A Project Report
on
Malware analysis using Machine Learning

ABSTRACT

This paper aims to demonstrate the functionality and accuracy of five different machine learning algorithms for detecting whether an executable is infected or clean. Overview of malware—a type of software or code that infiltrates computer systems to steal information or cause damage.. Following this introduction, we will explore the evolution of malware over time and examine various protection techniques.

we introduce the field of machine learning and discuss its benefits, particularly in tackling malware detection. We have highlighted the significance of machine learning in addressing malware threats, along with an overview of the algorithms used in this study and their advantages. Machine learning plays a crucial role in the field, powering antivirus and anti-malware software while also being used by malicious programs, such as polymorphic malware. This type of malware employs machine learning algorithms to encrypt itself uniquely each time it infects a new host, making it increasingly challenging to detect.

This report focuses on understanding Portable Executable (PE) files and their structure using the `pefile` module. This report also offers a detailed discussion on the Flask framework and its functions. The design of templates and static files, including HTML, CSS, Bootstrap, and JavaScript has been used.

Table Of Content

1: INTRODUCTION

1.1 Malware Definition:

"Malware" is an abbreviation for "malicious software", it is used as a single term to refer to Viruses, Trojans, Worms, etc. These programs have a variety of features, such as stealing, encrypting or deleting sensitive data, modifying or hijacking basic computer functions, and monitoring computer activity. show user permission.

1.2History :

The first versions of Malware were primitive, they infested various machines through floppy disks. With the evolution of Networking and the maturation of the Internet, malware authors have adapted their malicious codes to take full advantage of this new communication environment. Below is a brief overview of the evolution of malware over time.

1.2.1 The years 1971-1999

- 1971-Creeper: An experiment designed to test how a program can move between computers.
- 1974-Wabbit: A program that multiplies itself at an accelerated pace, until the speed of the system slows down, the performance is measured, the system is reduced and eventually collapses.

1982-Elk Cloner: Written by a 15-year-old child, Elk Cloner is one of the first viruses, and widespread, to multiply itself and display a short "poem" to the infected person : “It will get on all your disks; It will infiltrate your chips; Yes, it's a Cloner! ”

- 1986-Brain Boot Sector Virus: Considered the first virus to infect MS-DOS computers.
- 1986 — PC-Write Trojan: Malware disguised as one of the oldest Trojans as a popular program called "PC-Writer." Once on a system, it deletes all files of a user.
- 1988 — Morris Worm: Infected a substantial percentage of computers connected to ARPANET, the predecessor of the Internet, which brought the network to its knees in 24 hours. this Worm marked a new beginning for malicious software.
- 1991 — Michelangelo Virus: The virus was designed to erase information from hard drives on March 6, the birthday of the famous Renaissance artist.
- 1999 - Melissa Virus: used Outlook addresses from infected machines and was sent to 50 people at once.

1.2.2 The years 2000-2010

- 2000 – ILOVEYOU Worm: the worm infected about 50 million computers. The damage caused major corporations and government agencies, including portions of the Pentagon and the British Parliament, to shut down their e-mail servers. Worms have spread globally and cost more than $ 5.5 billion in damage.
- 2003 – SQL Slammer Worm: One of the fastest spreading worms of all time, SQL Slammer infected nearly 75,000 computers in ten minutes. The worm has had a

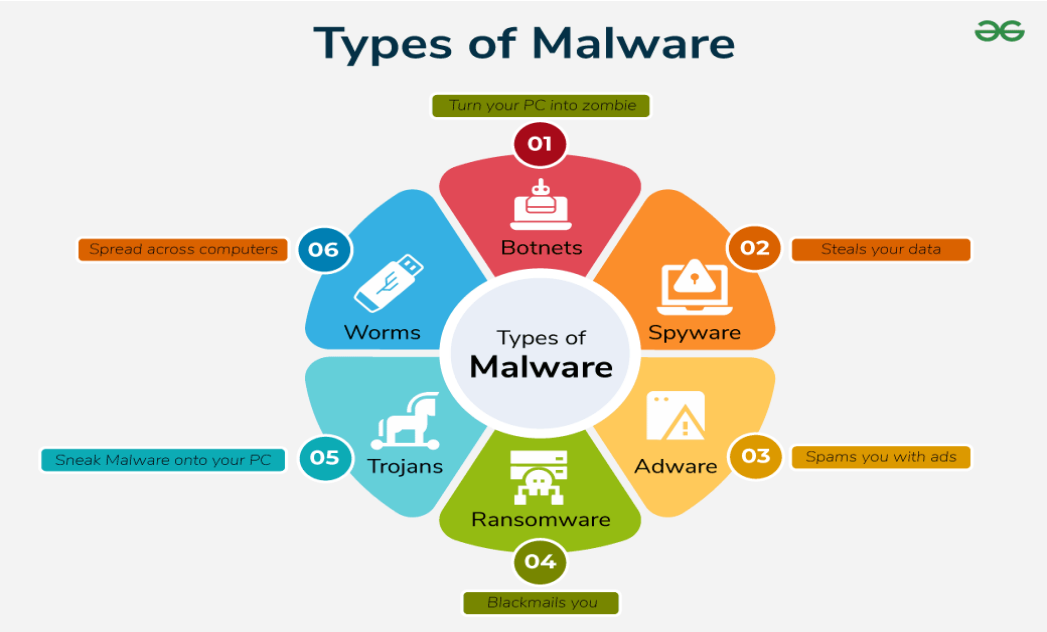
major effect worldwide, slowing down worldwide Internet traffic by denial of service.

- 2004 – Cabir Virus: Although this virus has caused some damage, it is noteworthy because it is widely recognized as the first cell phone virus.
- 2005 – Koobface Virus: One of the first cases of malware to infect PCs and then spread to social networking sites. If rearranged, the letters in "Koobface" are old, and you get "Facebook". The virus has also attacked other social networks such as MySpace and Twitter.
- 2008 – Conficker Worm: A combination of the words “configure” and “ficker”, this sophisticated worm has caused some of the 11 worst damage observed since Slammer appeared in 2003.

1.2.3 2010- present

- 2010 – Stuxnet Worm: Shortly after its release, security analysts openly speculated that the malware was designed to explicitly attack Iran's nuclear program and include the ability to affects hardware and software. The incredibly sophisticated worm is considered to be the work of a whole team of developers, making it one of the most intensive malware resources created to date.
- 2011 — Zeus Trojan: Often detected for the first time in 2007, the author of the Trojan Zeus released the code to the public in 2011, giving a new life to malware. Sometimes called the Zbot, this Trojan has become one of the most successful pieces of botnet software in the world, impacting millions of machines.
- 2013 – Cryptolocker: had a significant impact globally and helped fuel the ransomware era. • 2014 – Backoff: Malware designed to compromise Point-of-Sale (POS) systems to steal credit card data.
- 2016 – Cerberus: One of the most prolific crypto-malware threats. At one point, Microsoft found more company PCs infected with Cerberus than any other family of ransomware.
- 2017 – WannaCry Ransomware: Exploiting a vulnerability first discovered by the National Security Agent, WannaCry Ransomware has knelt down a number. systems in Russia, China, the United Kingdom and the United States, blocking access to data and demanding redemption or loss of everything. The virus has affected at least 150 countries, including hospitals, banks, telecommunications companies, warehouses and many other industries.

1.3 Types of Malware:



Computer virus
It is generally a program that is installed outside the user's will and can cause damage to both the operating system and the hardware (physical) elements of a computer.
Effects generated by the virus:

- destruction of files.
- changing the file size.
- Delete all information on the disc, including formatting it.
- destruction of the file allocation table, which makes it impossible to read the information on the disk.

Types of computer worms:

- E-Mail worms
- Instant messaging worms
- Internet worms
- IRC worms

Trojan Horse:

- File-sharing files on the network Trojan horses Trojan horses are "disguised" programs that try create gaps in the operating system to allow a user to access the system.
- Trojans do not have the facility to self-multiply like computer viruses.
- Trojan horses can be divided into several categories:
- backdoors: allows the attacker to take control of the victim's computer via the Internet;
- password stealer: programs that steal passwords (read data from the keyboard and store them in files that can be read later by the attacker or can be sent directly to the e-mail account). mail of the attacker);

Ransomware:

- Ransomware threats can be used to steal authentication names, passwords, valuable personal documents, identity data and other confidential information
- can quickly terminate the activity of an anti-virus, anti-spyware or any other software blocking its processes and disabling essential services in the system. root kit 7 .
- A rootkit software is generally a program that manages through a vulnerability of the host system to get full rights on a system, a system that modifies it so that it can use its resources undetected.
- can modify the “ps” utility on a Linux system, a utility that displays active processes, to make it NOT display the rootkit process .

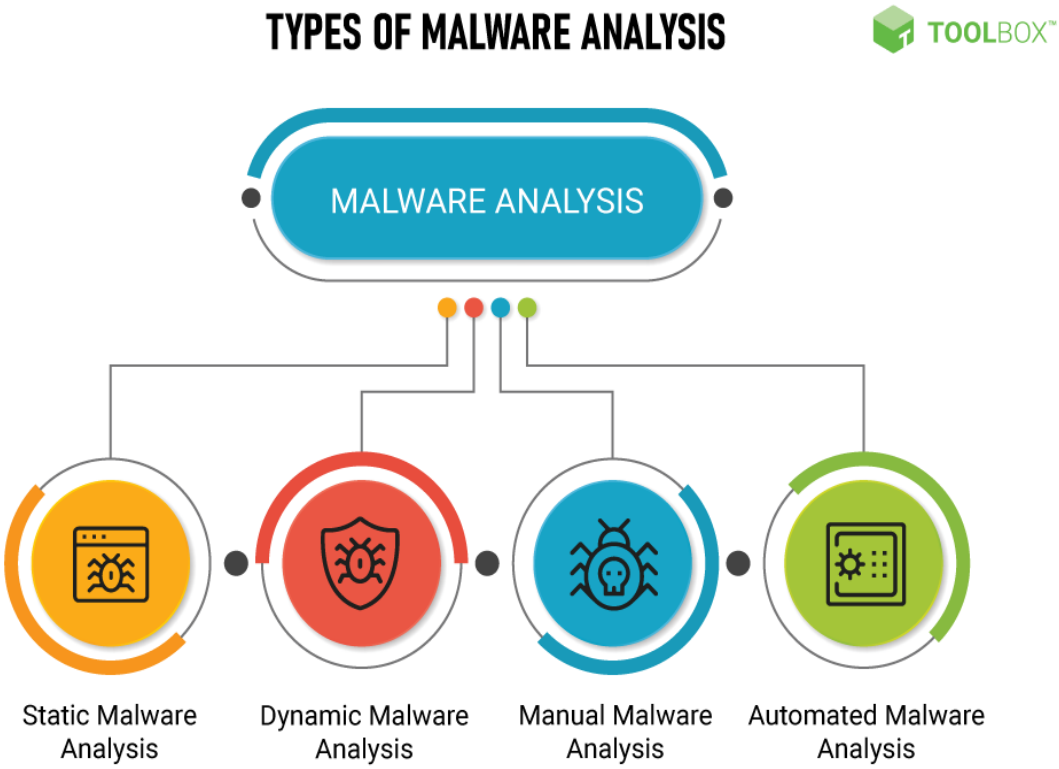
Spyware:

- Spyware is a category of cyber threats, which describes malware created to infect PC systems and then initiate illegal activities.
- In most cases, the functionality of these threats depends on the intentions of their vendors: some parts of spyware threats can be used to collect personal information (login names, passwords, and other personally identifiable data) and send them to their owners via hidden internet connections, while other spyware viruses can track their victims and collect information about their browsing habits.

Worms:

- A worm is a type of malicious software (malware) that self-replicates and spreads across networks without needing a host file or human intervention.
- Exploits vulnerabilities in operating systems or software.
- Can spread through email attachments, instant messages, or networked systems.
- Self-replicating: Copies itself without user action.
- Independent: Unlike a virus, it doesn't require attaching to a program or file.
- Can spread rapidly across networks.

1.4 Malware Analysis Techniques:



Malware analysis is necessary for the development of effective techniques for detecting infested files. This analysis is the process of observing the purpose and functionality of a malware program. There are 3 analysis techniques that have the same purpose: to explain how a malware works and what its effects are on the system, but the time and knowledge required are very different.

Static analysis

- It is also called code analysis. That is, the malware software code is observed to gain knowledge about the operation of malware functions.
- This reverse engineering technique is performed using disassembly, decompilation, debugging, and source code analysis tools. We will be following this technique as is that it is free from the overhead of execution time.

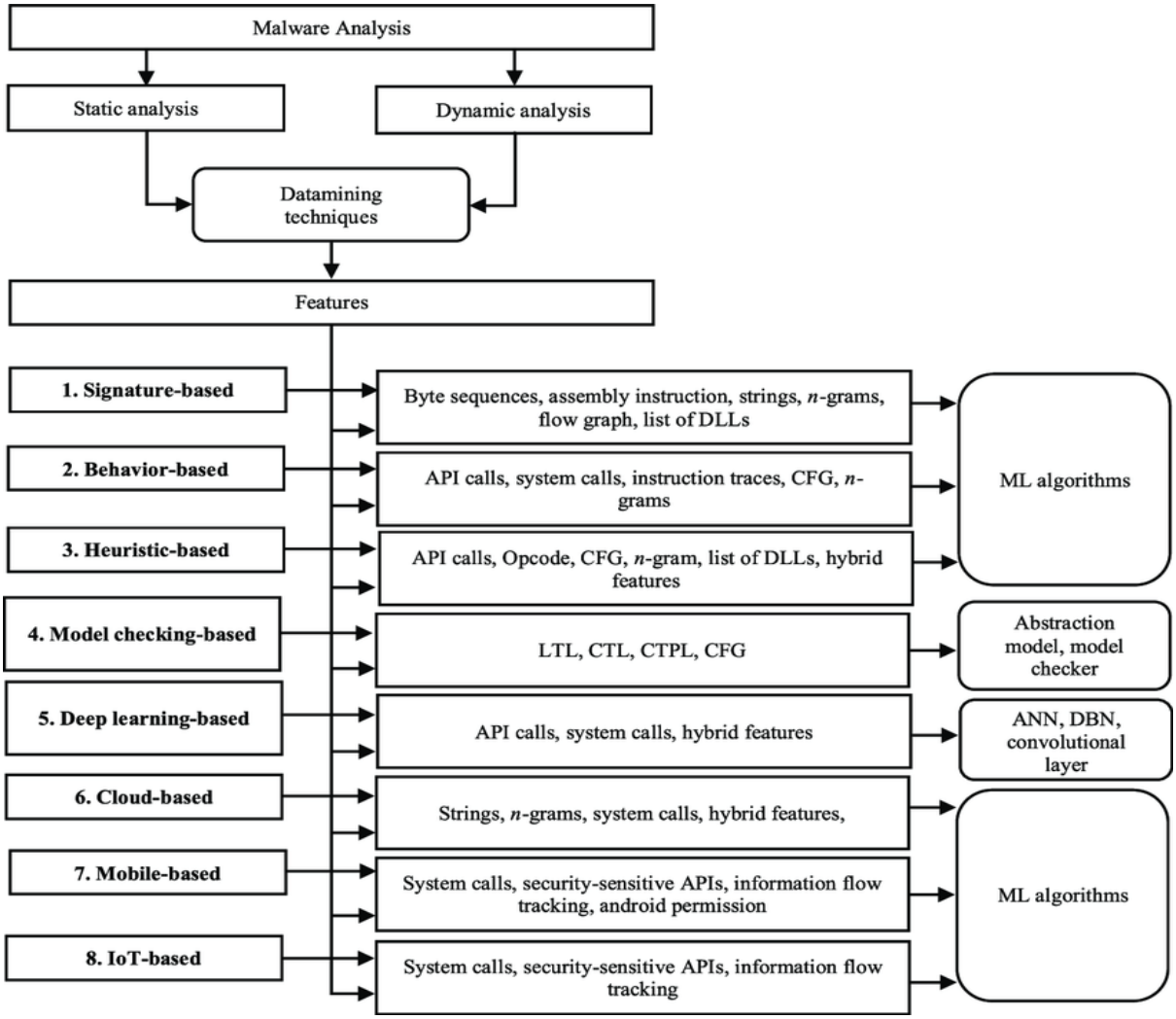
Dynamic analysis

- It is also called behavioral analysis .
- Infected files are analyzed during execution in an isolated environment such as a virtual machine, simulator, or emulator.
- After the execution of the file, the behavior and its effects on the system are monitored.

Hybrid analysis

- This technique is proposed to overcome the limitations of static and dynamic analysis.
- First, it analyses the specification of the signature for any malware code and then combines it with the other behavioural parameters to improve the complete analysis of malware.
- Due to this approach, hybrid scanning exceeds the limits of static and dynamic scans

1.5 Malware detection techniques:



Malware detection techniques are used to detect malware and prevent it. infestation of the system, protecting it from potential information loss and compromising the system. They are categorized into: signature detection, behaviour detection and feature detection.

Signature detection

- Signature-based detection is a process in which a unique identifier about a threat is established, it is known, so the threat can be detected. identified in the future.

- In the case of a virus scan, this can be a unique code template that attaches to a file, or it can be as simple as the hash of a bad file. known.

- If that specific pattern or signature is rediscovered, the file may be reported as infected. As malware has become more sophisticated, malware authors have begun to use new techniques, such as polymorphism, to change the pattern each time the object has spread from one system to another.

Behaviour Detection

- Unlike signature-based scanning, which shows If the signatures found in the file match that of a known malware database, the heuristic scan [5] uses rules and / or algorithms to search for commands that may indicate intent, evil.

- Using this method, some heuristic scanning methods are able to detect malware without the need for a signature.

- This is why most antivirus programs use both signature and heuristic methods in combination to capture any malware that might try to evade detection.

Feature detection

- Feature detection is a derivative of behaviour-based detection that attempts to overcome the typical rate of false alarms associated with it.
- Characteristic detection is based on program characteristics that describe the security behaviour of critical programs.

- This involves monitoring program executions and detecting deviations from the specification and its behaviour, instead of seemingly detecting specific attack patterns.

- This technique is similar to the detection of anomalies, different, being that it is based on features developed manually to capture the behaviour of the system instead of relying on machine learning techniques.

- The advantage of this technique is that it can instantly detect known and unknown malware, and the level of false positives is lower, but the level of false negatives is high and not as effective as behavioural detection. We will be using this technique.

Machine Learning

2.1 Definition:

- Machine Learning is a category of algorithms that allow software applications to predict much better results without being specifically programmed.
- The basic premise of machine learning is to build algorithms that receive input data and use statistical analysis to predict output data while output data is updated like many input data become valid.
- The processes involved in machine learning are similar to the processes of data mining and predictive modelling.
- Both require searching for certain patterns by date, and adjusting program actions accordingly.

2.2 How machine learning works:

Machine learning algorithms are categorized as both supervised and unsupervised.

Supervised algorithms

- They require a data researcher, or data analyst, who has the knowledge of machine learning to supply the desired input and output data, in addition to delivering feedback on the accuracy of the predictions; acute during algorithm training.
- Data researchers determine which variables, or characteristics, should be analysed by the model and used to develop predictions. Once the training is complete, the algorithm will apply what it has learned to new data.
- Supervised learning problems can be further grouped into regression and classification problems.
- Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.

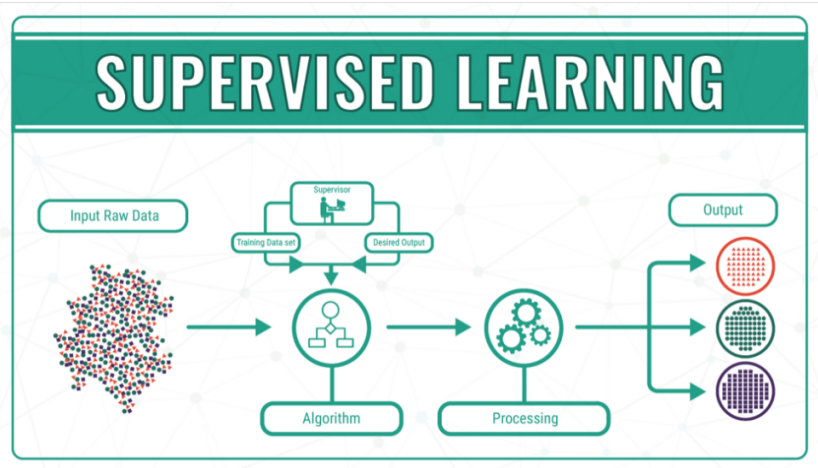


Figure 1 Supervised learning

Unsupervised algorithms

- Unsupervised and learned algorithms, also known as neural networks, are used for more complex processes than supervised algorithms, which include image recognition, speech-to-text, and natural language generation.
- These neural networks work by first combining millions of training examples with data and automatically identifying subtle correlations between multiple variables. Once trained, the algorithm can be used by associates to interpret new data.
- These algorithms become feasible only in the information age, because they require massive amounts of data to train.These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher.

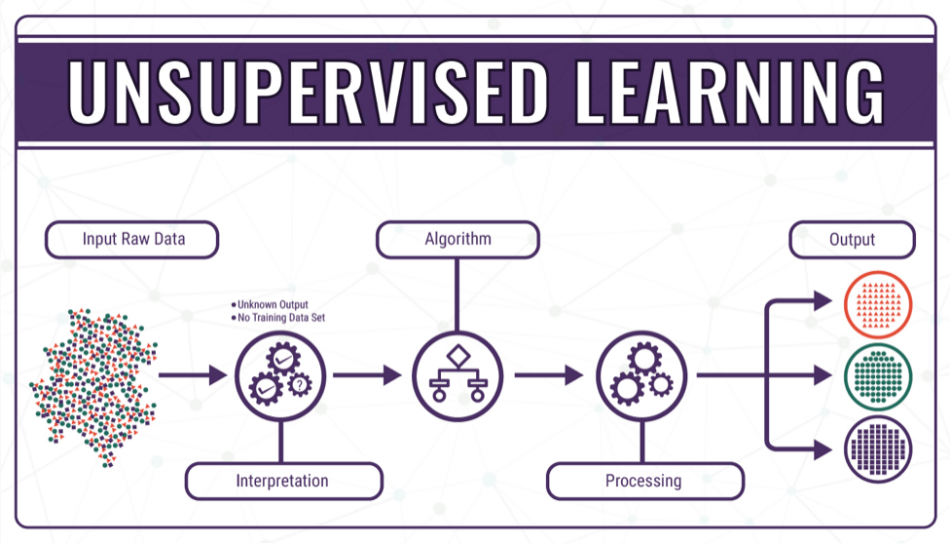


Figure 2 Unsupervised Learning

3. Algorithms used in this paper:

Random Forest

- An effective alternative is to use trees with fixed structures and random features . Tree collections are called forests, and classifiers built in so-called random forests.
- The random water formation algorithm requires three arguments: the data, a desired depth of the decision trees, and a number K of the total decision trees to be built, i. The algorithm generates each of the K trees. independent, which makes it very easy to parallelize.

- For each tree, build a complete binary tree. The characteristics used for the branches of this tree are selected randomly, usually with replacement, which means that the same characteristic can occur more than 20 times, even in a single branch.
- The leaves of this tree, where predictions are made, are completed based on training data. The last step is the only point at which the training data is used.
- The resulting classifier is just a K-lot vote, and random trees. The most amazing thing about this approach is that it actually works remarkably well. They tend to work best when all the features are at least, well, relevant, because the number of features selected for a particular tree is small. One intuitive reason that it works well is the following. Some trees will query unnecessary features. These trees will essentially make random predictions. But some of the trees will happen to question good characteristics and make good predictions (because the leaves are estimated based on training data).

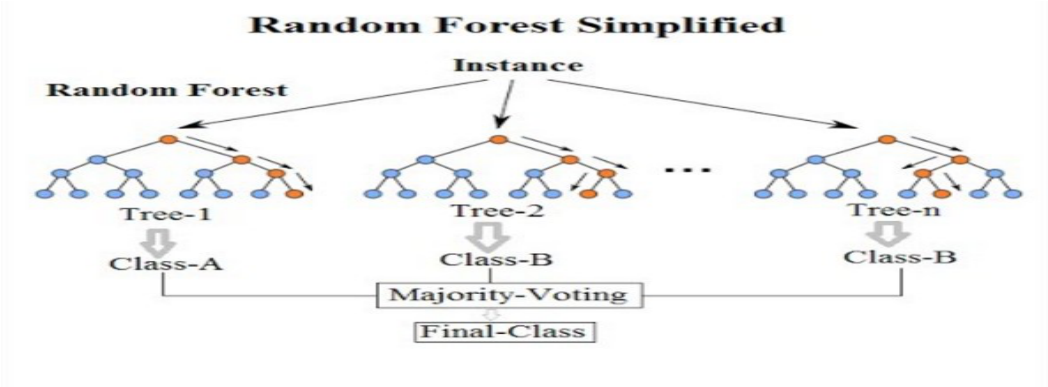


Figure 3 Random forest view

ADA Boost

- Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method.
- AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier.
- The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Any machine learning algorithm can be used as base classifier if it accepts weights on the training set.
- The classifier should be trained interactively on various weighed training examples.In each iteration, it tries to provide an excellent fit for these examples by minimizing training error.Initially, Adaboost selects a training subset randomly.It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
- It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.Also,
- It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.This

process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators.To classify, perform a "vote" across all of the learning algorithms you built.

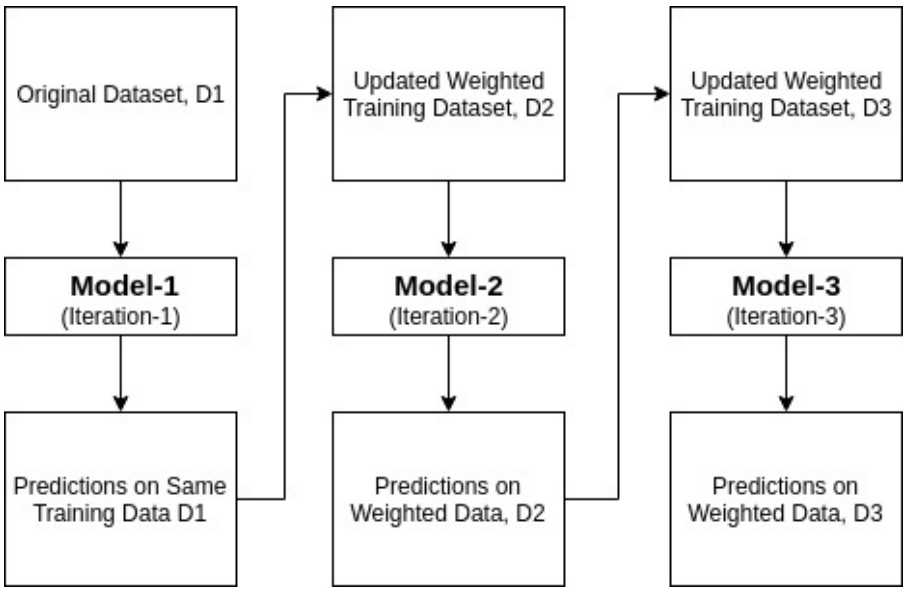


Figure 4 Ada Boost

Gradient boosting

- The idea behind "gradient boosting" is to take a weak hypothesis or weak learning algorithm and make a series of tweaks to it that will improve the strength of the hypothesis/learner.
- This type of Hypothesis Boosting is based on the idea of Probability Approximately Correct Learning (PAC).This PAC learning method investigates machine learning problems to interpret how complex they are, and a similar method is applied to Hypothesis Boosting.
- In hypothesis boosting, you look at all the observations that the machine learning algorithm is trained on, and you leave only the observations that the machine learning method successfully classified behind, stripping out the other observations.
- A new weak learner is created and tested on the set of data that was poorly classified, and then just the examples that were successfully classified are kept.
- The Gradient Boosting Classifier depends on a loss function. A custom loss function can be used, and many standardized loss functions are supported by gradient boosting classifiers, but the loss function has to be differentiable .
- Classification algorithms frequently use logarithmic loss, while regression algorithms can use squared errors. Gradient boosting systems don't have to derive a new loss function every time the boosting algorithm is added, rather any differentiable loss function can be applied to the system.

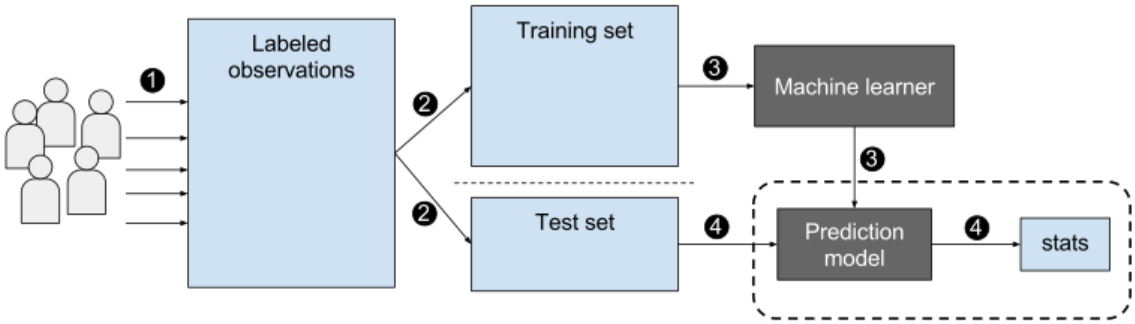


Figure 5 Gradient boost

Decision Tree

- A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.
- The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning.
- This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.
- Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm.
- The time complexity of decision trees is a function of the number of records and number of attributes in the given data.
- The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

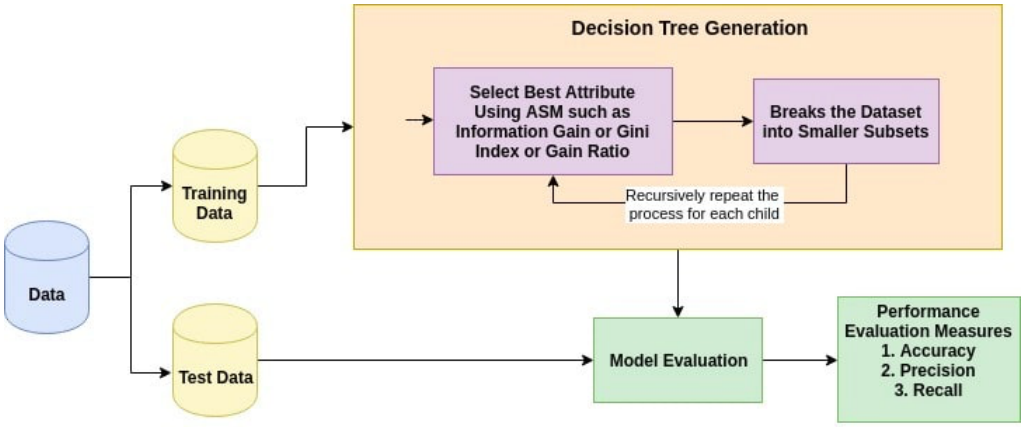


Figure 6 Decision Tree

Naïve Bayes

- Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms.
- Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.
- Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently.
- This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence.

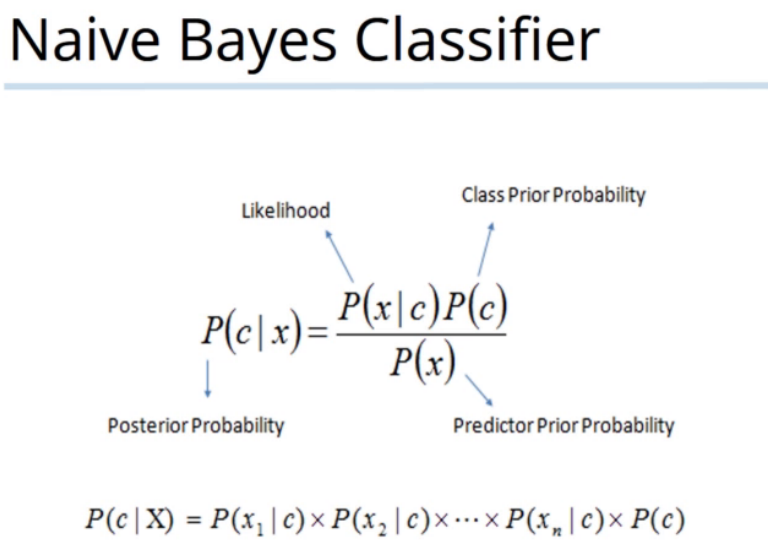


Figure 7 Naive Bayes

4 Materials Required:

Computer:

- A base machine with windows 8, 8 GB RAM, 250 GB of hard drive with core i5, installed

Operating System:

- The main OS to be used is Windows 10

Reporting tools:

- VScode as IDE
- Github for version control and remote work
- PeStudio is a free tool performing the static investigation of any Windows executable binary. A file being analyzed without PeStudio is never launched.

Flask framework:

- Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

Functional Partitioning of Project:

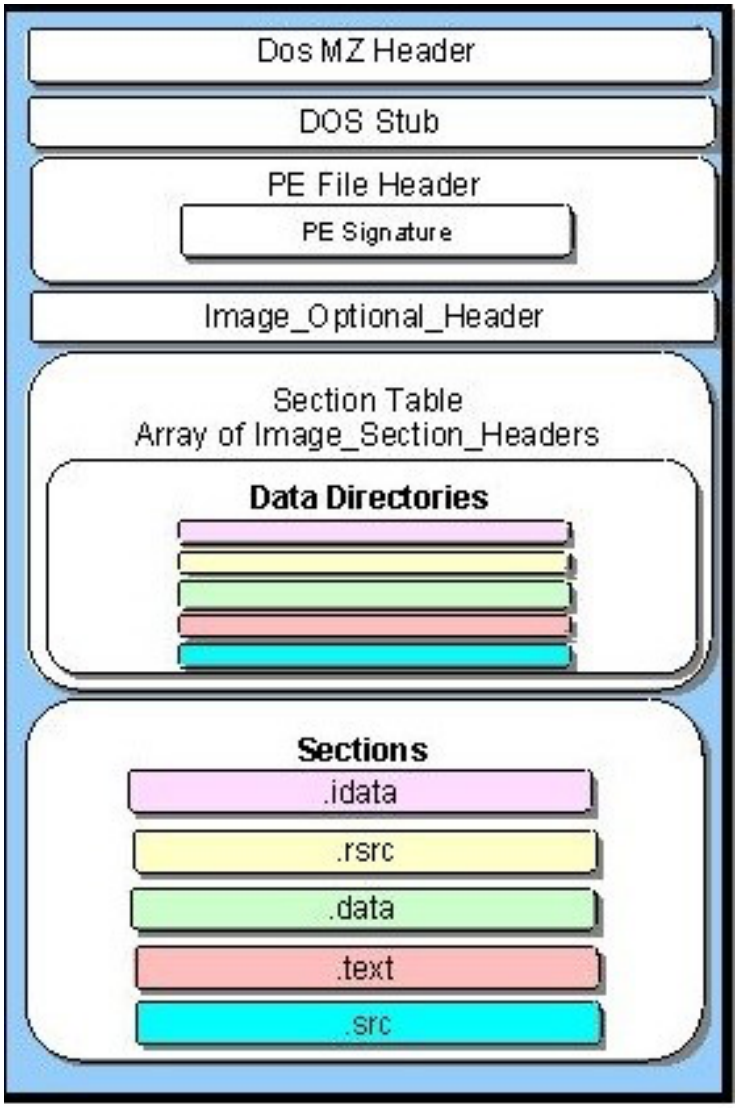
Overall Project is partitioned into various phases.

a) Phase 1: Downloading 1,38,000+ dataset of malicious and legitimate PE files

b) Phase 2: Performing Malware analysis and classification, choosing best out of 5 ML algos and feature classification

c) Phase 3: Deploying Full stack Flask framework using HTML,Bootstrap,CSS,Javascript for static analysis

5: Understanding PE files



Overview of PE files:

- PE stands for portable executable. It is the native executable format of Win32. Its specification is derived somewhat from the Unix COFF (common object file format).
- The format information of PE file is illustrated in Figure 8.
- It is basically a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code.
- A PE file consists of a PE file header and a section table (section headers) followed by the sections' data.
- The PE file header consists of a MS DOS header, the PE signature, the image file header, and an optional header.
- The file headers are followed immediately by section headers.
- Section header provides information about its associated section, including location, length, and characteristics. Section is the basic unit of code or data within a PE or COFF file.

Dataset

- Dataset is divided into malware and benign software.
- We should obtain enough representative training dataset in actual antivirus software. Because we just verify the feasibility of our method, we collected 41324 types of benign software and 96724 types of malware.
- All are in the Windows PE format.
- We obtained benign software from Windows folder and Program Files folder and used commercial software to verify that each executable was indeed benign.
- We obtained malware from the website VirusShare.

Feature Extraction

- There are many format features in PE files, but most of those features are not helpful in distinguishing malware and benign software.
- Based on our empirical studies and in-depth analysis of the format features of the PE files, we extracted 54 features that have the potential to distinguish between benign software and malware, from given PE files.
- These features are summarized in. In the below discussion, we gave a brief description of the extracted features in our study.9 important features are also classified.

DOS Header

- This field is used to identify an MS-DOS-compatible file type. All MS-DOS-compatible executable files set this value to 0x54AD,
- which represents the ASCII characters MZ. MS-DOS headers are sometimes referred to as MZ headers for this reason. It starts at offset 0 (this can be view with a hex editor).

DOS Stub

- The DOS stub usually just prints a string, something like the message, “This program cannot be run in DOS mode.” It can be a full-blown DOS program.
- When building applications on Windows, the linker sends instruction to a binary called winstub.exe to the executable file. This file is kept in the address 0x3c, which is offset to the next PE header section.

PE File Header

- Like other executable files, a PE file has a collection of fields that defines what the rest of file looks like.
- The header contains info such as the location and size of code, as we discussed earlier.
- The first few hundred bytes of the typical PE file are taken up by the MS-DOS stub.

- The PE file is located by indexing the e_ifanew of the MS DOS header. If a new simply gives the offset to the file, so add the file's memory-mapped address to determine the actual memory-mapped address.
- There are some basic sub-sections defined in the header section itself; they are listed below:
  Signature
  Machines
  NumberOfSections
  SizeOfOptionalHeader

We can see there are lots of headers and it is not possible to cover each and everything in detail due to space limitations, so we will discuss some of the important things that are necessary.

Characteristics

- **Signature**: It only contains the signature so that it can be easily understandable by windows loader. The letters P.E. followed by two 0's tells everything.

- **Machines**: This is a number that identifies the type of machine on the target system, such as Intel, AMD, etc. We will target a basic structure like Intel, as shown below:
- 0x14d
- Intel i860

We will see above characteristics in the tool later.

- **NumberOfSections:** This defines the size of the section table, which immediately follow the header.

- **SizeOfOptionalHeader:** This lies between top of the optional header and the start of the section table. This is the size of the optional header that is required for an executable file. This value should be zero for an object file.

- **Characteristics:** This is the characteristic flags that indicate an attribute of the object or image file. It has a flag called Image_File_dll, which has the value 0x2000, indicating that the image is a DLL. It has also different flags that are not required for us at this time

- **Image_Optional_Header:** This optional header contains most of the meaningful information about the image, such as initial stack size, program entry point location, preferred base address, operating system version, section alignment information, and so forth. We can see the information in the snapshot below.

- **ImageBase:** the preferred address of the image when loaded into memory. The default address is 0x00400000. An attacker can change this address depending on his requirement with an option like "-BASE:linker."

- **SectionAllignment:** The alignment of the section when loaded into memory. Section alignment can be no less than page size (currently 4096 bytes on the windows x86).

- **FileAlignment**: The granularity of the alignment of the sections in the file. For example, if the value in this field is 512 (200h), each section must start at multiples of 512 bytes. If the first section is at file offset 200h and the size is 10 bytes, the next section must be located at file offset 400h: the space between file offsets 522 and 1024 is unused/undefined.

- **MajorSubSystemVersion:** Indicates the Windows NT Win32 subsystem major version number, currently set to 3 for Windows NT version 3.10.

- **SizeOfImage:** The size of the memory, including all of the headers. As the image is loaded into memory, it must be a multiple of SectionAllignment.

The Section table

- This table immediately follows the optional header.
- The location of this section of the section table is determined by calculating the location of the first bytes after header. For that, we have to use the size of the optional header.
- The number of the array members is determined by NumberOfSections field in the file header (IMAGE_FILE_HEADER) structure. The structure is called IMAGE_SECTION_HEADER.

The PE File Section

- This section contains the main content of the file, including code, data, resources and other executable files. Each section has a header and body.
- An application in Windows NT typically has nine different predefined sections, such as .text, .bss, .rdata, .data, .rsrc, .edata, .idata, .pdata, and .debug. Depending on the application, some of these sections are used, but not all are used.

.

6: Understanding Flask

6.1 Flask:

- Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
- However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.
- Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit.

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def hello():
    return 'Hello, World!'
```

6.2 Jinja:

Jinja2 is a full-featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

6.3 Werkzeug:

werkzeug German noun: "tool". Etymology: *werk* ("work"), *zeug* ("stuff")
Werkzeug is a comprehensive WSGI web application library. It began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries.
Flask wraps Werkzeug, using it to handle the details of WSGI while providing more structure and patterns for defining powerful applications.
erkzeug includes:

- An interactive debugger that allows inspecting stack traces and source code in the browser with an interactive interpreter for any frame in the stack.
- A full-featured request object with objects to interact with headers, query args, form data, files, and cookies.
- A response object that can wrap other WSGI applications and handle streaming data.
- A routing system for matching URLs to endpoints and generating URLs for endpoints, with an extensible system for capturing variables from URLs.

- HTTP utilities to handle entity tags, cache control, dates, user agents, cookies, files, and more.
- A threaded WSGI server for use while developing applications locally.
- A test client for simulating HTTP requests during testing without requiring running a server.

Werkzeug is Unicode aware and doesn't enforce any dependencies. It is up to the developer to choose a template engine, database adapter, and even how to handle requests. It can be used to build all sorts of end user applications such as blogs, wikis, or bulletin boards.

6.4 Templates:

- The term **'web templating system'** refers to designing an HTML script in which the variable data can be inserted dynamically.
- A web template system comprises of a template engine, some kind of data source and a template processor.
- Flask uses **jinja2** template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

6.5 Static Files:

- A web application often requires a static file such as a **javascript** file or a **CSS** file supporting the display of a web page.
- Usually, the web server is configured to serve them for you, but during the development, these files are served from *static* folder in your package or next to your module and it will be available at ***/static*** on the application.
- A special endpoint 'static' is used to generate URL for static files.

Javascript

- JavaScript often abbreviated as JS, is a programming language that conforms to the ECMAScript specification.
- JavaScript is high-level, often just-in-time compiled, and multi-paradigm.
- It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

- As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles.
- It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

7: Implementation

Python Script:

There Will be two python scripts learning.py and checker.py . First one is for training and creating machine learning model and second one is for implementing the model on real time files

Importing Modules

Using Pip to install all the packages required for the project.

```
pip install [options] <requirement specifier> [package-index-options]
```

Then importing modules into files
For learning.py -

```
import pandas as pd
import numpy as np
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
from sklearn import tree
```

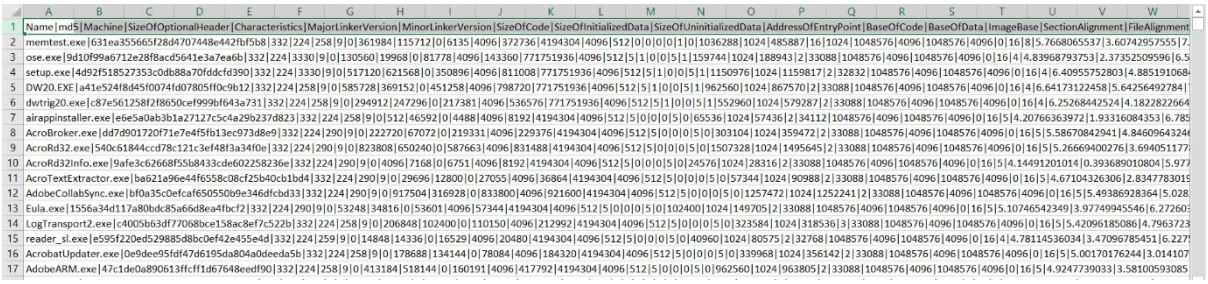
For checker .py –

```
import pefile
# pefile is a Python module to read and work with PE (Portable Executable) fil
es.
import os
import array
import math
import pickle
import joblib
import sys
import argparse
import os
import sys
import shutil
import time
import re
```

```
import pandas as pd
from flask import Flask, request, jsonify, render_template, abort, redirect, u
rl_for
from werkzeug import secure_filename
import joblib
from sklearn.ensemble import RandomForestClassifier
```

Handling Data:

Our data includes file name and the rest is PE file headers and sections separated by '|' respectively. Refer to Figure 9, Figure 10



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Name|md5|Machine|SizeOfOptionalHeader|Characteristics|MajorLinkerVersion|MinorLinkerVersion|SizeOfCode|SizeOfInitializedData|SizeOfUninitializedData|AddressOfEntryPoint|BaseOfCode|BaseOfData|ImageBase|SectionAlignment|FileAlignment																						
2	memtest.exe|631ea355665f28d4707448e442fbf5b8|332|224|258|9|0|361984|115712|0|6135|4096|372736|4194304|4096|512|0|0|0|1|0|1036288|1024|485887|16|1024|1048576|4096|1048576|4096|0|16|8|5.7668065537|3.60742957555|7.																						
3	ose.exe|9d10f99a6712e28f8acd5641e3a7ea6b|332|224|3330|9|0|130560|19968|0|81778|4096|143360|771751936|4096|512|5|1|0|0|5|1|159744|1024|188943|2|33088|1048576|4096|1048576|4096|0|16|4|4.83968793753|2.37352509596|6.5																						
4	setup.exe|4d92f518527353c0db88a70fddcfd390|332|224|3330|9|0|517120|621568|0|350896|4096|811008|771751936|4096|512|5|1|0|0|5|1|1150976|1024|1159817|2|32832|1048576|4096|1048576|4096|0|16|4|6.40955752803|4.885191068																						
5	DW20.EXE|a41e524f8d45f0074fd07805ff0c9b12|332|224|258|9|0|585728|369152|0|451258|4096|798720|771751936|4096|512|5|1|0|0|5|1|962560|1024|867570|2|33088|1048576|4096|1048576|4096|0|16|4|6.64173122458|5.64256492784|																					
6	dwtrig20.exe|c87e561258f2f8650cef999bf643a731|332|224|258|9|0|294912|247296|0|217381|4096|536576|771751936|4096|512|5|1|0|0|5|1|552960|1024|579287|2|33088|1048576|4096|1048576|4096|0|16|4|6.25268442524|4.1822822664																						
7	airappinstaller.exe|e6e5a0ab3b1a27127c5c4a29b237d823|332|224|258|9|0|512|46592|0|4488|4096|8192|4194304|4096|512|5|0|0|0|5|0|65536|1024|57436|2|34112|1048576|4096|1048576|4096|0|16|5|4.20766363972|1.93316084353|6.785																						
8	AcroBroker.exe|dd7d901720f71e7e4f5fb13ec973d8e9|332|224|290|9|0|222720|67072|0|219331|4096|229376|4194304|4096|512|5|0|0|0|5|0|303104|1024|359472|2|33088|1048576|4096|1048576|4096|0|16|5|5.58670842941|4.84609643246																						
9	AcroRd32.exe|540c61844ccd78c121c3ef48f3a34f0e|332|224|290|9|0|823808|650240|0|587663|4096|831488|4194304|4096|512|5|0|0|0|5|0|1507328|1024|1495645|2|33088|1048576|4096|1048576|4096|0|16|5|5.26669400276|3.6940511773																						
10	AcroRd32Info.exe|9afe3c62668f55b8433cde602258236e|332|224|290|9|0|4096|7168|0|6751|4096|8192|4194304|4096|512|5|0|0|0|5|0|24576|1024|28316|2|33088|1048576|4096|1048576|4096|0|16|5|4.14491201014|0.393689010804|5.977																						
11	AcroTextExtractor.exe|ba621a96e44f6558c08cf25b40cb1bd4|332|224|290|9|0|29696|12800|0|27055|4096|36864|4194304|4096|512|5|0|0|0|5|0|57344|1024|90988|2|33088|1048576|4096|1048576|4096|0|16|5|4.67104326306|2.8347783015																						
12	AdobeCollabSync.exe|bf0a35c0efcaf650550b9e346dfcbd33|332|224|290|9|0|917504|316928|0|833800|4096|921600|4194304|4096|512|5|0|0|0|5|0|1257472|1024|1252241|2|33088|1048576|4096|1048576|4096|0|16|5|5.49386928364|5.028																						
13	Eula.exe|1556a34d117a80bdc85a66d8ea4fbcf2|332|224|290|9|0|53248|34816|0|53601|4096|57344|4194304|4096|512|5|0|0|0|5|0|102400|1024|149705|2|33088|1048576|4096|1048576|4096|0|16|5|5.10746542349|3.97749945546|6.27260																						
14	LogTransport2.exe|c4005b63df77068bce158ac8ef7c522b|332|224|258|9|0|206848|102400|0|110150|4096|212992|4194304|4096|512|5|0|0|0|5|0|323584|1024|318536|3|33088|1048576|4096|1048576|4096|0|16|5|5.42096185086|4.7963723																						
15	reader_sl.exe|e595f220ed529885d8bc0ef42e455e4d|332|224|259|9|0|14848|14336|0|16529|4096|20480|4194304|4096|512|5|0|0|0|5|0|40960|1024|80575|2|32768|1048576|4096|1048576|4096|0|16|4|4.78114536034|3.47096785451|6.2275																						
16	AcrobatUpdater.exe|0e9dee95fdf47d6195da804a0deeda5b|332|224|258|9|0|178688|134144|0|78084|4096|184320|4194304|4096|512|5|0|0|0|5|0|339968|1024|356142|2|33088|1048576|4096|1048576|4096|0|16|5|5.00170176244|3.014107																						
17	AdobeARM.exe|47c1de0a890613ffcff1d67648eedf90|332|224|258|9|0|413184|518144|0|160191|4096|417792|4194304|4096|512|5|0|0|0|5|0|962560|1024|963805|2|33088|1048576|4096|1048576|4096|0|16|5|4.9247739033|3.58100593085																						

Figure 8 Data Format

Name|md5|Machine|SizeOfOptionalHeader|Characteristics|MajorLinkerVersion|MinorLinkerVersion|SizeOfCode|SizeOfInitializedData|SizeOfUninitializedData|AddressOfEntryPoint|BaseOfCode|BaseOfData|Im
ageBase|SectionAlignment|FileAlignment|MajorOperatingSystemVersion|MinorOperatingSystemVersion|MajorImageVersion|MinorImageVersion|MajorSubsystemVersion|MinorSubsystemVersion|SizeOfImage|Siz
eOfHeaders|CheckSum|Subsystem|DllCharacteristics|SizeOfStackReserve|SizeOfStackCommit|SizeOfHeapReserve|SizeOfHeapCommit|LoaderFlags|NumberOfRvaAndSizes|SectionsNb|SectionsMeanEntropy|Sect
ionsMinEntropy|SectionsMaxEntropy|SectionsMeanRawsize|SectionsMinRawsize|SectionMaxRawsize|SectionsMeanVirtualsize|SectionsMinVirtualsize|SectionMaxVirtualsize|ImportsNbDLL|ImportsNb|ImportsN
bOrdinal|ExportNb|ResourcesNb|ResourcesMeanEntropy|ResourcesMinEntropy|ResourcesMaxEntropy|ResourcesMeanSize|ResourcesMinSize|ResourcesMaxSize|LoadConfigurationSize|VersionInformationSize|
legitimate

Figure 9 Data format2

To train the data in supervise learning we first use pandas module to load the .csv file into our VScode IDE.

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

This way we are now just working with 'Legitimate' files . This is the first step in creating a machine learning model.

We collected 41324 types of benign software and 96724 types of malware.

Feature Selection

Extra Trees Classifier is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a "forest" to output it's classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

```
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) :}
```

Now converting in training and testing data in 20% range ! as total x is 138047 and testing is 138047*0.2=27610

```
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)  # as mentioned above
```

Sorting Important features for more accuracy

```
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
]]))
```

Output-

Researching important feature based on 54 total features
13 features identified as important:
1. feature DllCharacteristics (0.166928)
2. feature Characteristics (0.112034)
3. feature Machine (0.088879)
4. feature VersionInformationSize (0.075772)
5. feature Subsystem (0.066985)
6. feature ImageBase (0.051829)
7. feature SizeOfOptionalHeader (0.051030)
8. feature MajorSubsystemVersion (0.049418)
9. feature SectionsMaxEntropy (0.048206)
10. feature ResourcesMaxEntropy (0.035509)

11. feature ResourcesMinEntropy (0.033891)
12. feature MajorOperatingSystemVersion (0.023647)
13. feature SectionsMinEntropy (0.020279)

Important Features-

['Machine', 'SizeOfOptionalHeader', 'Characteristics', 'ImageBase', 'MajorOperatingSystemVersion', 'MajorSubsystemVersion', 'Subsystem', 'DllCharacteristics', 'SectionsMinEntropy', 'SectionsMaxEntropy', 'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'VersionInformationSize']

Algorithm training

We create a dictionary key = Name of algorithm , Value= Algorithm classifier code and loop over it
algorithms = { }

Random Forest

```
 "RandomForest": ske.RandomForestClassifier(n_estimators=50),
```

The max_depth parameter denotes maximum depth of the tree. In case, of random forest, the se ensemble classifiers are the randomly created decision trees. Each decision tree is a single classifier and the target prediction is based on the majority voting method.

Gradient Boosting and AdaBoost

```
    "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=50),
    "AdaBoost": ske.AdaBoostClassifier(n_estimators=100),
```

Ada mean Adaptive Both are boosting algorithms which means that they convert a set of wea k learners into a single strong learner. Theyboth initialize a strong learner (usually a decision tree) and iteratively create a weak learner that is added to the strong learner. They differ on h ow they create the weak learners during the iterative process.

Bayes Theorem

```
GNB": GaussianNB()
```

Bayes theorem is based on conditional probability. The conditional probability helps us calcu lating the probability that something will happen

Decision Trees

```
"DecisionTree": tree.DecisionTreeClassifier(max_depth=10)
```

Algorithm testing

Looping over the dictionary iterable of algorithms

```
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
```

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set** X_test, y_test. Note that the word "experiment" is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally. Here is a flowchart of typical cross validation workflow in model training. The best parameters can be determined by grid search techniques.

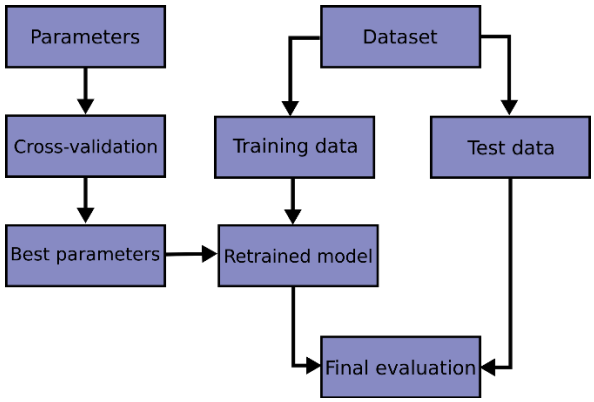


Figure 10 Cross-validation

Output-

Now testing algorithms
RandomForest : 99.406012 %
GradientBoosting : 98.804781 %
AdaBoost : 98.652662 %
GNB : 70.398406 %
DecisionTree : 99.058312 %

Winner algorithm is RandomForest with a 99.406012 % success
Saving algorithm and feature list in classifier directory...
Saved
[[19372 94]
 [53 8091]]
 False positive rate : 0.488759 %
False negative rate : 0.844243 %

Confusion Matrix

A confusion matrix, also known as an error matrix, is a specific table layout that allows visua lization of the performance of an algorithm, typically a supervised learning.

[[19372 94]
 [53 8091]]

Saving the features list

Save the algorithm and the feature list for later predictions
Persist an arbitrary Python object into one file.

```
joblib.dump(algorithms[winner], 'classifier/classifier.pkl')
open('classifier/features.pkl', 'wb').write(pickle.dumps(features))
```

joblib works especially well with NumPy arrays which are used by sklearn so depending on t he classifier type you use you might

Have performance and size benefits using joblib.Otherwise pickle does work correctly so savi ng a trained classifier and loadingit again will produce the same results no matter which of th e serialization libraries you use

Flask Scripts:

Parent Script

In order to test **Flask** installation, type the following code in the editor as **checker.py**
from flask import Flask

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

```
if __name__ == '__main__':
    app.run(debug=True)
```

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.
Flask constructor takes the name of **current module (__name__)** as argument.
The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.
app.route(rule, options)

- The **rule** parameter represents URL binding with the function.
- The **options** is a list of parameters to be forwarded to the underlying Rule object.

In the above example, **'/'** URL is bound with **various** functions. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.
Finally the **run()** method of Flask class runs the application on the local development server.
app.run(host, port, debug, options)
All parameters are optional

Sr.No.	Parameters & Description
1	**host** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to '0.0.0.0' to have server available externally
2	**port** Defaults to 5000
3	**debug** Defaults to false. If set to true, provides a debug information
4	**options** To be forwarded to underlying Werkzeug server.

Table 1

The above given **Python** script is executed from Python shell.
Python chekcer.py
A message in Python shell informs you that
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Open the above URL **(localhost:5000)** in the browser. **'index.html'** template will be displayed on it.
Debug mode
A **Flask** application is started by calling the **run()** method. However, while the application is under development, it should be restarted manually for each change in the code. To avoid this inconvenience, enable **debug support**. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application.
The **Debug** mode is enabled by setting the **debug** property of the **application** object to **True** before running or passing the debug parameter to the **run()** method.
app.debug = True
app.run()
app.run(debug = True)

Routing

Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.
The **route()** decorator in Flask is used to bind URL to a function.

```
@app.route('/')
```

Here, URL **'/'** rule is bound to the **'index.html'** template . As a result, if a user visits **http://localhost:5000/hello** URL, the output of the
***def* home():**
return render_template('index.html')
function will be rendered in the browser.

The **add_url_rule()** function of an application object is also available to bind a URL with a function as in the above example, **route()** is used.

```
@app.route('/uploader', methods=['GET', 'POST'])
```

Entropy calculation

In general words, entropy is referred as the measurement of particular data in digital values. Similar to this, the term File Entropy is the representation of data sets in specific file. That is, the phrase File Entropy is used to measure the amount of data which is present in a selected file. File Entropy is also use in the field of malware protection, in the process of malware analysis as there are all kind of security related tools that you check on the file to extract all kind of information from the file, to determine if the file is a malware or legit file, and if it is a malware this can be useful on the malware file entropy can be a useful method to quickly check if the malware file had been packed with one of the packed software it is also a good method to check if the file encrypted by one of the encryption algorithm.

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

    return entropy
```

Entropy calculation

Storing resources in dictionary res{} of out test file where we are supposed to test our machine learning model which we had developed in learning.py For this, We call 3 functions all using PE file module by Mr. Erro Cara

1.def get_resources(peFile)- to Extract resources [entropy, size]

```
resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in resource_id.directory.entries
:
                                data = pe.get_data(resource_lang.data.struct.O
ffsetToData,
                                                   resource_lang.data.struct.S
ize)
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)

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

2. def get_version_info(pe)- Returns version information of test.exe file

```
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
```

3. Get all the header files and features in a dictionary called res{}

def extract_infos(fpath):

```
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
```

Now we have extracted all the features and header files out test.exe , we will run our machine learning model to predict if test.exe is malicious or legitimate

Analysing alien file

Here, we upload the alien PE file and run malware analysis on it.
The result is displayed on ‘result.html’ template

```
@app.route('/uploader', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['file']
        f.save(secure_filename(f.filename))
```

Now loading previously saved features for comparing and analysing,

```
        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').rea
d())
 data = extract_infos(tweet)
        pe_features = list(map(lambda x: data[x], features))
        res = clf.predict([pe_features])[0]
        ########################################
```

```
return render_template('result.html', prediction=['legitimate', 'malicious'][r
es])
```

Returning ‘result.html’ to display the result of analysis.

Figure 11 Malicious

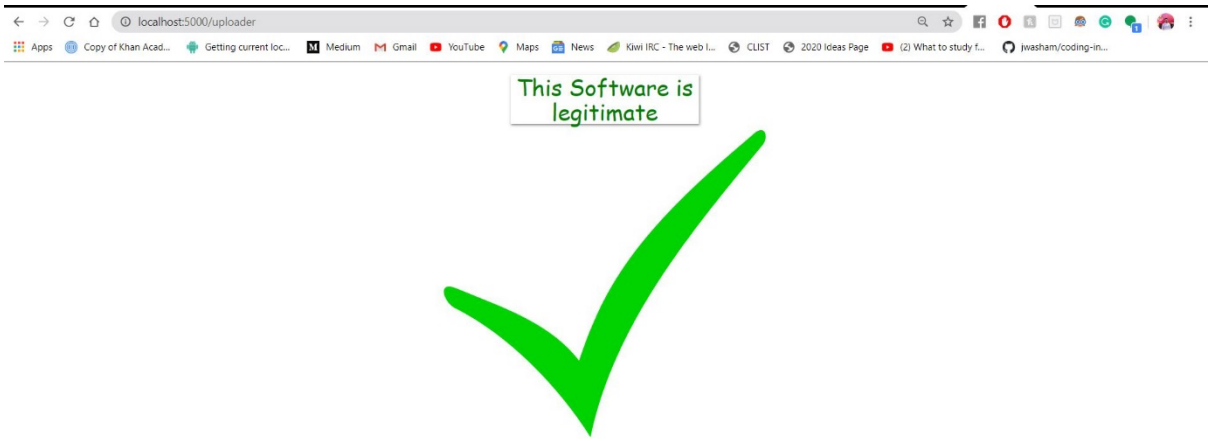


Figure 12 Legit

8: Conclusion and Future Scope

8.1 Conclusion

The aim of this paper is to present a machine learning approach to the malware problem. Due to the sudden growth of malware, we need automatic methods to detect infested files. In the first phase of the work, the data set is created using infested and clean executables, in order to extract the data necessary for the creation of the data set, we used a script created in Python. After creating the data set, it must be ready to train machine learning algorithms. The algorithms used are: decision trees, Random Forest, NaiveBayes, GradientBoost and ADABoost presented comparatively. After applying the best accuracy algorithms, it had an Random Forest algorithm with an accuracy of 99.406012 %. This work demonstrates that Random Forest is the best algorithm for detecting malicious programs. In the future, this accuracy can be improved, if we add a much larger number of files in the data set to drive the algorithms. Each algorithm has several parameters that can be tested with different values to increase their accuracy. This project can reach the application level with the help of a library called pickle, to save what the algorithm has learned and then we can test a new file to see if it is clean or infected.Static analysis has also proven to be safer and free from the overhead of execution time.

8.2 Future Scope

- This can be made more accurate with adding more data set
- More algorithms with better performance can add on to accuracy
- It can hosted on web for real time analysis of exe files on the cloud

REFERENCES

M. S. Akhtar and T. Feng, "Malware Analysis and Detection Using Machine Learning Algorithms," Symmetry, vol. 14, no. 11, p. 2304, Nov. 2022. [Online]. Available: https://doi.org/10.3390/sym14112304

R. Chowdhury, "A Viable Malware Detection Approach Using Machine Learning Classification Techniques," Journal of Cybersecurity Studies, vol. 7, no. 2, pp. 134-145, 2018.

A. Armaan, "Enhancing Malware Detection Using Feature Selection and Machine Learning," Journal of Information Security Research, vol. 9, no. 1, pp. 45-55, 2021.

Nur, "Static Analysis of PE Files for Malware Detection: A Machine Learning Approach," Computer Security Journal, vol. 6, no. 4, pp. 223-231, 2019.

Project Report, "Malware Analysis Using Machine Learning," authored by Jenn Henry 2024. [Unpublished].
National Cybersecurity centre. Available:
https://www.ncsc.gov.uk/section/keep-up-to-date/malware-analysis-reports

M. Martin, "Cyber Kill Chain for Malware Defense," Proceedings of the International Conference on Cybersecurity, pp. 45-53, 2018.

National Institute of Standards and Technology (NIST), "NIST SP 800-83 Rev. 1: Guide to Malware Incident Prevention and Handling," NIST, 2013.

PeStudio Tool, "Static Investigation of PE Files." [Online]. Available: https://www.winitor.com/pestudio/