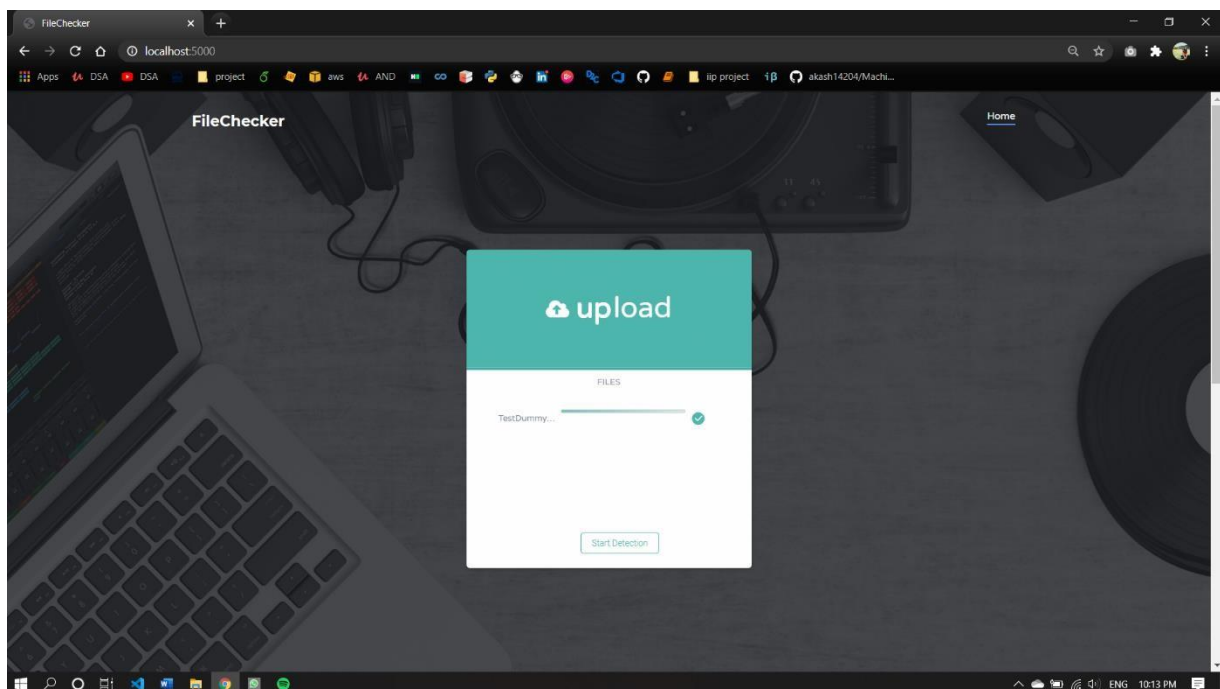
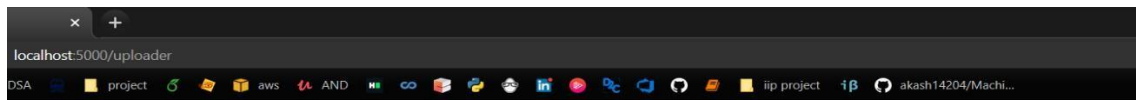


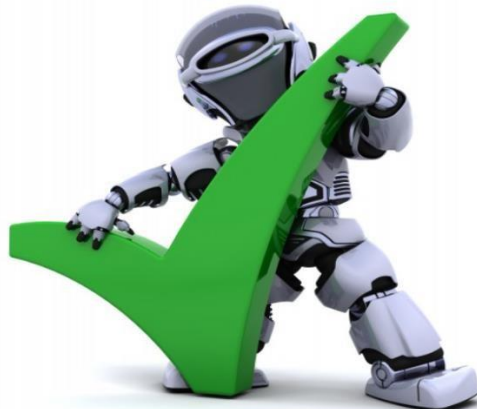
Fig.8



**After testing the file through our machine learning model, it will provide the result whether the software is malicious or legitimate.**



This Software is  
legitimate



This Software is  
malicious



## CONCLUSION:

From above analysis and results we can say that irregular words are probably the best model for our undertaking. From these results, we can assume that the most accommodating segment for perceiving compassionate PE records and dangerous PE archives is the best entropy of all the PE region entropies. This observation fits with our doubts that high entropy isn't essential with kind PE records. Moreover, apparently there is remarkable centrality to the imprint status of the record. To be explicit, if the PE record isn't checked or it is set apart with an unverified imprint there is a high probability that it is a malicious PE document. The following most critical features are related to zone names and



approvals. Malware often uses squeezing procedures to go without being recognized by antivirus marks. This results in nonstandard fragments names and form approvals. We also notice that the classes of the questionable import influenced the model precision. In these features, we accumulated differing questionable Programming interface works by characterizations, for instance, shirking, encryption, distant task, etc. In each social occasion, there can be a couple of limits from different DLLs. This allowed us to get comfortable with the noxious activity without overfitting to express limits.

## REFERENCES:

---

- [1] Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. "Analysis of machine learning techniques used in behavior-based malware detection." *2010 second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010.
- [2] Mokoena, Tebogo, and Tranos Zuva. "Malware analysis and detection in enterprise systems." *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE, 2017.
- [3] Xu, Zhixing, et al. "Malware detection using machine learning based analysis of virtual memory access patterns." *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017.
- [4] Mas' ud, Mohd Zaki, et al. "Analysis of features selection and machine learning classifier in android malware detection." *2014 International Conference on Information Science & Applications (ICISA)*. IEEE, 2014.
- [5] Agrawal, Monika, et al. "Evaluation on malware analysis." *International Journal of Computer Science and Information Technologies* 5.3 (2014): 3381-3383.
- [6] Ahmed, Faraz, et al. "Using spatio-temporal information in API calls with machine learning algorithms for malware detection." *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*. 2009.
- [7] Sethi, Kamalakanta, et al. "A novel malware analysis framework for malware detection and classification using machine learning approach." *Proceedings of the 19th International Conference on Distributed Computing and Networking*. 2018.
- [8] Ye, Yanfang, et al. "Combining file content and file relations for cloud based malware detection." *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011.

- [9] Bearden, Ruth, and Dan Chai-Tien Lo. "Automated microsoft office macro malware detection using machine learning." *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017.
- [10] Sewak, Mohit, Sanjay K. Sahay, and Hemant Rathore. "Comparison of deep learning and the classical machine learning algorithm for the malware detection." *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2018.

---

---