Beyond Signatures: An ML Based Two-Stage Engine for Early Ransomware Detection on Windows Systems

Ouedraogo Martial[1] [0009-0009-8592-8601], Nikiema Benito[2] [0009-0008-2031-285X], Tamiano Banda[3] [0009-0006-7674-7647] and Subrata Sahana[4]

[1,2,3,4]*Department of Computer Science and Engineering*
School of Engineering and Technology
Sharda University, Greater Noida, UP, 201306, India

[1]*2020000338.martial@ug.sharda.ac.in,*
[2]*2020819715.nikiema@ug.sharda.ac.in,*
[3]*2020000117.tamiano@ug.sharda.ac.in,*
[4]*subrata.sahana@sharda.ac.in*

Abstract

Ransomware remains one of the most prevalent threats in the Information Technology landscape. It is a type of malware that blocks access to the target resource for a ransom payment. The damage occurred can be critical for affected organization as there is no guarantee even if the ransom payment is paid. Hence there is need for early detection. To address this situation, we propose an early detection approach based on two stage detection. The first stage is a static detection engine that identifies ransomware based on the PE header. The second stage is a dynamic detection engine that will analyze the predicted negatives from the first stage to detect polymorphic and zero-day ransomware. This detection engine is based on the early Application Programming Interface (API) calls. The ransomware detected at both stages will have their signature stored in a database for future detection. The first engine is optimized for low false positive while the second engine is optimized for low false negative. We evaluate the two-stage detection with many machine learning algorithms. Based on the results, the static engine achieves an accuracy of 98.11% with Gradient Boost and the dynamic analysis achieve an accuracy of 98,32% with Random Forest after 10-fold cross validation. The combination of both engine in a two-stage detection obtained an accuracy of 97.83% with 0% false negative rate and 4.76% false negative rate when evaluated against unknown ransomware from different families. This result is achieved when Gradient Boost is selected for the static detection engine and Random Forest is selected for the dynamic detection engine.

Keywords: *Ransomware Detection, Static Analysis, Dynamic Analysis, Portable Executable Header, Application Programming Interface, Machine Learning*

I. Introduction

Ransomware have emerged in the recent years as one of the most active threats to the digital security of companies across the world. The Wannacry outbreak in 2017 that cost around $4 billion symbolizes this recent surge of ransomware [1]. This unprecedented attack demonstrated the devastating potential of ransomware. It kept

growing ever since with a multiplication of ransomware outbreak each year. This situation can be attributed to the COVID-19 pandemic that forced many companies to adopt a remote working approach ,increasing the attack surface for hackers [1].Besides, this situation is correlated to the development of RaaS (Ransomware as a Service) because it made it easy and simple for non-technical people to craft sophisticated ransomware attacks [2]. As a result, an increase of 51% in 2020, 37% in 2021 ,66% in 2022 and 2023 [3].

Ransomware can be defined as any code that is unconsented, run on a digital system to block the legitimate user access to resource for a ransom [4]. Like other types of malware, it uses similar evasion techniques to propagate and infect more systems. Ransomware is generally categorized into Crypto-ransomware and Locker-ransomware. They respectively encrypt the victim files and block the access to the files of the target computer. It also as a wide range of attacks as Windows, Linux, Mobile and Internet of Things (IoT) ecosystem are regularly the target of attacks. Naturally, many detection solutions have been developed to tackle the ransomware threat. The detection approach can be categorized in static analysis and dynamic analysis.

Static analysis is based on the structural information of the code of the ransomware. It is fast but can be easily bypassed by obfuscation, packing and recompiling [5]. It examines the binary file of the ransomware without execution, and can reveal important information about the malicious file like the control infrastructure, targets, behavior and persistence mechanism. Static analysis provides features like PE header, hashes, strings opcodes, byte sequence that can provide us insight of the malicious code behavior [6].One of the most important benefits of static analysis is the speed because it gives faster results compared to other type of analysis. On the other side it is obsolete against current variant of ransomware the uses obfuscation techniques [7].We will focus on the PE header for this research.

The PE header, or Portable Executable header, is a crucial component of executable files (.exe), object code (.obj), and Dynamic Link Libraries (DLLs) used in Windows operating systems and some other environments. It acts like a roadmap, providing essential information for the operating system to load and execute the program correctly. The structure of the PE header is shown in figure 1. The PE header mainly consist of the MS-DOS stub, the PE signature, the CODD file header and the optional header. The PE header is followed by the section header and the different sections. Here is a brief description of each part:

- MS-DOS stub is a legacy mark form the MS-DOS era. It was first intended to allow basic execution of PE files. It became obsolete to modern loader. It consists of a 64-byte header followed by some code displaying “This program cannot be run in DOS mode”.
- PE signature is a fixed 4-byte value generally displayed as PE\0\0 that indicates the file format as Portable Executable
- COFF stands for Common Object File Format. It is a mandatory header that gives critical information about the file’s object code structure valid for both executable and object files.
- Optional Header provides more details relative to PE files. It is mandatory for executables but not object files.
- Section table describe logical division of the PE file, each section with specific code or data.

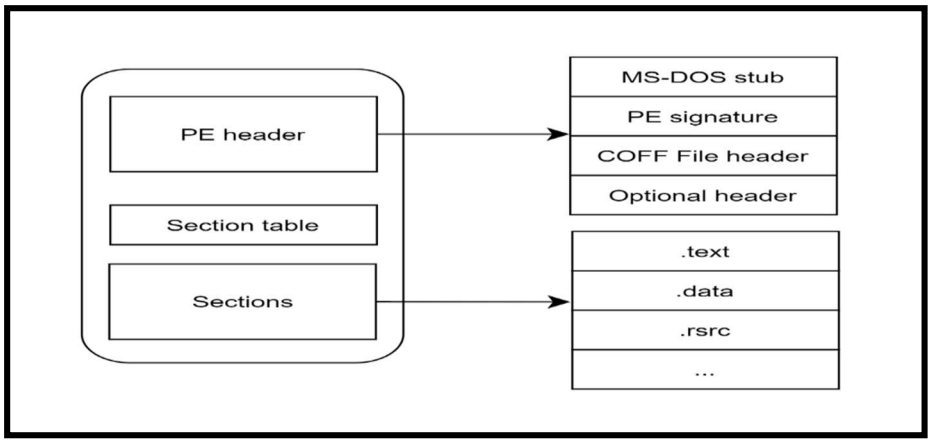

Figure 1. PE header structure [8]

Dynamic analysis investigates the behavior of the ransomware generally in an isolated environment. It is more robust but can be slow as the ransomware is run on an isolated environment. It requires a sample of a malicious software to be run in a restricted and monitored environment different from the host system. This enables to observe the behavior of that specific malware without infection risk. For that purpose, virtual machines and sandboxes like cuckoo sandbox are used to monitor the malicious activity of the ransomware. The report includes the API calls pattern or frequency, the files and process created, the network traffic and other relevant information. Dynamic analysis is considerably slower than the static analysis. Nevertheless, it is more robust against polymorphic ransomware and zero-day ransomware [6].We will focus on API call frequency in this research.

API (Application Programming Interface) calls are requests made by a software program to interact with the operating system or other software components. They make it possible for programs to access different system functions without having to comprehend the complex workings of how those functions are implemented. Common API calls associated with ransomware include file system operations (creating, reading, writing, and deleting files), registry access, encryption and decryption functions, and network-related functions (e.g., socket creation, data transmission) [8].By analyzing API call, we can detect malicious behavior such as file encryption, or attempts to disable security software.

Both approaches have their strength and weaknesses making the timely and correct classification of ransomware a difficult task. Detecting ransomware as early as possible is crucial to prevent any critical damage.

Therefore, this paper proposes a two-stage solution. The first stage of detection is based on the PE header analysis. The predicted negatives from the first stage will undergo a second detection based on early 45 seconds API calls frequency analysis. The SHA256 signature of the ransomware detected at any stage is stored in a signature database for future quick detection. We evaluated both detection engine separately with different machine learning classifiers The static detection engine is optimized for low false positive and the dynamic detection engine is optimized for low false negative. The overall approach favors early detection of the ransomware with minimal false negative rate and tolerant false positive rate.

The major contribution of our research is outlined as follows:

- To prepare an open-source dataset of the frequency of the early API calls of ransomware and benign softwares.
- To evaluate how well ransomware can be distinguished from benign software based on the frequency of the API calls captured in the first 45 seconds of execution
- A novel two stage solution based on PE header and API calls frequency for early detection of ransomware in windows platform with minimal false negative rate.

The rest of the paper is organized as follows. Section II delves into related work about ransomware detection. Our proposed novel solution is explained in section III. The evaluation and results of the research is described in section IV. Finally, the section V consists of the conclusion and future work.

II. Related work

The ransomware detection landscape has gone a long way with the researcher developing new security mechanism to tackle ransomware threat. This section delves into the various advances achieved in the field. The major focus of the research has been shifted towards machine learning based approaches as they offer robust detection against zero-day ransomware [6].

Wani et al. [9] implemented a solution for IoT environment that extracts CoAP headers as well as TCP/IP headers and uses machine learnings algorithms. In that study an accuracy of 98% was achieved using NB algorithm and PCA.

Hwang et al. [10] proposed Markov model, random forests technique in a two-stage detection approach. They have collected 3048 samples in all, 1909 of which are ransomware and 1139 of which are typical. The API sequences were obtained by means of the Cuckoo Sandbox. Markov chain model is used in the first phase, then in the second phase, the residual data is classified using the RF method. This model has an overall accuracy score of 97.28%,

Azween et al. [11] pre-encryption detection technique. The goal is to detect the ransomwre before the encryption process terminates. This technique, known as the Pre-Encryption Detection Algorithm (PEDA), works at two different detection levels. The system first looks for matches with known ransomware signatures. Then, it uses RF technique to identify unknown ransomware. The overall accuracy obtained is 99%.

A detection method based on the random forest algorithm was elaborated by Khammas et al. [12]. The dataset they used accounted 1680 binary samples, with the first half made of ransomware, and the second half of benign files. Their approach mainly consisted of four steps which include data gathering, preprocessing, feature selection and classification. Khammas et al stated an accuracy of 97,74%.

Almomani et al. [13] presented an evolutionary machine learning based detection method for the Android environment. It optimizes training and testing by extracting API calls from an unbalanced dataset. This approach is meant to tackle the practical aspect of ransomware detection with limited data in android devices. A detection accuracy of 97.5 was reached using SVM algorithm.

Sharma et al. [14] developed a detection method based on the app permission, text, image and java methods (Lock, Encrypt, Encode). They used a dataset of 2076 APK

ransomware coupled with 2000 APK non ransomware. The accuracy obtained is 98.08%.

Usharani et al. [15] created a solution based on dynamic features, such as CPU, network data, and privilege evaluation using methods including Adaboost, Random Forest, Gradient Tree Boost, Support Vector Machine, Linear Regression, and Naïve Bayes. The outcomes demonstrated that the SVM algorithm yields the highest level of accuracy with 98.45%.

Talabani et al. [16] utilized Bitcoin transaction data and Rule-Based algorithms to categorize Bitcoin ransomware attacks. Ten descriptive and decision characteristics and 61,004 addresses were included in the Bitcoin dataset. categorization using partial decision trees (PART) fared better 96.01% accuracy in classification using decision tables.

Ahmed et al. [17] offered a novel approach to system behavior-based ransomware threat detection. Peeler combines machine learning models with rule-based detection methods, such as identifying malicious commands and I/O pattern matching, to increase detection accuracy and decrease detection times. The machine learning models are employed to precisely identify ransomware that evades the pattern matcher, such as Crypto-ransomware and Screen Locker malware. The I/O pattern matcher is used by the system to detect the majority of Crypto ransomware. The accuracy yielded is 99.52%.

The access levels of the process memory were used in Singh et al. [18] solution. It helps to identify the ransomware key features with an accuracy of 96.28%.

A technique for differentiating between ransomware and safe file sharing traffic via SMBv2, SMBv3, and NFS protocol was put into place by Berruata et al. [19]. Three aspects are extracted by the method: write bytes, read bytes, and control commands. This extraction is done in a constrained amount of time. Neural networks, tree ensembles, and decision trees have all been used; the neural network produced the best outcomes at 97.7%.

Albanaa et al. [20] developed a hybrid detection method for ransomware detection. It employs both dynamic and static analysis to extract API calls and file headers. Research environments are safe when Cuckoo Sandbox and VirtualBox are used. The study evaluates various models by analyzing 324 ransomware, 320 other malware, and 315 goodware samples from VirusShare. Of these, Random Forest achieves the greatest accuracy of 96% and the lowest false positive rate of 0.018.

Herrera et al. [21] proposed a detection technique based on static analysis. The gathered samples are processed in a sandbox setting so that the dynamic characteristics can be taken out. The machine learning algorithm uses those attributes as input. An accuracy of 99.63% was achieved.

Deng et al. [22] proposed a static analysis framework based on portable executable (PE) header to provide early ransomware detection with the use of Deep Reinforcement Learning. The accuracy obtained from the work is 97.9%.

M. Farnoush et al. [23] proposed a static approach for ransomware detection using specific sections of the PE Header of Windows portable executable files. both malign and benign binary headers were compares using Needleman Winsch algorithm and calculation of alignment scores. Also 10-fold cross validation was applied on all results. The method made use of a total of 3 datasets, and the accuracy stated was 95%.

The different existing methods are compared in the table 1.

Table 1. Comparison of Existing Works in Ransomware Detection

Ref.	Features	Accuracy	Merits	Limitations
Wani et al. [9] 2020	Resource ID, Resource Type, URI, Multipurpose Internet Mail Extensions (MIME) type, Application & message keys, IoT device's IP address	98%	Network focused, IoT environments oriented	Limited research about locker ransomware
Hwang et al. [10] 2020	API calls	97.%	Strong detection due to a two-model architecture, Low false negative rate	High false positive rate
Azween et al. [11] 2020	API calls	99%	Early detection capabilities, High accuracy, Highly practical	Little study over static features
Khammas et al. [12] 2020	Raw bytes	97.74%	High focus on windows platform	Limited research on dynamic features
Almomani et al. [13] 2021	API call permission	97.5%	High focus on android platform	Vulnerable to ransomware with camouflage abilities
Sharma et al. [14] 2021	Apps permissions, intents, text, images, java methods (Lock, Encrypt, Encode)	98.08%	Good efficiency regardless of imbalanced data	Does not work well against obfuscated samples
Usharani et al. [15] 2021	IP address, File metadata, URL features, HTTP connections, ports, MIME, Evaluated privilege, Extensions, CPU usage, Process ID, Services, Payload features	98.45%	Architecture independent,	Little focus on static features
Ahmed et al. [17] 2021	Process starts, Process ends, DLL image loads, DLL image unload, File reads, File writes, Thread starts, Thread ends	99.52%	Cross platform, Architecture independent, High accuracy	May fail in detection of evolving and polymorphic ransomware which can evade i/o pattern matching
Talabani et al. [16] 2022	N/A	96.01%	Detection process involves bitcoin transaction information	Does not make use of traditional features
Singh et al. [18] 2022	Access privileges: Read, Write, Execute, Copy	96.28%	Cross platform, Architecture independent	Insufficiency of the access levels and size of the training set
Berruata et al. [19] 2022	Written bytes, Read bytes, Control commands	99.7%	network environment oriented, High accuracy	Locker ransomware is not tested
Albanaa et al. [20] 2023	PE header, API calls	96%	Low false positive rate, Other malware categories are included in the research	Not enough work on the false negative for a more balance

				detection
Herrera et al. [21] 2023	System calls, Processes and process trees, Modified system registries, Files and directories created, modified, or deleted, Network connection established, Network protocols used	99.63%	High accuracy, High focus on behavioral features	Vulnerable to ransomware with data exfiltration, little discussion of dynamic behavior
Deng et al. [22] 2023	PE header	97.9%	Provide early and efficient detection	Vulnerable to ransomware using obfuscation techniques
Manavi et al. [23] 2023	PE header	95%	Use of a large dataset of samples	Limited study over dynamic behavior

III. Proposed two-stage ransomware detection

This section delves into the two-stage detection and the machine learning workflow.

1. Architecture

Our method is based in two model: The first one uses PE header features while the second one uses early API calls frequency. The flow of the detection is as presented in figure 3. Whenever a new file is downloaded or transferred to the system, the SHA-256 hash of the file is created and is compared against a database of ransomware file signature. This is a fast and trivial process as the goal there is to remove generic and already detected ransomware file. But it is obsolete against sophisticated ransomware [1].

Then, the PE header features of the file will be extracted. Those features will be passed down to the static model for the prediction. Based upon the result, we have two options. Either the file is a ransomware and its signature will be stored and alert generated; or it is flagged as benign and it will undergo the second detection process. Because some ransomware can easily bypass the static analysis with code obfuscation [5] [7], this detection can be incomplete. Nevertheless, it does reduce considerably the number of files allowed for the next step.

Finally, the files that made it to this level will undergo detection through the dynamic model. This model is based on the frequency of the early API calls. A time limit of 45 seconds of execution is set. The number of API is also limited to 550000 API calls because some processes produced enormous amounts of calls. The goal here is to capture and analyze API calls before any important damage can be done [35]. This enables to reduce the extraction time. After the final prediction of the dynamic model, the files flagged as ransomware will have their signature stored in the database and their execution blocked while the ones flagged benign will execute normally. The signatures of benign files are not stored because they can be compromised by some ransomware to bypass security measures [11]. Figure 2 and figure 3 illustrate the approach as pseudocode and flowchart of the two-stage detection respectively:

```
#Initial Signature-based Filtering:
    Select the file to analyse
   Calculate the SHA256 hash of the file.
   Check if the hash exists in the Ransomware Signature Database.
         If yes, the file is known ransomware, so generate an alert and stop
         If not, proceed to the next step.

#Static Model Prediction:
        Extract features from the Portable Executable (PE) header of the file.
        Use a static model to predict if the file is ransomware based on these features.
              If the prediction is ransomware:
                   Generate an alert for ransomware detection
                   Add the file's hash to the Ransomware Signature Database and stop
             If the prediction is benign, proceed to dynamic detection

#Dynamic Model Detection:
             Analyze early 45 seconds API calls' frequency in the file
             Use a dynamic model to predict if the file is ransomware based on these API call
frequencies.
                    If the prediction is ransomware:
                          Block the file and generate alert
                          Add the file's hash to the Ransomware Signature Database.
                    If the prediction is benign:
                          Allow the file's execution.
                         Return "Benign file, execution allowed"and stop
```

Figure 2. Pseudocode of two-stage detection

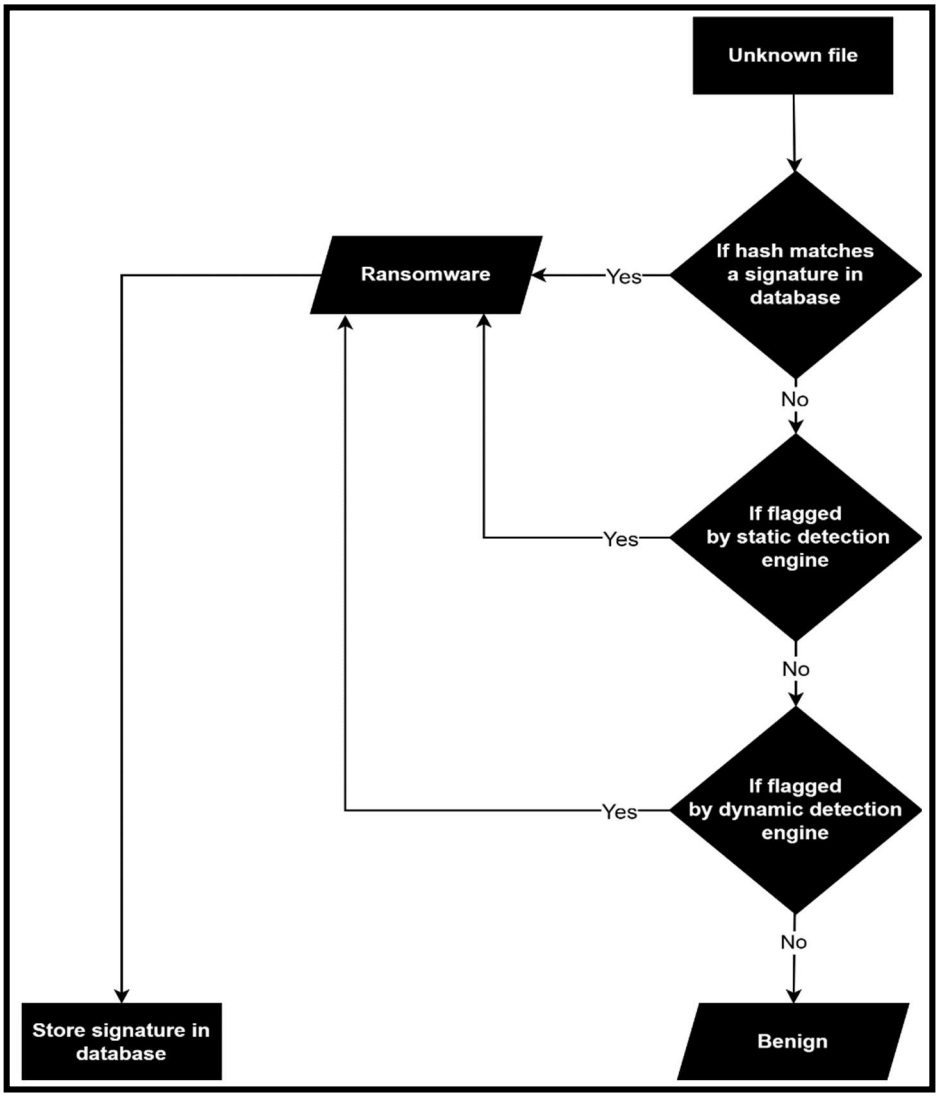

Figure 3. Working of the Proposed Detection Framework

2. Signature database

The signature database's objective is to prevent the verification of known ransomware. The burden on the other primary detection engines can then be decreased. Only the ransomware files' SHA-256 signatures will be kept in this database. The benign files are deleted because there is a chance that malicious activities could be carried out by compromised legitimate software. That why we store only the signature of ransomware. This process is illustrated by figure 4.

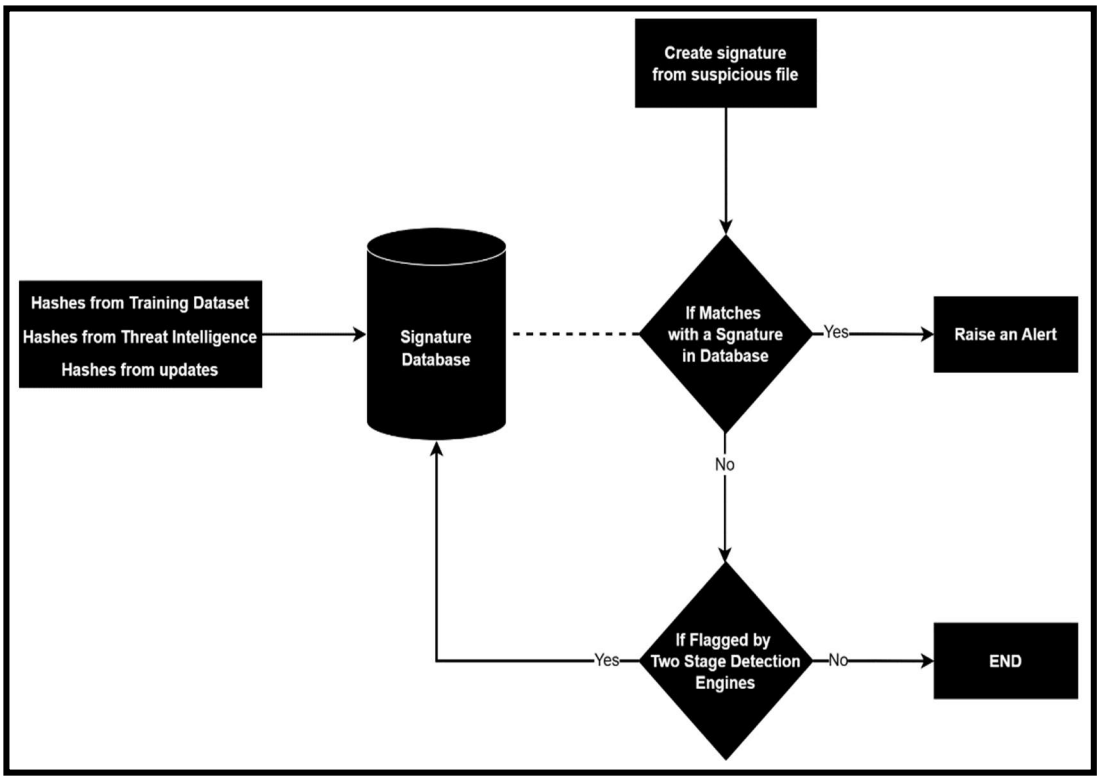

Figure 4. Working of the Signature Database

3. Static detection engine

The static detection engine is the first line of defense. It is mainly based on the PE header information extracted from exe files as shown in figure 5.

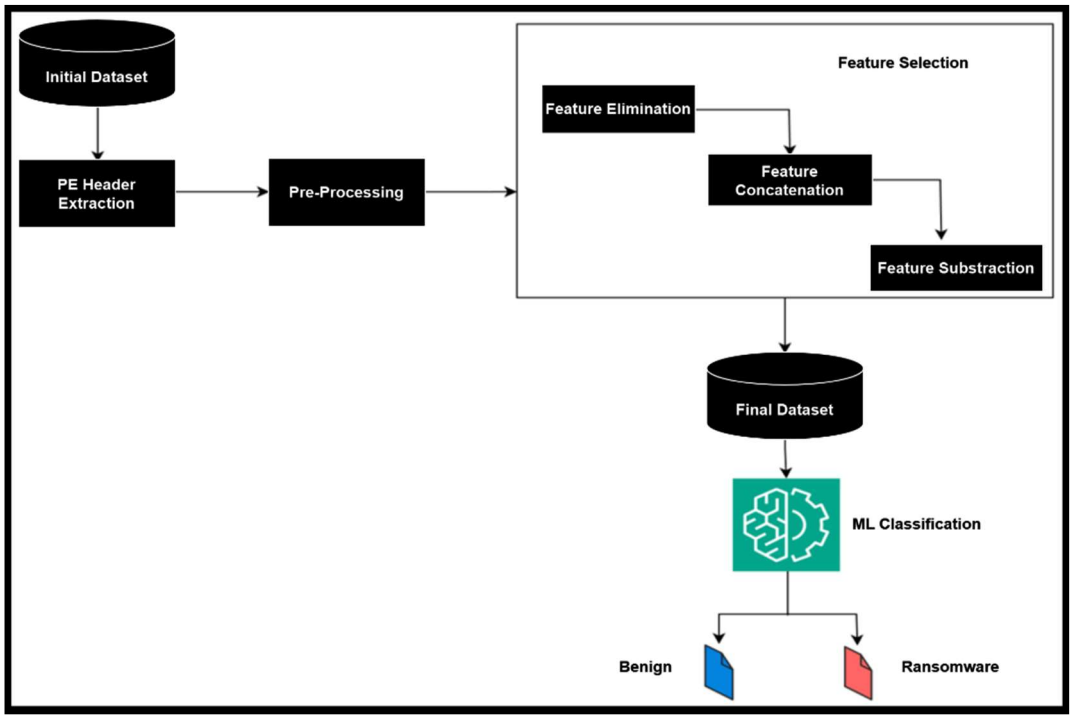

Figure 5. Static Engine Implementation

3.1. Dataset description

For the static engine implementation, we use the PE header of 1484 executables, as shown in table 1. In this dataset we have 729 benign files taken from their respective official trusted websites and the 760 ransomware samples from 71 families taken from Vx Underground [24]. Those ransomwares are presented in table 2.

Table 2. Dataset repartition for the static detection engine

	Benign	Ransomware	Total
Number of static samples	729	760	**1484**

Table 3. Ransomware Samples Repartition for the static detection

Family	Number of exe	Family	Number of exe
AESRTR	1	Koxic	1
AXLocker	4	LockBit	78
Adhubllka	18	LockerGoga	19
Agenda	2	LokiLocker	48
Akira	4	Lorenz	15
AvosLocker	4	Maze	105
Azov	2	MedusaLocker	9
Babuk	11	Meow	1
BandarChor	7	Midas	1
BianLian	11	Moisha	1
BlackBasta	6	MortisLocker	4
BlackByte	6	NightSky	2
BlackCat	15	Nokoyawa	10
BlackSnake	1	Onyx	1
BlueSky	1	PLAY	4
Cactus	1	Pandora	2
Cerber	3	Paradise	1
Chaos	14	Phobos	9
Cl0p	3	REvil	97
Clownic	1	Rhysida	9
Cryptolocker	57	Roadsweep	1
Cryptowall	2	Rook	4
Crysis	1	Royal	2
Crytox	3	Ryuk	5
Cuba	4	SFile	11
Curator	1	Samsam	3
DarkBit	1	ScareCrow	3
Darkside	17	Sugar	12
DearCry	1	SunnyDay	1
Decaf	2	SynAck	1
Diavol	3	Trigona	1
EvilNominatus	4	Venus	1
GandCrab	78	Vohuk	3
Goodwill	1	WannaCry	1
Haron	1	Yanluowang	2
Jaff	1	**Total:**	**760**

3.2. PE header feature extraction

For the PE header features, a dedicated python library called PE file was used [25]. It is a python module that allows us to extract information from portable executable files. It can retrieve the PE header and section information [8].For the creation of the dataset for the static model the features form the PE Optional header and section table information are extracted.

A total of 69 features are extracted for each sample. Among them there are 6 features from the PE file header, 21 from the optional header, 32 from the data directories and 10 from the section table. These features help to characterize ransomware samples

without the need of executing them.

3.3. Feature selection

To choose the best features for increased performance, some features engineering is needed.

Dropping irrelevant columns

Some features in the PE header are not indicative of the malicious characteristic of PE file because they hold the same values for every file and are not impacting in the prediction of the model. So we dropped the following feature: SizeOfOptionalHeader, Magic, SizeOfStackReserve, SizeOfHeapReserve, ExceptionTableSize, CertificateTableRVA, ArchitectureSize, BoundImportSize, IATRVA, ReservedSize.

Combining features by concatenation

Features like the MajorOperatingSystemVersion and MinorOperatingSystemVersion do not have a full meaning on their own. They need to be combined into one single feature. We do it by concatenating them as shown in equation (1):

$$\forall x \in X\,, \forall y \in Y \rightarrow xy \in Z \quad (1)$$

Where: X= major feature, Y=minor feature and Z= combined feature

The table 3 describe the new features obtained and the features dropped

Table 4. Feature Concatenation

Major feature	Minor feature	combined feature
MajorLinkerVersion	MinorLinkerVersion	LinkerVersion
MajorOperatingSystemVersion	OperatingSystemVersion	OperatingSystemVersion
MajorImageVersion	MinorImageVersion	ImageVersion
MajorSubsystemVersion	MinorSubsystemVersion	SubsystemVersion

Combining features by subtraction

The features from the section table depicts the virtual size and the raw size in memory of the different section of the exe file. Generally, the raw size is greater or equal to the virtual size. Hence, cases where the virtual size is greater than the raw size are suspicious although not in every case [26].So we create new features based on computation of the difference between the SizeOfRaw and VirtualSize, as shown in equation (2):

$$\forall x \in X, \forall y \in Y \rightarrow (x - y) \in Z \quad (2)$$

The table 4 describes the new features obtained and the features dropped

Table 5. Feature Subtraction

SizeOfRawData	VirtualSize	Obtained Feature
.text_SizeOfRawData	.text_Misc_VirtualSize	.text
.rdata_SizeOfRawData	.rdatat_Misc_VirtualSize	.rdata
.datat_SizeOfRawData	.data_Misc_VirtualSize	.data
.rsrc_SizeOfRawData	.rsrc_Misc_VirtualSize	.rsrc
.reloc_SizeOfRawData	.reloc_Misc_VirtualSize	.reloc

After the feature selection process, 50 features remain and will be used for the training of the static detection engine. Those features are described in table 5

Table 6. Final Features

Features		
Machine	ImportTableRVA	BoundImportRVA
NumberOfSections	ImportTableSize	IATSize
TimeDateStamp	ResourceTableRVA	DelayImportDescriptorRVA
PointerToSymbolTable	ResourceTableSize	DelayImportDescriptorSize
Characteristics	ExceptionTableRVA	CLRHeaderRVA
SizeOfCode	CertificateTableSize	CLRHeaderSize
SizeOfInitializedData	BaseRelocationTableRVA	ReservedRVA
SizeOfUninitializedData	BaseRelocationTableSize	.text
AddressOfEntryPoint	DebugRVA	.data
BaseOfCode	DebugSize	.rdata
ImageBase	ArchitectureRVA	.rsrc
SizeOfImage	GlobalPtrRVA	.reloc
CheckSum	GlobalPtrSize	LinkerVersion
Subsystem	TLSTableRVA	OperatingSystemVersion
DllCharacteristics	TLSTableSize	ImageVersion
ExportTableRVA	LoadConfigTableRVA	SubsystemVersion
ExportTableSize	LoadConfigTableSize	**Total: 50**

4. Dynamic detection engine

The dynamic detection engine is the second and last defense. It takes the relay to analyze the samples labelled benign by the static detection and gives the final verdict. This engine analyzes the API calls frequency from the processes of each sample. Figure 6 explain the process of elaboration of the dynamic detection engine.

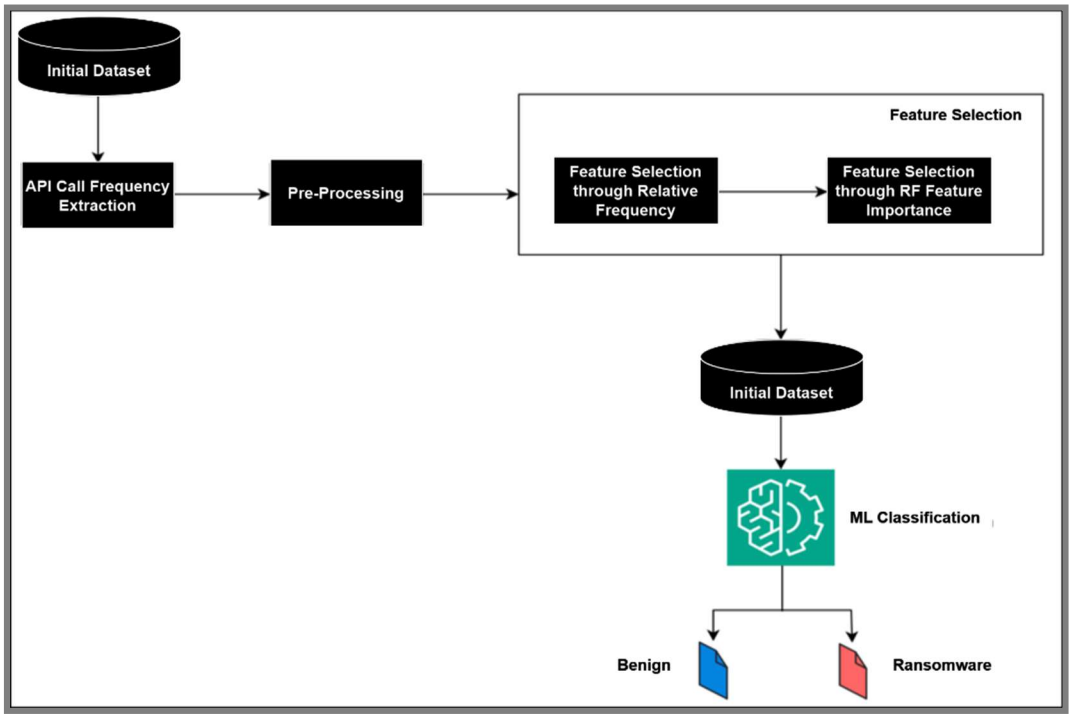

Figure 6. Dynamic Engine Implementation

4.1. Dataset description

The dataset is a composed of 345 samples consisting 195 benign software and 150 ransomwares. However, this dataset, is based on the API calls extracted from each sample. The dataset is very diverse to capture real world diversity of the ransomware and benign samples. Table 6 shows dataset repartition, while table 7 shows ransomware repartition specifically.

Table 7. Dataset repartition for the dynamic detection engine

	Benign	Ransomware	Total
Number of dynamic samples	195	134	**329**
Number of processes	261	274	**535**

Table 8. Ransomware analyzed for the dynamic detection

Family	Number of exe
AESRTR	1
Agenda	2
Akira	4
AvosLocker	4
AXLocker	4
Azov	2
BandarChor	7
BlackBasta	5
BlackSnake	1
BlueSky	1
Cerber	1
Chaos	8
Cryptolocker	10
Crysis	1
Cuba	1

Family	Number of exe
Curator	1
DarkBit	1
Darkside	14
DearCry	1
Decaf	1
Diavol	2
EvilNominatus	1
Haron	1
LockBit	10
LokiLocker	10
Lorenz	10
Maze	20
MedusaLocker	5
Phobos	5
Total:	**134**

4.2. API calls feature extraction

To capture the early API calls, API monitor is used. It is a software that allows the capture of API calls and related information for both x32 and x64 on windows computer. A time of 45 seconds is set to keep only the early calls. In addition, a maximum number of 550000 is set to limit the capture calls as some processes produces enormous number of calls. Without a limit the number of API captured can take a lot time to process. As our research is based on early detection, a number limit is necessary. As a result of this limit, some processes have their monitoring stopped before the first 45 seconds. Only the API relevant to ransomware encryption and related suspicious behavior are monitored.

A total of 3600 unique API is captured across the different processes of the whole dataset. Then the frequency of each unique API is determined by cumulating the number of times a process calls this particular API. The final dataset obtained consist of 535 rows and 3600 columns. This dataset is available in this GitHub repository (*https://github.com/Benivio/two-stage-detection*).

4.3. Feature selection

For an improved performance and less use of resources, there is a need to reduce the dimensionality of the dataset. For that, the features are reduced through relative frequency and Random Forest feature importance. Figure 7 illustrates the selection process

Figure 7. Feature Selection for the Dynamic Model

Feature selection by relative frequency

To have the most important features, we proposed a feature selection based on two types of frequencies. We have the value frequency of the API calls and the binary frequency of the API calls.

- **Value frequency:** The value frequency is based on the average of values of each column. We separate the dataset in two categories that are the API calls having more value for the benign and the ones having more values for ransomware. This approach characterizes the important API calls but can lead to error if there is an outlier with a huge number of calls. After this phase we get 864 API for the ransomware and 2700 for the benign softwares. The pseudo-code for the relative frequency approach is shown in figure 8.

```
# Split data into ransomware and benign files
ransomware_calls = Filter rows with 'Label' as 1
benign_calls = Filter rows with 'Label' as 0

# Count total API calls for each class
total_ransomware_calls = Sum values in each column of ransomware_calls
total_benign_calls = Sum values in each column of benign_calls

# Calculate relative frequency of API calls for each class
relative_frequency_ransomware = Divide total_ransomware_calls by number of rows in ransomware_calls
relative_frequency_benign = Divide total_benign_calls by number of rows in benign_calls

# Identify API calls more frequent in ransomware and benign files respectively
api_calls_ransomware_higher = Select API calls where relative frequency in ransomware is greater
api_calls_benign_higher = Select API calls where relative frequency in benign is greater
```

Figure 8. Pseudocode of Value Frequency Approach

- **Binary frequency:** The binary frequency is based on whether there is a non-zero value in each column. If a value is superior to zero, we consider it as 1 and if it is equal to zero then it is considered as 0. The relative average is computed for the calls more present in ransomware and the one more present in benign software. This approach is robust against outliers but does not characterize high frequency API calls. The pseudo-code for the binary frequency approach is shown in figure 9.

```
# Split data into ransomware and benign files
ransomware_calls = Filter rows with 'Label' as 1
benign_calls = Filter rows with 'Label' as 0

# Count total API calls for each class
total_ransomware_calls = Count the number of non-zero values in each column of ransomware_calls
total_benign_calls = Count the number of non-zero values in each column of benign_calls

# Calculate relative frequency of API calls for each class
relative_frequency_ransomware = Divide total_ransomware_calls by number of rows in ransomware_calls
relative_frequency_benign = Divide total_benign_calls by number of rows in benign_calls

# Identify API calls more frequent in ransomware and benign files respectively
api_calls_ransomware_higher = Select API calls where relative frequency in ransomware is greater
api_calls_benign_higher = Select API calls where relative frequency in benign is greater
```

Figure 9. Pseudocode of Binary Frequency Approach

After this phase we get 835 API. Because the benign software is very diverse, we only took the first 1088 most frequent API.

- **Selection:** After computing the relative frequencies, we compute the intersection of the value frequency and binary frequency for the ransomware to get the final ransomware API. Then we do the same for the benign. The result is 699 for the ransomware and 925 for the benign.

  Finally, we perform the union of those results to get the final set of columns which is 1624 columns. These columns represent the API which the higher value frequency and higher binary frequency for each ransomware and benign software.

Random forest feature importance

When dealing with ransomware detection features, we need feature selection that considers the relationship between features. That is the case with the Random Forest feature importance. It gives us insight on the features that contributed in the ability of the model to distinguish between ransomware and benign files.

First, we select an impurity measure in this case, the Gini impurity. Gini impurity measures the probability of misclassifying a randomly selected element in the dataset if that element were randomly labeled according to the distribution of labels in the node.
For a node t with N_t samples and K classes, the Gini impurity $G(t)$ is calculated as shown by equation (3):

$$G(t) = 1 - \sum_{i=1}^{K} p(i|t)^2 \quad (3)$$

Where: $p(i|t)$ is the proportion of samples of the class i in node t.

The feature importance FI for a single feature X is computed based on the decrease in impurity (ΔI) that happens when making use of feature X for dividing nodes in the trees of the Random Forest. The decrease of impurity is calculated for every node where feature X is used for splitting and is determined by the number of samples in each node. Then, the feature importance of feature X is calculated by taking the average of the decrease in impurity across all the nodes in all tress where this feature is applied for splitting, weighted by the number of samples in each node. The decrease in impurity for feature X is calculated as shown in equation (4):

$$\Delta I = N_t * I_{parent} - N_{left} * I_{left} - N_{right} * I_{right} \quad (4)$$

Where:

- Nt is the number of samples in the current node
- I_{parent} is the impurity of the parent node before the split
- N_{left} and N_{right} are the number of samples in the left and right child nodes after the split
- I_{left} and I_{righ} are the impurities of the left and right child nodes after the split

After that, normalization is applied so that the importance rate adds up to 1 for ease of interpretation. The higher the rate, the higher the influence on the model decision.

For our dataset, we selected the features having a rate superior to 0. 00088.That gives us the best 248 features out of the 1624 initial features. The final dataset before training consists of the 248 columns representing the most important API calls frequency.

IV. Evaluation and results

1. Evaluation metrics

The following metrics, represented as equations (5), (6), (7), (8), (9) and (10) are considered for the evaluation of the two-stage detection:

- $Accuracy = \frac{(TP+TN)}{(TP+TN+FP+F\ )}$ (5)
- $Precision = \frac{(TP)}{(TP+\ \ )}$ (6)
- $Recall = \frac{(TP)}{(TP+FN)}$ (7)
- $F1\ score = \frac{(2 \times precision \times recall)}{(precision + recall)}$ (8)
- $False\ Negative\ Rate(FNR) = \frac{(FN)}{(TP+FN)}$ (9)
- $False\ Positive\ Rate(FPR) = \frac{(FP)}{(FP+TN)}$ (10)

2. Evaluation of the static detection engine

For the evaluation of the static detection engine, 7 ML classifiers are used. They are Random Forest, Support Vector Machine-Nearest Neighbors, Decision Tree, Logistic Regression, Gradient Boost, AdaBoost. We applied 10-fold cross validation and calculate the mean form the results of each fold to get more reliable estimate of the performance of the model. Table 8 and figure 10 show the results

Table 9. 10-Fold Cross Validation Performances of the ML Algorithms Used for the Static Detection Engine

ML classifier	Precision	Recall	F1 Score	Accuracy	FNR	FPR
Random Forest	98.09	97.97	98.02	98.05	2.03	1.88
Adaboost	97.09	97.40	97.23	97.24	2.60	3.00
Decision Tree	97.23	96.87	97.03	97.04	3.13	2.92
Gradient Boost	98.41	97.78	98.07	98.11	2.22	1.65
SVM	96.53	92.96	94.66	94.75	7.04	3.42
K-NN	94.39	96.15	95.22	95.08	3.85	5.86
LR	90.90	92.44	91.61	91.44	7.56	9.58

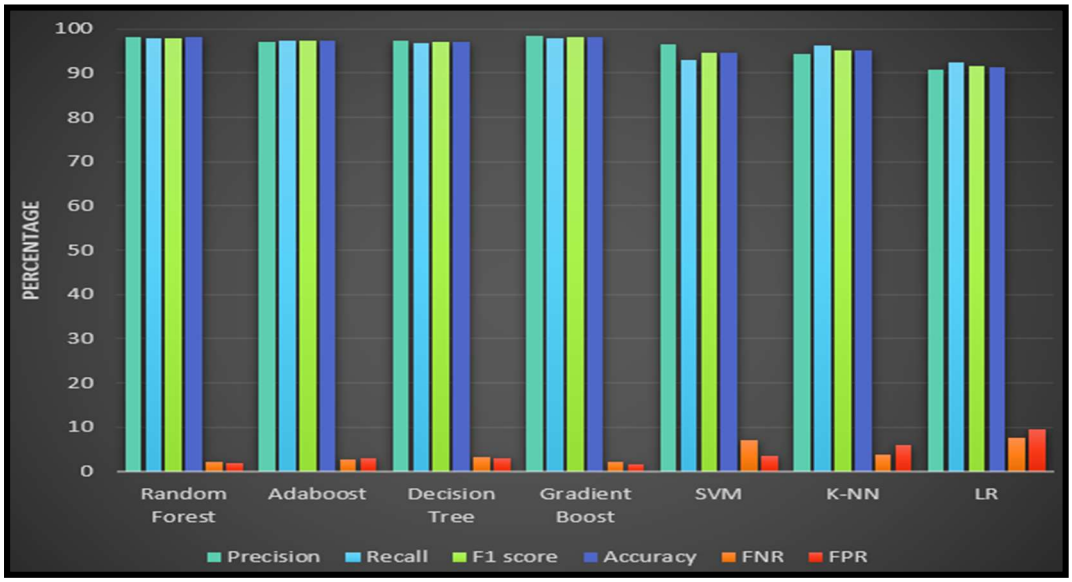

Figure 10. 10-Cross Validation Results for the Static Detection Engine

After the 10-cross validation, the results are mostly similar to the ones of the train-test split with some slight difference. The best model is Gradient Boost with an accuracy of 98.11% with a low FPR of 1.65%.

3. Evaluation of the dynamic detection engine

The testing is done using the train-test split and the 10 cross fold validation. As the last line of defense, the goal of the dynamic engine is to have a low FPR and but more importantly a low FNR. This detection engine is evaluating the processes of the exe files. Hence if one process belonging to a program is flagged malicious, all the other processes of the program are considered the same. But it will not apply in the case where a process is flagged benign. This approach reduces the FNR rate. The goal is to reduce the FNR as much as possible while having a reasonable low FPR.

For the implementation of the dynamic model, we use 8 ML classifiers that are are Random Forest, Support Vector Machine-Nearest Neighbors, Naïve Bayes, Decision Tree, Logistic Regression, Gradient Boost, AdaBoost. After performing 10-fold cross validation, we calculate the mean from every fold the following results are obtained as described in the table 9 and figure 11 below.

Table 10. 10-Fold Cross Validation Results of the ML Algorithms Used for the Dynamic Detection Engine

ML classifier	Precision	Recall	F1 Score	Accuracy	FNR	FPR
Random Forest	98.09	98.66	98.32	98.32	1.34	1.82
SVM	93.42	95.30	94.22	94.01	4.70	7.56
LR	95.79	96.42	96.06	95.87	3.58	4.86
Decision Tree	95.58	96.48	95.90	95.88	3.52	4.50
Adaboost	97.85	97.73	97.74	97.95	2.27	1.63
Naives Bayes	81.96	98.27	89.24	88.19	1.73	22.92
KNN	91.66	92.70	91.97	91.95	7.30	9.44
Gradient Boost	97.38	99.35	98.31	98.31	0.65	2.58

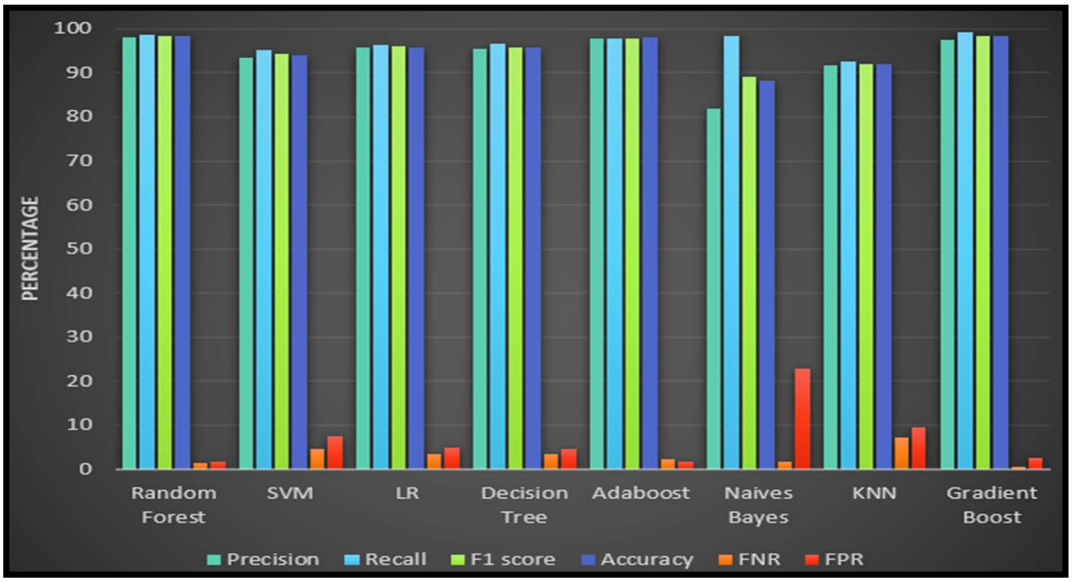

Figure 11. 10-Fold Cross Validation Results of the Dynamic Detection Engine

After the cross validation, we obtained more reliable estimate of the performance of the different models. The best is Random Forest with an accuracy of 98.32%, but with a FNR of 1.34% and a FPR of 1.82%. The second-best model is Gradient Boost with an accuracy of 98.31%, a FNR of 0.65% and a FPR of 2.58%. Even though Gradient Boost has a low FNR, the FPR is higher compare to Random Forest which is more balance with both low FNR and FPR.

4. Evaluation of the two-stage detection with unseen ransomware and benign samples

After testing each engine separately, we perform a combined testing to determine how the combination of the static and dynamic engine perform against unseen data. For that reason, we test them against 46 samples consisting of 26 benign and 20 ransomwares. The ransomwares are from families different than the one used in the training above and are listed in table 10.

Table 11. Ransomwares from Unseen Families

Ransomware Families	Number of exes
Djvu	1
Karma	1
RagnarLock	1
RTMLocker	1
Hive	16
Total:	**20**

First, we extract the PE header features of the 46 executables, including both bening and ransomware. We test them with the Gradien Boost model trained before. It has been chosen as it obtained the best performance. We get the results in table 11 below

Table 12. Results of the First-Stage Static Detection Engine

ML Classifiers	Test exes	Predicted Negatives	FP Samples	FN Samples	Accuracy
GB	46	27	1	1	95.65

From the table above, out of 47 exe samples, there are 1 FP and 1 FN. We also count 27 predicted negatives which are the samples flagged as benign. As described in our two-stage detection architecture, the samples flagged as benign by the static engine will undergo a second test based on the early API calls of their processes. The processes of the test executables are extracted and tested against the classifiers of the dynamic engine trained prior. Random Forest is chosen for the dynamic detection as it showed the best performance in the training phase. The results are displayed in table 12 below.

Table 13. Results of the Second-Stage Dynamic Detection Engine

ML Classifiers	Test exes	Processes	FP Processes	FN Processes	Accuracy
RF	27	38	0	0	100

The RF model has correctly classified all the samples. It successfully detected the ransomware sample that bypassed the static detection. To get the overview of the performance of the two-stage detection we combine the results by considering the final classification of the initial exes. The results are described in the table 13 below.

Table 14. Combined Results of the Two-Stage Detection Engine

ML Classifiers for Two Stage Detection	Test exes	Combined FNR	Combined FPR	Combined Final Accuracy
GB + RF	46	0	4.76	97.83

From the table we notice that the best results of our two-stage detection are obtained when Gradient Boost classifier is chosen for the static detection engine and Random Forest is chosen for the dynamic detection engine. It has an accuracy of 97.83 %. The FNR is 0% which is crucial as ransomware outbreak are so devastating. Only one FP exe has been observed causing a FPR rate of 4.76%.

The results were obtained based of ransomware from different families that the ones used in the training. This shows that this approach is robust against zero-day ransomware.

5. Discussion

The two-stage detection achieved the following points. The first stage of detection based on PE header is a relatively fast process that takes in average 0.5 seconds and the API calls extraction is limited to 45 seconds. In addition, the static detection reduces the number of executables that undergo the second detection, reducing the detection time.

We achieved an accuracy of 98.11% with Gradient Boost for the static detection and an accuracy of 98.32% with Random Forest for the dynamic detection after 10-fold cross validation. To confirm that the two-stage detection, perform well against unseen data, we tested it against ransomware from 5 families, different from the one used in training. We achieve an accuracy of 97.83% with 0 FNR. This is crucial to protect critical infrastructure from ransomware.

In table 14 we compare the two-stage detection against existing work

Table 15. Comparison Against Existing Methods

Authors	Early Detection	Tested Against Ransomware from Different Families	Accuracy
Manavi et al. [23]	✓	✓	95.20%
Talabani et al. [16]	×	×	96.01%
Singh et al. [18]	×	×	96.28%
Hwang et al. [10]	×	×	97%
Khammas et al. [12]	✓	×	97.74%
Two-stage detection	✓	✓	97.83%

As we can see, the two-stage detection provides early ransomware detection, with a high accuracy of 97.83% against unseen ransomware.

V. Conclusion and future work

This paper proposed a two-stage static and dynamic approach for early ransomware detection in Windows environment. The method mainly consists of a static detection engine that classify the files based on their PE header features. The files flagged benign will undergo the final stage of detection based on the API calls frequency. If a file is flagged as ransomware a SHA 256 signature is created and stored for future quick detection. The uniqueness of the approach is that optimize the detection an early detection and low FNR while keeping a reasonable FPR. The two-detection engine are optimized separately to maximize their strength. We achieved an accuracy of 98.11 % with Gradient Boost for the

static detection engine and an accuracy of 98.32% for Gradient Boost and Random Forest after 10-fold cross validation.

To confirm that the two-stage detection, generalizes well against unseen ransomware, we tested both detection engine against ransomware from different families than the ones used for training. We selected Gradient Boost for the static detection engine and Random Forest for the dynamic detection engine. The accuracy obtained is 97.83% with a FNR of 0%.

Our dynamic detection is based on the API calls capture in the first 45 seconds. However, in this lapse of time a big amount of damage can occur on the host computer. Some ransomware takes less than 45 seconds to encrypt files and this can be problematic for this specific framework. That is why a combination with a decoy technique is an interesting alternative to recover potential encrypted files. One other alternative is to explore how early we can distinguish between benign files and ransomware by reducing the capture time of API calls. Third party application like API monitor was used to capture the API. Including all the necessary module for the detection in a standalone application can also reduce the detection time. All these points are to be explored in our future work.

References

[1] A. A. Duong, A. Bello and A. Maurushat, "Working from home users at risk of COVID-19 ransomware attacks," *Cybersecurity and Cognitive Science,* pp. 51--87, 2022.

[2] N. Kshetri and J. Voas, "Ransomware as a Business (RaaB)," *IT Professional,* pp. 83-87, 2022.

[3] Sophos, "The State of Ransonware 2023," Sophos, 2023.

[4] T. McIntosh, A. Kayes, Y.-P. P. Chen, A. Ng and P. Watters, "Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions," *ACM Computing Surveys (CSUR),* vol. 54, no. 9, pp. 1--36, 2021.

[5] R. Sihwail, K. Omar and K. A. Zainol Ariffin, "A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis," vol. 8, p. 1662, 2018.

[6] S. a. F. C. a. M. C. Razaulla, A. Gawanmeh, W. Mansoor, B. C. Fung and C. Assi, "The Age of Ransomware: A Survey on the Evolution, Taxonomy, and Research Directions," *IEEE Access,* 2023.

[7] C. Beaman, A. Barkworth, T. D. Akande, S. Hakak and M. K. Khan, "Ransomware: Recent advances, analysis, challenges and future research directions," *Computers \& security,* vol. 111, p. 102490, 2021.

[8] Microsoft, "learn.microsoft.com," [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format. [Accessed 06 03 2024].

[9] A. Wani and S. Revathi, "Ransomware protection in loT using software defined networking," *Int. J. Electr. Comput. Eng,* pp. 3166--3175, 2020.

[10] J. Hwang, J. Kim, S. Lee and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Personal Communications,* vol. 112, no. 4, pp. 2597--2609, 2020.

[11] S. Kok, A. Azween and N. Jhanjhi, "Evaluation metric for crypto-ransomware detection using machine learning," *Journal of Information Security and Applications,* vol. 55, p. 102646, 2020.

[12] B. M. Khammas, "Ransomware detection using random forest technique," *ICT Express,* vol. 6, pp. 325--331, 2020.

[13] I. Almomani, R. Qaddoura, M. Habib, S. Alsoghyer, A. Al Khayer, I. Aljarah and H. Faris, "Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data," *IEEE Access,* pp. 57674--57691, 2021.

[14] S. Sharma, C. R. Krishna and R. Kumar, "RansomDroid: Forensic analysis and detection of Android Ransomware using unsupervised machine learning technique," *Forensic Science International: Digital Investigation,* vol. 37, p. 301168, 2021.

[15] S. Usharani, P. M. Bala and M. M. J. Mary, "Dynamic analysis on crypto-ransomware by using machine learning: Gandcrab ransomware," *Journal of Physics: Conference Series,* vol. 1717, p. 012024, 2021.

[16] H. S. Talabani and H. M. T. Abdulhadi, "Bitcoin ransomware detection employing rule-based algorithms," *Science Journal of University of Zakho,* vol. 10, no. 1, pp. 5--10, 2022.

[17] M. E. Ahmed, H. Kim, S. Camtepe and S. Nepal, "Peeler: Profiling kernel-level events to detect ransomware," *Computer Security--ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4--8, 2021, Proceedings, Part I 26,* pp. 240--260, 2021.

[18] A. Singh, R. A. Ikuesan and H. Venter, "Ransomware detection using process memory," *arXiv preprint arXiv:2203.16871,* 2022.

[19] E. Berrueta, D. Morato, E. Magaña and M. Izal, "Crypto-ransomware detection using machine learning models in file-sharing network scenarios with encrypted traffic," *Expert Systems with Applications,* p. 118299, 2022.

[20] A. Albanaa, S. Sahana and J. Ali, "Development of a Stochastic Model for the Detection of Ransomware Malware Using Hybrid Analysis and Machine Learning Techniques".

[21] J. A. Herrera-Silva and M. Hernández-Álvarez, "Dynamic feature dataset for ransomware detection using machine learning algorithms," *Sensors,* p. 1053, 2023.

[22] X. Deng, M. Cen, M. Jiang and M. Lu, "Ransomware early detection using deep reinforcement learning on portable executable header," *Cluster Computing,* pp. 1--15, 2023.

[23] F. Manavi, M. E. Samie and A. Hamzeh, "Detecting Ransomware Using Alignment of the Different Sections of the PE Header," 2023.

[24] "vx-underground," [Online]. Available: https://vx-underground.org/samples.html. [Accessed 2023].

[25] "PE file (github)," [Online]. Available: https://github.com/erocarrera/pefile. [Accessed 2023].

[26] M. sikorski and A. Honig, Practical Malware Analyisis, San Francico: William Pollock, 2012.