MALWARE DETECTION AND ANALYSIS

PROJECT REPORT
E1 SLOT

Submitted by

MOHD UMAR 18BCE0196

ROHIT MEHTA 18BCE0199

LAKSHIT DHANUKA 18BCE0217

NISHITH SURANA 18BCE0220

Course Code: **CSE4003**

Course Title: **Cyber Security**

Under the guidance of

PROF. MANIKANDAN K

B.Tech

in

Computer Science and Engineering

VIT®

Vellore Institute of Technology

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

1. ACKNOWLEDGEMENT

A most profound appreciation and true gratitude to **Prof. MANIKANDAN K** in helping us complete our Project with several learning results. We feel profoundly obliged to thank the SCOPE (School of Computer Science and Engineering) Department and the VIT University for their administrations delivered and for allowing us a chance to make such activities alongside our examinations at the University.

18BCE0196 - MOHD UMAR

18BCE0199 - ROHIT MEHTA

18BCE0217 - LAKSHIT DHANUKA

18BCE0220 – NISHITH SURANA

2. ABSRACT

This project aims to present the functionality and accuracy of five different machine learning algorithms to detect whether an executable file is malicious or legitimate. Malware discovery is typically consummated with the assistance of hostile to infection programming which believe about each program in the substructures to known malwares. One. We could utilize the known highlights of malwares and train a model to anticipate if a program is a malware. Along these lines, we will utilize Machine Learning calculations to anticipate if a specific program is a malware or not. Information has been soaring since the appearance of web. Additionally, the kind of information is changing quickly with time. Henceforth, we have to discover devices that could cycle and help in examining various sorts of information effectively and rapidly as the datasets of genuine world have gigantic information storehouses. In this task we plan to do as such by utilizing Ember dataset which is an open Dataset for Training Static PE Malware Machine Learning Models. The dataset incorporates highlights separated from 1.1M double records: 900K preparing tests (300K malevolent, 300K favourable, 300K unlabelled) and 200K test tests (100K noxious, 100K kind). To go with the dataset, we likewise discharge open source code for extricating highlights from extra parallels so extra example highlights can be attached to the dataset. This dataset makes up for a shortcoming in the data security AI people group: an amiable/pernicious dataset that is enormous, open and general enough to cover a few fascinating use cases.

Table of Contents

3. INTRODUCTION

Malware is any software purposely designed to cause defilement to a computer, server, client or computer network. This single term circumscribe Viruses, Trojans, Worms etc., whereas Malware Analysis allude to the process of understanding the behaviour and purpose of a suspicious file or URL. Machine Learning can be an enchanting equipment for either an essential discovery capacity or analysis of Malware. Machine learning models consequently misuse complex connections between document ascribes in preparing information that are separating among vindictive and generous examples. Besides, appropriately regularized AI models sum up to new examples whose highlights and names follow a comparative dispersion to the preparation information. In any case, it is generally identified in the security network that the current mark-based way to deal with infection identification is not, at this point satisfactory. 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. Static executable investigation offers a potential answer for the issues of dynamic examination. Static investigation takes a gander at the structures inside the executable that are fundamental with the end goal for it to run. Since these structures are commanded by the record type, they can't be eliminated, encoded (in spite of the fact that their code substance might be), or muddled without any problem. One proposed approach (solution) is by using automatic dynamic (behaviour) malware analysis combined with data mining tasks, such as, machine learning (classification) techniques to achieve effectiveness and efficiency in detecting malware. A survey on different machine learning methods that were proposed for malware detection is given in Additionally, on the grounds that it just includes parsing structures, it is substantially less computationally costly than dynamic examination. Some exploration has just been done into static executable examination for Windows compact executable (PE) documents. Of specific note, the last 4 or 5 years has seen various progressed tenacious danger (APT) malware crusades focused on explicitly at Macs. Investigating instruments to resist these dangers currently guarantees that we will have the option to all the more likely handle the expanding danger later on.

4. LITERATURE SURVEY

[1] Analysis of Machine Learning Techniques Used in Behaviour-Based Malware Detection: 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 (behavior monitoring) of both the malware and benign instance data sets. This process is done by submitting each and every 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.

[2]Malware Analysis and Detection in Enterprise Systems: The purpose of this research is to investigate techniques that are used in order 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 analyzed.

[3] Malware Detection using Machine Learning Based Analysis of Virtual Memory Access Patterns: Malicious software, referred to as 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.

[4]Analysis of features selection and machine learning classifier in android malware detection: The proliferation of Android-based mobile devices and mobile applications in the market has triggered the malware author to make the mobile devices as the next profitable target. 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 propose the suitable system call features to be used in machine learning classifying algorithm for classifying the benign and malicious android application. For the purpose of this research, the study has investigated 30 normal android's application 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.

[5] Evaluation on Malware Analysis: In this technological era everyone's life is influenced by Internet. It plays an essential role in today's life style and businesses. Sharing information, communication, socializing, shopping, running businesses and many more are now easily achievable by internet. In spite of 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 then old era. They utilize numerous methods to escape of the anti-virus programs and have a tendency to work silently at the backdrop. 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. This evaluation work is embraced by a portion of the researchers which is in advancement for the planning of the security structure from protecting your system to get infected with malware.

[6] Using Spatio-Temporal Information in API Calls with Machine Learning Algorithms for Malware Detection: Run-time monitoring of program execution behavior is widely used to discriminate between benign and malicious processes running on an end-host. Towards this end, most of the existing run-time 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.

[7] A Novel Malware Analysis Framework for Malware Detection and Classification using Machine Learning Approach: Nowadays, the 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 classifier 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, behavioral 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.

[8]Combining File Content and File Relations for Cloud Based Malware Detection: Due to their damages to Internet security, malware (such as virus, worms, trojans, spyware, backdoors, and rootkits) detection has caught the attention not only of anti-malware industry but also of researchers for decades. 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. A comprehensive experimental study on a large collection of PE files obtained from the clients of anti-malware products of Comodo Security Solutions Incorporation is performed to compare various malware detection approaches. 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. Our system has already been incorporated into the scanning tool of Comodo's Anti-Malware software.

[9] Automated Microsoft office macro malware detection using machine learning: 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 re-emerged 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. This study achieves a 96.3% file classification accuracy on a sample set of 40 malicious and 118 benign MS Office files containing macros, and it demonstrates the effectiveness of this approach as a potential defence against macro malware. 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: Recently, Deep Learning has been showing promising results in various Artificial Intelligence applications like image recognition, natural language processing, language modeling, 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. We studied the performance of the classical RF and DNN with 2, 4 & 7 layers architectures with the four different feature sets, and found that irrespective of the features inputs, the classical RF accuracy outperforms the DNN.

5. PROBLEM STATEMENT

Malwares are a prominent form of attack in the Cyber security domain which harms the user data, network, server and so on. Detecting these Malwares is a major challenge as the Malwares authors always come up with unique ways of hiding the source and leveraging the weaknesses of the existing system. We have to identify the parameters and patterns in the PE file which is a challenging task. In this project, we use Machine Learning to detect Malicious file.

6. OBJECTIVE

The scope of this project circumscribe the following objectives:-

1. Identify the parameters that distinguish a Malicious PE file from a genuine one.
2. Identify patterns in these PE files.
3. Train various classifier models to predict the legitimacy of PE file .
4. Define relevant metrics to gauge the accuracy of the model.
5. Factoring in the statistical records and make predictions on the given data set and identify the malicious PE file.

7. 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. An assets area may contain assets, for example, needed for UIs: cursors, textual styles, bitmaps, symbols, menus, and so on A fundamental PE record would regularly contain a .text code area and at least one information areas (.information, .rdata or .bss). Movement tables are normally put away in a .reloc area, utilized by the Windows loader to reassign a base location from the executable's favored base. A .tls area contains unique string neighborhood stockpiling (TLS) structure for putting away string explicit nearby factors, which has been misused to divert the passage purpose of an executable to initially check if a debugger or different investigation device are being run. Segment names are discretionary from the viewpoint of the Windows loader, yet explicit names have been embraced by point of reference and are overwhelmingly normal. Packers may make new segments, for instance, the UPX packer makes PX1 to house stuffed information and a vacant area UPX0 that saves a location range for runtime unloading.

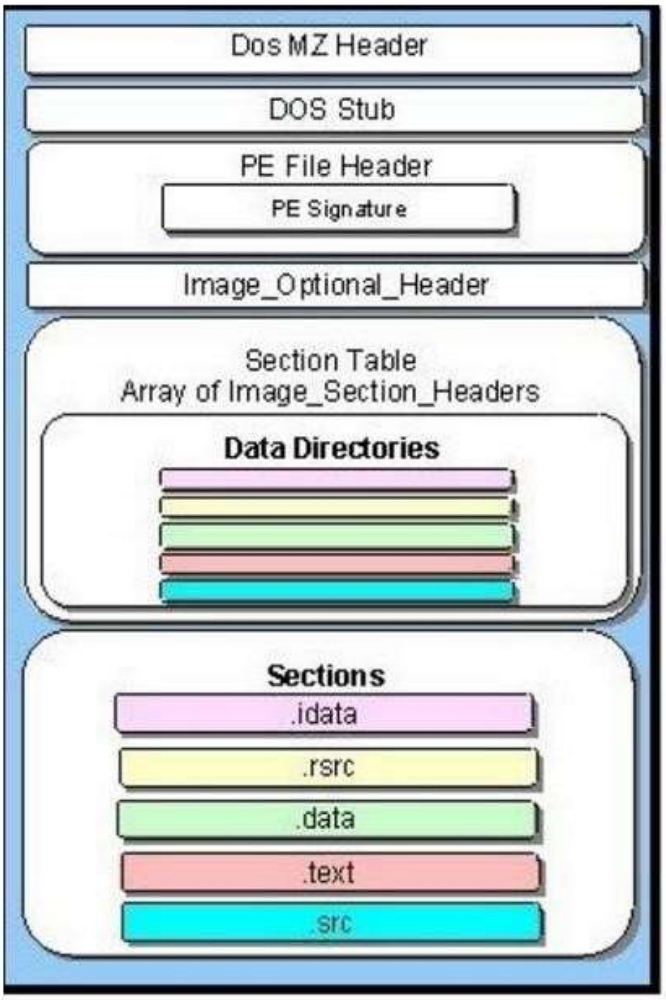


Figure 1 PE file Format

8. 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.

9. FEATURES

9.1 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.

9.2 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.

9.3 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:

Section Name	Builders/Compilers/Packing Infrastructure
Mpres	Mpres Packer
Upx	UPX
Tsuarch	TSULoader
Petite	Petit packer
Rmnet	Ramnit Packer

Nsp	NsPack packer
Aspack	Aspack packer
Themeda	Themida
Adata	Armaddillo packer
Vmp	VMProtect
Mew	MEW Packer
Neolit	ImpRec-created section
Newsec	LordPE Builder
Taz	PESPIN
gfids	Visual Studio (14.0)
stab	Haskell compiler (GHC)

9.4 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.

9.5 PE IMPORTS

A PE can import code from other Pes. To do so, it specifies the PE file name and the functions to import. It is important to analyze 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 example of potential malicious imports:

Import Name	Potential Malicious Usage
KERNEL32.DLL!MapView0	Code Injection
KERNEL32.DLL!IsDebuggerPresent	Anti-Debugging
KERNEL32.DLL!GetThread	Code Injection
KERNEL32.DLL!ReadProcess	Code Injection
KERNEL32.DLL!ResumeThread	Code Injection
KERNEL32.DLL!ResumeThreadPresent	Code Injection
KERNEL32.DLL!WriteProcess	Code Injection
KERNEL32.DLL!SetfileTime	Stealth
USER32.DLL!SetWWindows	API Hooking
KERNEL32.DLL!MapViews	Code Injection
ADVAPI31.DLL!CryptGenFiles	Encryption
ADVAPI31.DLL!CryptAcquiredFiles	Encryption
KERNEL32.DLL!CreateTockens	Process Enumeration
ADVAPI31.DLL!OpenThread	Token Manipulation
ADVAPI31.DLL!DuplicateThread	Token Manipulation
CRYPT32.DLL!CerDuplicateCertificateContext	Encryption

10. 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. In order to reduce the dimensionality of our dataset and improve the performance of the machine learning algorithm, we used Fast Correlation-based Feature Selection (FCBF) method. FCBF is a feature selection method that starts with a full set of features and uses symmetrical uncertainty to analyze the correlation between features and remove redundant features. FCBF consists of two parts:

- Select subset of Machine Learning Cyber Security F subset features that are relevant to the target class. Remove redundant features and save only predominant ones in Machine Learning Cyber Security F subset a.
- The FCBF method stops when there are no redundant features left to eliminate. The FCBF runs, in general, significantly faster than other subset selection methods, and is scalable and independent of a learning algorithm. As a result, feature selection needs to be performed only once. After we applied the FCBF method, each PE file is represented by 55 relevant, non-redundant features.

11. ALGORITHMS MODELS

11.1 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.

11.2 ADABOOST

AdaBoost, short for Adaptive Boosting, is a machine learning that 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.

11.3 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.

11.4 DECISION TREE

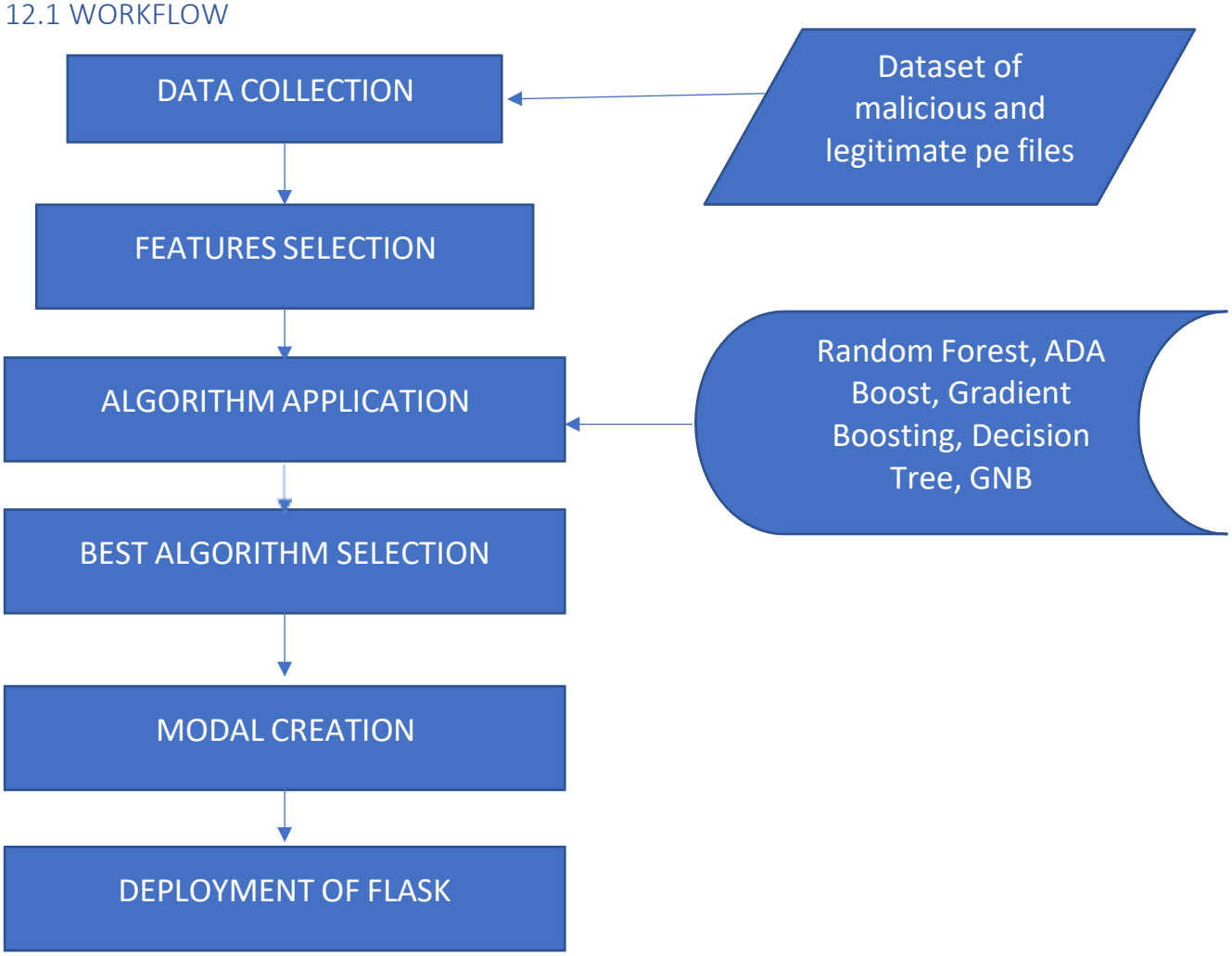
Decision Tree is an ensemble learning method for classification, regression and other tasks. It works by classifying various instances by sorting the down the tree from the root to some leaf node, which provides the class of the instance. Each node tests the attribute of an instance whereas each branch descending from the node specifies one possible value of this attribute.

11.5 GNB

Gaussian Naïve Bayes Classifier is a subset of classification algorithms based on Bayes' Theorem. It works on assumption that every pair of features being classified is independent of each other and affects the dependent variable equally. These assumptions are not generally correct in real-world situations but often works well in practice.

12. PROCEDURES FOR PROPOSED METHOD WITH AN ALGORITHM

12.1 WORKFLOW



12.2 ALGORITHM:-

- Collection of data. The data is taken from VirusShare and has 138047 instances across 54 different attributes.
- As many features are not helpful to differentiate between legitimate and malicious file, the important features are selected using tree classifier.
- A total of 5 supervised classification namely Random Forest, ADA Boost, Gradient Boosting, Decision Tree, GNB are used.
- The best algorithm(Random Forest) is decided on the basis of accuracy of different algorithm.
- Saving the algorithm(classifier.pkl) and feature list(features.pkl) for later use.
- Deploying flask web framework.

13. IMPLEMENTATION

First learning.py python file. This python file is for important features selection, training, and creating machine learning model. For selecting features **Extra Tree Classifier** is used, it is a type of classifier with concept similar to Random Forest and only difference being in the manner of its construction of decision tree.

The output of this file gives the list of important features and the best model for the undertaking. It also saves the list of important features as features.pkl and best algorithm as classifier.pkl with the help of library called pickle

```
from sklearn.metrics import confusion_matrix

data = pd.read_csv('data.csv', sep='|')
X = data.drop(['Name', 'md5', 'legitimate'], axis=1).values
y= data['legitimate'].values

print('Researching important feature based on %i total features\n' % X.shape[1])
# Feature selection using Trees Classifier
fsel = ske.ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(fsel, prefit=True)
X_new = model.transform(X)#now features are only 9
nb_features = X_new.shape[1]#will save value 13 as shape is (138047, 13)



X_train, X_test, y_train, y_test = model_selection.train_test_split(X_new, y ,test_size=0.2)
features = []

print('%i features identified as important:' % nb_features)


#important features sored
indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
for f in range(nb_features):
    print("%d. feature %s (%f)" % (f + 1, data.columns[2+indices[f]], fsel.feature_importances_[indices[f]]))

# mean adding to the empty 'features' array the 'important features'
for f in sorted(np.argsort(fsel.feature_importances_)[::-1][:nb_features]):
    features.append(data.columns[2+f])

#Algorithm comparison
algorithms = {
        "DecisionTree": tree.DecisionTreeClassifier(max_depth=10),
        #The max_depth parameter denotes maximum depth of the tree.

        "RandomForest": ske.RandomForestClassifier(n_estimators=50),#In case, of random forest, these ensemble
                                                                    #created decision trees Each decision tree
```

Figure 2 Code snip of learning.py

```
(base) F:\STUDY\SEM 6\CyberSec\FileChecker\FileChecker>python learning.py
Researching important feature based on 54 total features

12 features identified as important:
1. feature DllCharacteristics (0.180327)
2. feature Machine (0.107003)
3. feature Characteristics (0.104303)
4. feature VersionInformationSize (0.062060)
5. feature Subsystem (0.059317)
6. feature SectionsMaxEntropy (0.053374)
7. feature ImageBase (0.051660)
8. feature SizeOfOptionalHeader (0.047946)
9. feature MajorSubsystemVersion (0.043654)
10. feature ResourcesMinEntropy (0.036848)
11. feature ResourcesMaxEntropy (0.036760)
12. feature MajorOperatingSystemVersion (0.022898)

Now testing algorithms
DecisionTree : 99.040203 %
RandomForest : 99.344440 %
GradientBoosting : 98.743209 %
AdaBoost : 98.576603 %
GNB : 70.043463 %

Winner algorithm is RandomForest with a 99.344440 % success
Saving algorithm and feature list in classifier directory...
Saved
False positive rate : 0.548144 %
False negative rate : 0.906673 %
```

Figure 3 Implementation of learning.py

After the model is saved, flask is deployed by running checker.py python file. Flask act as the backend server, whereas for front-end HTML, CSS and JavaScript is being deployed. The checker.py applies the features.pkl and classifier.pkl on the file submitted. To give the result the user is sent to result.html.

```
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())
     ########################################
     #tweet = request.form['tweet']
     #tweet=cutit(f.filename, 12)
    tweet =f.filename
    print(tweet);
     ########################################
    data = extract_infos(tweet)
    pe_features = list(map(lambda x: data[x], features))
    res = clf.predict([pe_features])[0]
        ########################################
     #print('The file %s is %s' % (os.path.basename(sys.argv[1]),['malicious', 'legitimate'][res]))
    return render_template('result.html', prediction = ['malicious', 'legitimate'][res])
```

Figure 4 Code snip of Checker.py

```
(base) F:\STUDY\SEM 6\CyberSec\FileChecker\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 windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 839-459-369
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET / HTTP/1.1←[0m" 200 -
 * Detected change in 'C:\\ProgramData\\Anaconda3\\Lib\\encodings\\__pycache__\\unicode_escape.cpython-38.pyc', reloading
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/font-awesome.min.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/owl.theme.default.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/magnific-popup.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/owl.carousel.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/styles.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/bootstrap.min.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/css/style.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/js/main.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:30] "←[37mGET /static/js/index.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/js/jquery.magnific-popup.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/js/bootstrap.min.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/js/jquery.min.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/js//owl.carousel.min.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[33mGET /static/img/team27.png HTTP/1.1←[0m" 404 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/team24.png HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/team26.png HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/team23.png HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/team25.png HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/team22.png HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/blog2.jpg HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[37mGET /static/img/background1.jpg HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:23:31] "←[33mGET /static/img/aa.jpg HTTP/1.1←[0m" 404 -
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 839-459-369
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
TestDummy.exe
127.0.0.1 - - [06/Jun/2021 23:24:10] "←[37mPOST /uploader HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:24:10] "←[37mGET /static/css/bootstrap.min.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:24:10] "←[37mGET /static/js/scripts.js HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:24:11] "←[37mGET /static/img/tick.jpg HTTP/1.1←[0m" 200 -
127.0.0.1 - - [06/Jun/2021 23:24:11] "←[33mGET /static/img/aa.jpg HTTP/1.1←[0m" 404 -
```

Figure 5 Implementation of checker.py

14. RESULT AND DISCUSSION

The below table is the shows the Features which affects the machine learning model the most along with their values. It is obtained from learning.py.

Feature Name	Value
Dll Characteristics	0.180327
Machine	0.107003
Characteristics	0.104303
Section Max Entropy	0.053374
Resource Max Entropy	0.036760
Subsystem	0.059317
Version Information Size	0.062060
Resource Min Entropy	0.036848
Image Base	0.051660
Major Subsystem Version	0.043654
Size Of Optional Header	0.047946
Major Operating System Version	0.022898

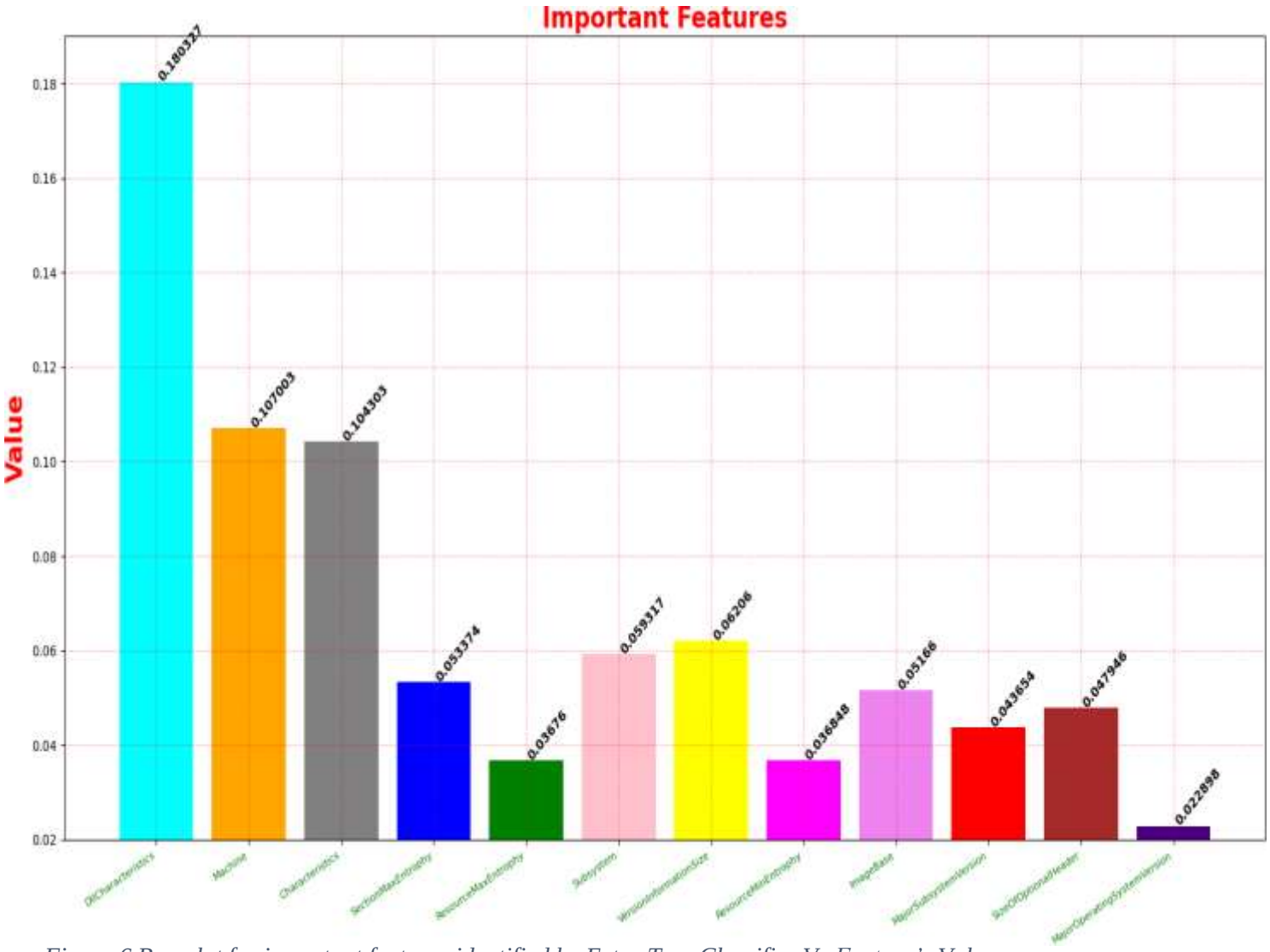


Figure 6 Bar-plot for important features identified by Extra Tree Classifier Vs Feature's Value

The below table is the shows the accuracy of each machine learning model used in the project. It is obtained from learning.py.

Algorithm	Accuracy
Decision Tree	99.040203
Random Forest	99.344440
Gradient Boosting	98.743209
ADA Boosting	98.576603
GNB	70.043463

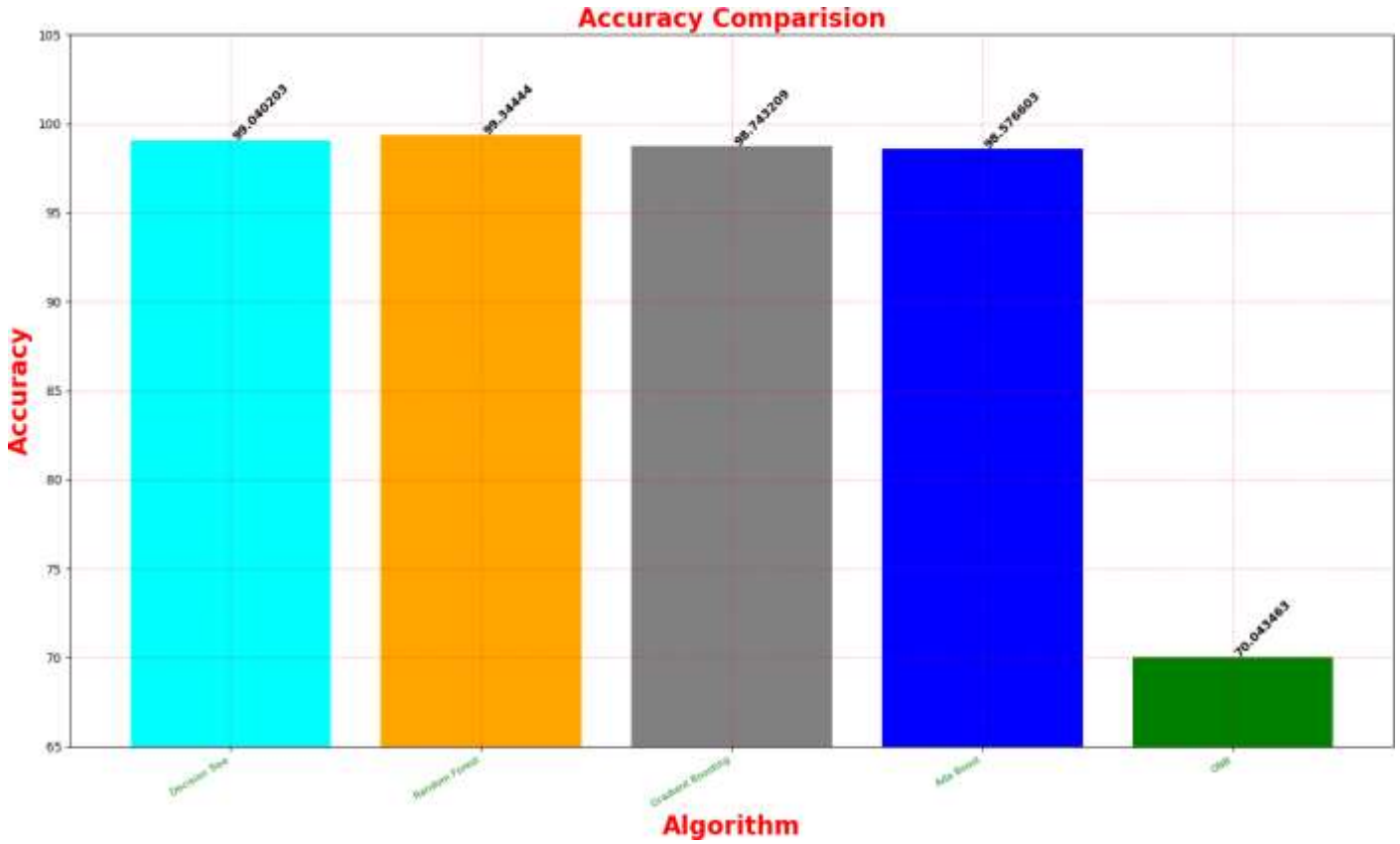


Figure 7 Bar-plot for Different Machine learning Algorithm used Vs Accuracy

As it can be concluded from the above table that the accuracy of Random Forest is highest, therefore Random Forest is used to test the new PE file to classify them into malicious or legitimate file.

Deploying the Flask app using checker.py script.

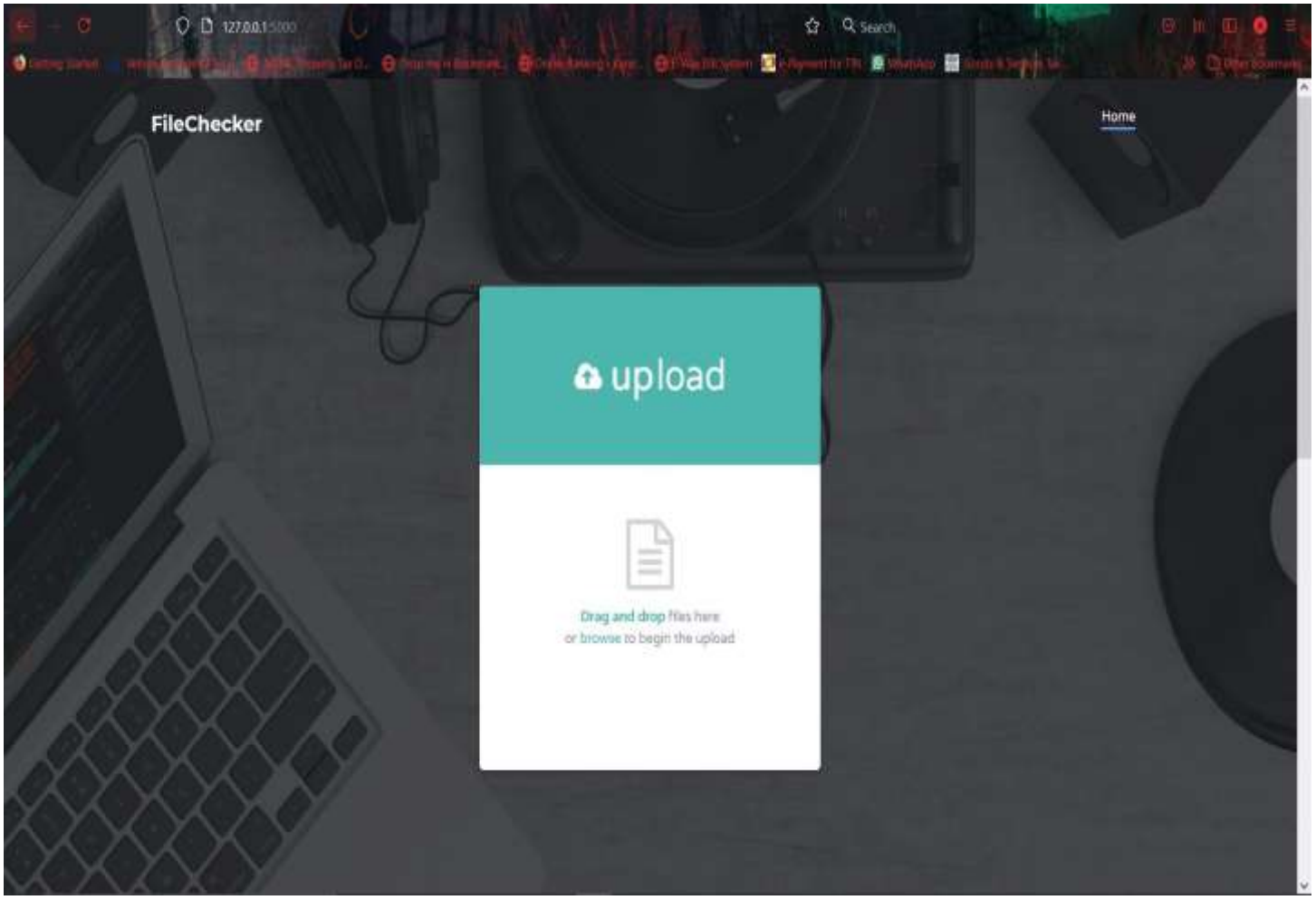


Figure 8 This is the home page (index.html) where the user is prompted to Upload the PE file

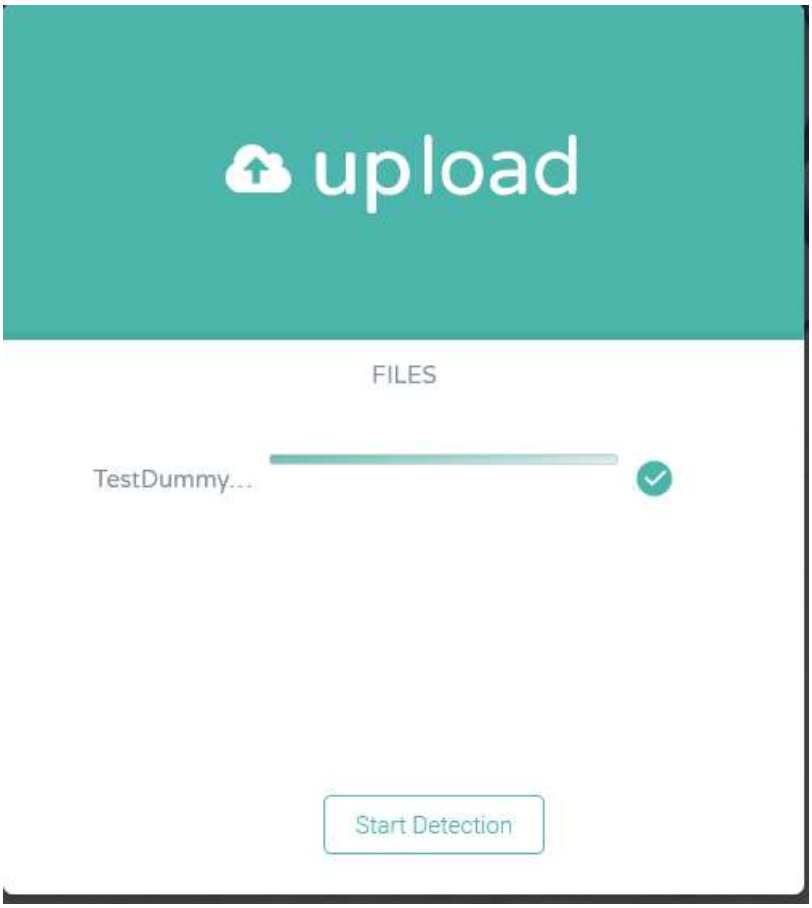


Figure 9 The user can select the PE file by drag or drop or can browse the local machine. After the file is uploaded the user is prompted to click on Start Detection button.

Figure 10 After testing the file with our model, the user will be prompted to result page(result.html).

Figure 11 If the file is Malicious user will get this output.

```
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET / HTTP/1.1←[0m" 200 -
 * Detected change in 'C:\\ProgramData\\Anaconda3\\Lib\\encodings\\__pycache__\\unicode_escape.cpython-38.pyc', reloading
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/css/bootstrap.min.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/css/owl.theme.default.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/css/magnific-popup.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/css/font-awesome.min.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/css/style.css HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[33mGET /static/img/team27.png HTTP/1.1←[0m" 404 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[37mGET /static/img/blog2.jpg HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:43:19] "←[33mGET /static/img/team27.png HTTP/1.1←[0m" 404 -
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 839-459-369
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
TestDummy2.exe
127.0.0.1 - - [07/Jun/2021 02:43:28] "←[37mPOST /uploader HTTP/1.1←[0m" 200 -
TestDummy.exe
127.0.0.1 - - [07/Jun/2021 02:43:37] "←[37mPOST /uploader HTTP/1.1←[0m" 200 -
basic-miktex-21.2-x64.exe
127.0.0.1 - - [07/Jun/2021 02:44:28] "←[37mPOST /uploader HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:44:28] "←[37mGET /static/img/cross.jpg HTTP/1.1←[0m" 200 -
127.0.0.1 - - [07/Jun/2021 02:44:28] "←[33mGET /static/img/aa.jpg HTTP/1.1←[0m" 404 -
```

Figure 12 The backend program in which the test file are sent to the build model to get classified.

15. CONCLUSION

The aim of the project is to present the machine learning approach to malware problem has been full filled. The machine learning algorithms applied were Decision Tree, Random Tree, Naïve Bayes, Gradient Boosting and ADA Boosting. After application it was observed that the Random Forest is the best algorithm for our under taking with an accuracy of 99.344440. This project can reach the application level with the help of library called pickle, to save what the algorithm has learned.

16. FUTURE WORK

- The model can be more accurate and time efficient by adding more data.
- To host the model on web for real time analysis of exe files on the cloud.
- Increasing the scope of model by including not just static but analysis of Dynamic Malware too.
- Applying different algorithms to improve the performance.

17. 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. 36 | P a g e

[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.