Automated Hybrid Ransomware Family Classification

George Raul Michael Dunca[a] and Ioan Bădărînză[b]
Department of Computer Science, Babes-Bolyai University, Str. M. Kogalniceanu, Cluj-Napoca, Romania

Keywords: Ransomware, Windows Portable Executables, Random Forest, Hybrid Analysis, Features.

Abstract: Ransomware is one of the most destructive forms of malware that exists today, posing a continuous and evolving threat to everyone from a regular user to a large corporation. Mainly ransomware can be analyzed in three ways: statically which involves extracting information without execution, dynamically which implies running the program in a controlled environment and observing its behavior, and hybrid which addresses the limitation of the previously specified two approaches by combining them. The aim of this study is to maximize the number of features extracted from Windows portable executables (PE) utilizing a hybrid approach and find what are the most useful attributes for differentiating between various ransomware families. A total of 707 samples across 99 families were successfully examined, from which 783 features were identified as the most informative. This data was then used to train a Random Forest model, which conducts the classification. RansoGuard was also developed. This is a graphical user interface Windows application that extracts hybrid attributes from a specified portable executable file. Then it uses the Random Forest model to output a prediction about the ransomware family to which the file belongs and finally generates a detailed report. The results obtained are promising, with the model achieving an accuracy of 71.83%, along with a precision of 0.79 and recall of 0.72.

1 INTRODUCTION

The number of devices has increased in recent years, making the Internet an essential part of daily life for almost every member of society (Aslan and Samet, 2020). This broad connectivity, while offering convenience and accessibility, has also led to a growth in cyberattacks and raised various security concerns. One of the most popular threats in this context is malware, or malicious software, which can compromise personal information or cause damage to services. Malware can be categorized into multiple classes, with the most dangerous being ransomware. This type is installed on the victim's system without their knowledge, and then encrypts valuable information and files, making them inaccessible. The attacker then requests a ransom payment in return for the decryption key. This "business model" is favored by cybercriminals, as evidenced by the fact that last year 72.7% of all organizations fell prey to a ransomware attack (sta, 2024b).

This expansion of ransomware incidents makes malware analysts perform some repetitive tasks when attempting to identify patterns or characteristics of a sample. In case of an incident response, these tasks need to be performed quickly as time is of the essence. Additionally, anti-malware solutions generally rely on signature-based detection as their initial layer of defense, which can be easily bypassed. For instance, malware authors can insert random prints or assignments that will change the sample's signature. Furthermore, many antivirus scanners concentrate more on distinguishing malicious files from benign ones, or on classifying general malware types, rather than focusing specifically on ransomware families.

This study aims to automate the process of identifying the ransomware family of a malicious file. To do so, we first need to extract as much information as possible from various instances. Out of this information, only the most useful attributes will be retained to form the training data for a Random Forest model, which will be used for labeling. To the best of our knowledge, no prior research on ransomware family classification has considered the combination of these multiple features: Strings, Metadata, PE sections, PE headers, MITRE ATT&CK techniques, behavioral signatures, and network activity.

The problems discussed have led to the exploration of the following research questions:

i. What tool combination is the most effective for

[a] https://orcid.org/0009-0005-3236-5666
[b] https://orcid.org/0000-0001-8233-8264

Dunca, G. and Bădărînză, I.
Automated Hybrid Ransomware Family Classification.
DOI: 10.5220/0013065400003825

In *Proceedings of the 20th International Conference on Web Information Systems and Technologies (WEBIST 2024)*, pages 409-416
ISBN: 978-989-758-718-4; ISSN: 2184-3252

extracting the maximum amount of information from portable executables?
ii. Which static and dynamic features are essential for accurate ransomware classification?

This paper will specifically focus on Windows portable executable files since it is the most used operating system (sta, 2024a) and as a result, the primary target for malware authors and cybercriminals.

2 RELATED WORK

Different approaches have been proposed to detect and classify ransomware or malware. The authors of (Rizvi et al., 2022) presented PROUD-MAL, a novel static analysis-based approach for malware detection in portable executables. The dataset used in the study is collected from real-time sources by deploying low and high-interaction honeypots on an organization network, gathering over 15,000 portable executable samples of both malicious and benign samples. Because most of the data was unlabeled, this framework utilizes cascading blocks of unsupervised clustering to create pseudo labels, which are later fed to the Feature Attention-based Neural Network (FANN). This method prioritizes the significant features and finds patterns within a dataset. The framework was compared with some supervised machine learning algorithms and the results show that PROUD-MAL outperforms these algorithms, having an accuracy over 98%.

The study from (Abbasi, 2023) concentrated on automating the process of detecting and classifying ransomware into families, using a dynamic approach. Initially, it suggested a way to choose the appropriate features without the need for expert input, by utilizing Particle Swarm Optimization (PSO). This approach consists of two phases: Stage 1, which employs the Mutual Information Criterion to pick an equal number of top-ranked features, and Stage 2, where additionally an optimal number of features is selected from each family. The procedure was evaluated by using five machine learning algorithms, demonstrating results comparable with other state-of-the-art methods that do not require human intervention and showing the lowest number of features. Furthermore, the study introduced both a Genetic Programming Malice Scoring Method and a Genetic Programming Multi-Model Malice Scorer. Although interesting, it's important to note that these techniques lie outside the scope of this study. Finally, an early detection approach that uses API call sequences is presented. This novelty combines call names with specific call arguments, particularly from system-type API calls, leading to a notable improvement in early ransomware detection performance.

In (Aurangzeb, 2018) the authors focused on the binary classification of ransomware while introducing two hybrid methodologies. The novelty lies in using hardware performance counters for the feature vector. The initial approach, called Hybrid Hierarchy-based Ransomware Classification (HHRC) begins with signature-based analysis and if a match is not found it continues with static evaluation. Furthermore, if the sample is not classified as ransomware, it proceeds to dynamic examination. To address the high dimensionality problem of feature vectors, the authors employed the Information Gain method. The second approach, Hybrid-Combined Ransomware Classification (HCRC), merges static and dynamic attributes into a single feature vector for training a machine-learning model. Testing reveals that HCRC outperforms HHRC by approximately 3% regarding Area Under the ROC Curve (AUC). However, overall performance shows a negligible difference, only 0.01%, but HHRC demonstrates lower computing costs.

3 METHODOLOGY

3.1 Lab Analysis Setup

Before starting the ransomware analysis using a hybrid approach, a safe and controlled environment was needed to proceed. A self-hosted laboratory was favored, choosing an Ubuntu machine for the setup, with Oracle VM VirtualBox installed. As for the virtual machine, Windows 10 Enterprise was used, despite it only offering a 90-day free trial. This configuration was preferred because utilizing different operating systems for the host and guest can enhance security. On the host machine Flare VM was installed, which is a popular collection of software deployment scrips designed for malware analysis. As part of the setup process, it is necessary to disable Windows Defender and Windows Firewall to prevent interference when detonating common malware samples within the environment. Detailed installation steps and additional information about Flare VM are described at (fla, 2024b). To make sure that the host can't communicate with the internet or other devices a Host-only Adapter was employed, which restricts the communication solely between the VM and the computer it runs on. Additionally, a Bridged Adapter was utilized when internet access was required for tasks such as installing dependencies, and packages, or making API calls.

3.2 Dataset Selection

For this study, the samples were obtained from VirusShare (vir, 2024a), a repository containing malware instances from various families. It was created to offer security researchers and forensic analysts access to samples of active malicious code. Specifically, the study utilized the "Special Request" section within the Torrents tab, which includes a Crypto Ransomware entry. There, a zip file was available containing approximately 8GB of ransomware samples, totaling around 38,000 instances. The signatures of these entities were checked to determine if the dataset contained any duplicates, and the result was negative. Some important notes here are the fact that not all the samples were necessarily portable executable files, and they were not labeled with the corresponding family.

3.3 Labeling

Before starting the analysis process, the samples were classified by a script that created a dictionary, with each sample's name as a key and its corresponding family name as the value. The goal was to achieve an evenly distributed dataset, allowing a maximum of 120 entries per label. However, only an upper limit was imposed, resulting in a database with many families, some containing just a few instances. AVClass (avc, 2024), an open-source command line tool, was utilized to classify the data. It processes a VirusTotal (vir, 2024b) JSON report and outputs the most likely family name for the sample. With a free VirusTotal account, only 500 API requests per day are allowed and since submitting a file and retrieving the JSON report requires two calls, 250 entries can be classified daily. During the labeling process, a problem was identified: some samples were not categorized as ransomware families but as various Trojan or Spyware types. Initially, AVCLass was suspected to be the source of the problem, especially because some instances were incorrectly categorized as benign, thus showing false positive results. However, after manually investigating and relabeling some random samples, it was observed that the issue was the VirusShare dataset, which although claiming to contain only crypto-ransomware, does not. Despite these challenges, the decision was to proceed with all the analyzed samples. The final dataset consists of 707 entries, distributed across 99 labels, and the top 5 most populated ones are shown in Figure 1, with a short description below:

The most common family in the dataset is zbot, also known as Zeus Trojan. Despite its name, it actually combines multiple malware behaviors. Firstly, it makes the infected local machine part of a botnet, while also granting attackers access to the machine's data. Additionally, it installs a keylogger on the infected system. Some variants include a "web inject" component that adds malicious JavaScript code to a bank page, tricking users into leaking sensitive information (pro, 2024). There is also a variant called Game Over Zeus which in addition to the bank account stealing component it installs ransomware, and it is very possible that the zbot label returned by AVClass refers to this specific family.

Xorist is a ransomware family created using Encoder Builder. This tool allows cybercriminals to customize the ransomware by choosing the file encryption algorithm (XOR or Tiny Encryption Algorithm), the ransom-demand message, or the file types to be encrypted and can be considered one of the first steps of ransomware as a service (RaaS).

Reveton is a form of ransomware that differs from Xorist in the sense that it uses intimidation tactics to pressure the victims to pay the ransom (Kara and Aydos, 2022). It usually displays a notice that claims that the user has committed a crime, and can also hijack their webcams, making the victim believe they are being recorded by the police.

Onlinegames is a Trojan variant designed to steal confidential information from players of popular online games. It achieves this by reading the process memory of certain game executables or by accessing variables from the game's configuration files (fse, 2024).

Wapomi is commonly detected as a Worm or Trojan and infects machines by exploiting a Windows feature called "autorun", in which a program is automatically executed when a USB stick or removable driver is plugged to a machine (bad, 2024).

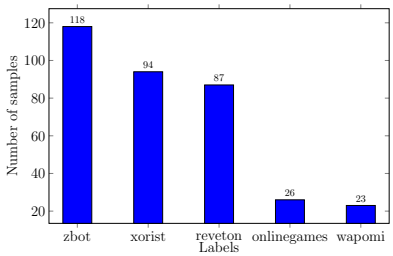

Figure 1: Database family distribution.

3.4 Static Extraction

The objective of the study at this stage was to extract as much information as possible from a given file. To achieve this, multiple tools that could cover fea-

tures from different areas of a file were needed. After thorough research, a final selection of four tools was made:

1. Pefile (pef, 2024) is a Python library used for parsing and analyzing portable executable files. This tool can extract information like PE headers, PE sections, and imported and exported symbols and can be considered the base of the four.

2. Flare-Floss (fla, 2024a) is designed to automatically extract and deobfuscate strings from malware binaries utilizing advanced static analysis techniques. It is similar to the traditional Linux "strings" command but additionally can handle obfuscation, an approach commonly used by ransomware authors to hide the true intentions and functionality of their program.

3. Exiftool (exi, 2024) is a command-line utility used to get meta information about a file. Unlike the other three tools, which specialize in portable executables, this one accepts files of any type. This command was used together with "-n" option to output numeric values without formatting, facilitating smoother parsing.

4. Dependencies (dep, 2024) is a modern and faster rewrite of Dependency Walker, available as open-source software. This tool focuses on the extraction of all Dynamic Link Libraries (DLLs) that a program depends on.

To optimize the static analysis process, a script was developed to concurrently execute four threads. For every file in a dictionary, each thread executes one of the tools specified, parses its output and saves the information to a global dictionary. Depending on the tool, the value can indicate various aspects such as the number of times the key appears in the program, as in the case of floss output, the presence, with a value of 1 or an actual integer or float value. Furthermore, the script generates a single CSV file with exactly one row for each sample, writing the keys of the dictionary as columns and the values are placed in the corresponding first row. This approach ensures that no information is lost for already analyzed samples in case of an error. Given that Floss and Dependencies tools may require longer computational time to correctly analyze a file, a timeout mechanism was implemented. The script will wait a maximum of four minutes for the Floss process to complete and eight minutes for Dependencies, meaning that samples that take longer will be dropped.

3.5 Dynamic Extraction

An approach that is straightforward, always available, and easy for users to install was desired for the dynamic analysis. For these reasons, Cuckoo Sandbox, a popular open-source automated malware analysis system that executes samples in a controlled environment, was excluded. Moreover, Cuckoo can be considered outdated nowadays, as it only supports Python 2 and Ubuntu 18.04. Subsequently, API approaches were researched because such a solution would meet the proposed requirements. Among the limited options, the two best candidates were selected: VirusTotal and Hybrid-Analysis, from which only one should be chosen. Both alternatives have similar functionalities, providing endpoints to submit a file and get its behavioral report. The API calls can be utilized in a script to extract the dynamic features of a sample.

VirusTotal is primarily known for aggregating multiple antivirus engines to scan a given file concurrently, with each engine determining whether it is malicious or safe. The documentation states that the submitted samples are automatically executed in a sandboxed environment with their behavior recorded. However, the dynamic report is available instantly after submitting a sample, which raises suspicions that the sample may not actually be run in a controller environment. Additionally, the reports show inconsistency in the information provided, with some samples executing in multiple sandboxes and thus offering more details while others executing in only one. No documentation was found regarding how VirusTotal decides which sandboxes to run the sample in, or how the dynamic report is available instantly. For these reasons, Hybrid-Analysis was chosen.

Hybrid-Analysis (hyb, 2024) offers a Falcon Sandbox public API with various endpoints, though a free account has restricted access to them. Permissions are granted for the essential ones, allowing the upload of files for analysis and fetching the report summary of a sample. Other endpoints, such as those retrieving the extracted binaries files or memory dumps, which would've provided additional information, could not be used. Another notable limitation is that the API only permits 100 daily file submissions, thus slowing down the analysis process.

Instead of submitting one sample at a time and waiting for the behavior analysis to be completed, a script was employed to submit 100 samples simultaneously, taking advantage of Hybrid-Analysis's ability to process submissions in parallel. The scrip makes API calls to the '/submit/file' endpoint with the following supplementary input parameters: **environment_id** was set to 160, specifying the operating system of the sandbox, in this case meaning Windows 10 64 bit; **experimental_anti_evasion** was set to true, this applies experimental techniques to prevent malware evasion tactics that detect sandbox environment and avoid execution; **script_logging** was

enabled to capture more details regarding any scripts run by the sample; **network_settings** was set to 'simulated' to simulate network traffic during the analysis; **input_sample_tampering** was set to true to allow manipulation of samples in a way that disrupts or reveals evasion attempts. After waiting about an hour for all the samples to complete successfully, another script was run to retrieve the summary reports, parse the JSON response, and store the output in a dictionary, similar to the approach used in static analysis. The JSON response does not provide popular dynamic analysis information such as API calls or registry activities. Instead, it offers details about network activity, file operations conducted by the sample, MITRE ATT&CK techniques present in the program's behavior, and signatures, which provide more specific behavioral features observed during analysis. Finally, a single CSV file per sample is generated, which only contains the extracted dynamic features.

At this point, all the necessary information about a sample was available: the malware family it belongs to, stored in a dictionary and two CSV files, one containing the static features and the other containing dynamic features. To generate the final database, a script that combines two CSV files into a single one was employed, basically merging the static and dynamic features for each sample. Then the add_label program was used to write a label column in all of the previously generated files, utilizing the information from the dictionary. Finally, a script to merge all the CSVs into one database was needed. It works by creating a set of all the unique column names from the files and then writing this set to the final database. For each sample, a row is created in the table and because it is possible for a column from the set to not appear in a file, a value of 0 is assigned to that column in the corresponding sample row.

3.6 Feature Selection

The final database consists of approximately 1.3 million static features and around 6,000 dynamic features. This discrepancy comes from the number of tools used and their methodology. For instance, Hybrid-Analysis follows a consistent algorithm, while tools like Floss extract all the strings from a sample and as a result, even if only one entry has a unique string, it will be counted as a new static feature. The aim of this step was to reduce computational cost while also maintaining the accuracy of the classification model. Given the large number of attributes in the dataset, WEKA (wek, 2024) was used to identify the most valuable features and drop the redundant ones. WEKA is a collection of machine learning and data mining algorithms that can be used through a graphical user interface. As seen in (Aurangzeb, 2018) the InfoGainAttributeEval method from WEKA was used for the attribute selection. This function evaluates the worth of an attribute by measuring the information gained with respect to the class. The information gain of an attribute A with respect to a class C is calculated as:

$$\text{InfoGain}(C, A) = H(C) - H(C|A) \quad (1)$$

where H is the entropy function. In this case, the class is the label column from the database.

An issue encountered when using WEKA was the fact that special characters were not supported in the column name of the database. To address this problem, a program was developed that deletes all the restricted characters from these names. Additionally, if duplicate column names are generated as a result, the program keeps only the first occurrence of each column in the database. Unfortunately, after this process, only around 800,000 total features remained. Another obstacle faced was not having enough heap memory when loading the dataset in WEKA. After adjusting some settings and using nearly all of the system's RAM, the database eventually loaded successfully. However, the same error occurred when attempting to perform InfoGainAttributeEval, and since no more system resources were available, the decision to split the database in two batches was made. The optimal approach would have involved applying the information gain method on the entire dataset. This would have led to a more accurate identification of the most informative features, ensuring maximum discriminatory power.

For splitting the data in the two batches the final CSV files used to create the initial database were utilized. Each ransomware family's samples were evenly split between the batches, except for the ones having just one sample, where the assignment was randomly determined. Then the merge_csv script was used to create the two databases on which the WEKA's InfoGainAttributeEval method was applied. The algorithm yielded 523 features with a score above zero from the first batch and 632 from the second batch. Only these features were considered because having a score greater than zero means they provide useful information to some degree. Before combining the features, both groups of scores were normalized to fit within the [0,1] range, ensuring easier comparability. The two dictionaries containing top features and score pairs can have common keys, so when combining them into the final output, the higher score for each common feature was considered. The final dictionary contains 783 features and the best 20

ranked ones are shown in Figure 2. In total 233 features were produced by dynamic analysis, while static analysis yielded 550 features. An interesting observation is that none of the 783 selected features originate from the Dependencies tool, being the only component used that did not contribute with any useful information.

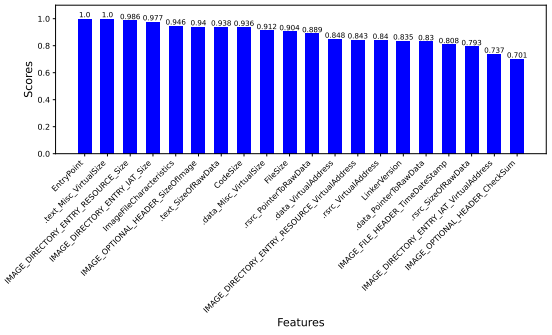

Figure 2: Best ranked features by information gain method.

3.7 AI Model

The final dataset used to train the model contains only the best features found in the previous step. By selecting only the most useful attributes, the database was reduced from 1 GB to 1 MB, thus decreasing the computational cost while at the same time improving the accuracy of the classification model. The scope of this paper was not to find the best-supervised learning algorithm for classifying ransomware/malware because several studies cover this aspect (Aurangzeb, 2018) (Yoo et al., 2021) (Poudyal et al., 2018) (Singh and Singh, 2022). These papers also show that the best overall performing algorithm in this context is Random Forest, so this was the selected choice. Random Forest is a classifier that uses multiple decision trees, each one having a random subset of data and features. This randomness brings variability to the individual trees, reducing the risk of overfitting and improving overall prediction performance. In the final prediction, the algorithm uses a voting mechanism to aggregate the results of all the trees. The RandomForestClassifier from the 'sklearn.ensemble' python library was used to create the model. Default parameters were utilized, besides n_estimators, which represents the number of decision trees in the forest. After conducting manual testing, it was observed that the best accuracy was returned when setting the number of decision trees to 86. Finally, 80% of the data was allocated for training the model and the rest for testing.

3.8 RansoGuard

Following the work described so far, RansoGuard, a Windows desktop application, was created. The app implementation and the scripts discussed can be found at (ran, 2024). For an easy integration of the Random Forest model in the application, PyQt5 was used for its development. While other frameworks can integrate Python machine learning models, doing so typically involves more complexity and might require additional wrappers. The software offers a user-friendly graphical user interface (GUI) through which users can upload one file at a time. The file must be a portable executable since the tools used to extract its features have this requirement. Following the upload, three static tools and one dynamic tool ran in parallel and extracted the 783 selected features identified as the most valuable. If one fails to execute successfully on the uploaded file, any already running processes are allowed to finish operating. However, the application will not proceed to the next step and instead will notify the user of the error and its originating source.

Initially, static analysis employed four tools, but "Dependencies" was excluded since it doesn't contribute with any useful information. The static tools used are: Floss for extracting the strings from a file, Exiftool for obtaining metadata, and the pefile Python library for retrieving PE-specific information. Dynamic analysis is performed using the Hybrid-Analysis API. The application initiates a request to submit the file for examination in a sandbox environment and then waits for completion before making a final API call to retrieve a summary report. From this report details about network activity, MITRE ATT&CK techniques present in the program, and signatures that indicate specific behavioral characteristics are extracted. With a Hybrid-Analysis free account, users are limited to a maximum of 100 sandbox submissions per day, meaning they can utilize the application for up to 100 files daily. Once both the static and dynamic analysis are completed, the Random Forest machine learning model predicts in what ransomware family the file belongs to based on the extracted features.

Finally, a report window appears on the user's screen, allowing them to upload a file again, while at the same time having the option to review a generated report containing the model's prediction and the 783 features with the extracted values. The application offers a history tab, where users can see a list of their previously analyzed file names and the family prediction. Clicking on an item in the list opens the report window for that file, allowing users to revise the information. Additionally, they can visit the help

tab for an explanation of the application's functionality.

4 RESULTS

The aim of this chapter is to evaluate the Random Forest model. The dataset used for both testing and training contains 707 samples from 99 different malware families and for each instance, 783 features were extracted signifying the most useful attributes as explained in 3. The goal was to achieve an evenly distributed dataset with a maximum of 120 entries per family. However, only an upper limit was imposed, resulting in a database with many families relative to the number of samples, some containing just a few instances. This will potentially affect the model by lowering its accuracy, especially for underpopulated families, and by making it prone to bias towards the majority classes.

To evaluate the performance of the Random Forest model on the obtained database, the following metrics were used:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}} \tag{2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

$$\text{F1 score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

The model accuracy is 71.83% and the other metrics presented were calculated using a weighted approach. In this way, each class' contribution to the overall metric is proportional to the number of true instances of that label in the dataset. This was preferred over a normal average because the database is imbalanced. The model demonstrates an overall precision of 0.79, recall of 0.72, and f1 score of 0.66. Table 1 shows the metrics for the five most populated families in the database. In this context, support represents the number of instances of each class present in the testing data. It can be observed that for these five families, recall is higher than precision, whereas overall the opposite is true. This is probably due to the high difference between the samples in certain classes. The reduced precision says that the algorithm often predicts these five families for instance belongings to something else, showing that the model is biased towards the majority classes.

Additionally, the data presented in Table 2 illustrates the metrics for five randomly selected classes,

Table 1: Metrics for top 5 most populated malware families.

Class	Precision	Recall	F1-Score	Support
zbot	0.69	0.83	0.75	29
xorist	0.84	0.94	0.89	17
reveton	0.64	0.93	0.76	15
onlinegames	0.57	0.80	0.67	5
wapomi	0.50	1.00	0.67	2

since the database contains 99 families it is not feasible to include all of them. It was noticed that the same trend persists, classes having recall higher than precision. Furthermore, it was observed that multiple minority labels like msil have a maximum precision and minimal recall. This indicates that these classes were never predicted by the model and would explain the trend present in the more populated families.

Table 2: Metrics for 5 randomly selected ransomware families.

Class	Precision	Recall	F1-Score	Support
dalexis	0.80	1.00	0.89	4
poison	0.57	0.67	0.62	6
urausy	0.80	1.00	0.89	4
winwebsec	0.50	0.33	0.40	3
msil	1.00	0.00	0.00	2

The confusion matrix is presented as the final evaluation method for the model. This matrix was created including only the ten families for which the metrics were provided earlier. This evaluation technique shows how many instances were correctly classified and, for those that were misclassified, reveals the family they were predicted as. For example, in Figure 3 it can be seen that four zbot instances were wrongly identified as xorist and that the two msil samples were incorrectly predicted as poison. The main diagonal of the confusion matrix represents the number of correctly classified instanced for that family.

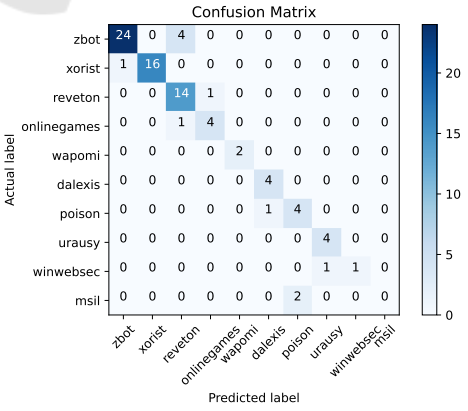

Figure 3: Database family distribution.

5 CONCLUSION & FUTURE WORK

This article focuses on analyzing Windows portable executable files to identify key features that help classify samples into ransomware families. We have extracted extensive data using four static analysis tools and the Hybrid-Analysis API for behavioral analysis in a sandbox environment. We have examined 707 samples with a hybrid approach and used Weka to manage the high-dimensionality of the feature vector, and selecting 783 useful attributes to train a Random Forest classification model. The resulting application, RansoGuard, extracts these features from files and predicts ransomware families, generating a report on the predictions and feature values. The model demonstrated promising results, achieving an accuracy of 71.83%, precision of 0.79, and recall of 0.72.

The research faced several limitations. It was hard to find a good database of ransomware samples due to a lack of options, and the chosen dataset, initially claimed to contain only crypto-ransomware, turned out to be mislabeled. Although AVClass was used for labeling, its accuracy was questionable. Additionally, the APIs (VirusTotal and Hybrid-Analysis) imposed submission rate limits that slowed down the analysis. Besides these, we also had to split the data into two batches because Weka required substantial RAM due to the large number of attributes.

For future work, we plan to include benign programs in the dataset, thus making the model able to distinguish between malicious and non-malicious software. There is also the intention to validate the framework on a larger, more evenly distributed database. In addition, we can also use ransomware samples from different operating systems such as Linux, macOS, and Android.

REFERENCES

(2024). AVClass. https://github.com/malicialab/avclass. [Online; accessed 05-April-2024].

(2024). Dependencies. https://github.com/lucasg/Dependencies. [Online; accessed 05-April-2024].

(2024). ExifTool. https://exiftool.org/. [Online; accessed 05-April-2024].

(2024a). Flare-Floss. https://github.com/mandiant/flare-floss. [Online; accessed 05-April-2024].

(2024b). Flare VM. https://github.com/mandiant/flare-vm. [Online; accessed 03-April-2024].

(2024). Hybrid-Analysis. https://hybrid-analysis.com/docs/api/v2. [Online; accessed 24-April-2024].

(2024a). Market share held by the leading computer operating systems. https://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems-since-2009/. [Online; accessed 05-April-2024].

(2024). Pefile. https://github.com/erocarrera/pefile. [Online; accessed 05-April-2024].

(2024). RansoGuard. https://github.com/raul-dunca/ransoguard. [Online; accessed 05-April-2024].

(2024b). Total annual amount of money received by ransomware actors worldwide from 2017 to 2022. https://www.statista.com/statistics/1410498/ransomware-revenue-annual/. [Online; accessed 02-April-2024].

(2024). Trojan-PSW:W32/OnlineGames. https://www.f-secure.com/v-descs/trojan-psw-w32-onlinegames.shtml. [Online; accessed 04-June-2024].

(2024a). VirusShare. https://virusshare.com/. [Online; accessed 29-March-2024].

(2024b). VirusTotal. https://www.virustotal.com/. [Online; accessed 05-April-2024].

(2024). Wapomi. https://docs.badrap.io/types/malware-wapomi.html#malware-wapomi. [Online; accessed 04-June-2024].

(2024). Weka. https://waikato.github.io/weka-site/index.html. [Online; accessed 15-April-2024].

(2024). What Is Zeus Trojan (Zbot)? https://www.proofpoint.com/us/threat-reference/zeus-trojan-zbot. [Online; accessed 03-June-2024].

Abbasi, M. S. (2023). Automating Behavior-based Ransomware Analysis, Detection, and Classification Using Machine Learning.

Aslan, Ö. A. and Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE access*, 8:6249–6271.

Aurangzeb, S. (2018). A machine learning based hybrid approach to classify and detect windows ransomware. *MS (CS) dissertation, Capital Univ. Sci. Technol., Islamabad, Pakistan*.

Kara, I. and Aydos, M. (2022). The rise of ransomware: Forensic analysis for windows based ransomware attacks. *Expert Systems with Applications*, 190:116198.

Poudyal, S., Subedi, K. P., and Dasgupta, D. (2018). A framework for analyzing ransomware using machine learning. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1692–1699.

Rizvi, S. K. J., Aslam, W., Shahzad, M., Saleem, S., and Fraz, M. M. (2022). Proud-mal: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable. *Complex & Intelligent Systems*, pages 1–13.

Singh, J. and Singh, J. (2022). Assessment of supervised machine learning algorithms using dynamic api calls for malware detection. *International Journal of Computers and Applications*, 44(3):270–277.

Yoo, S., Kim, S., Kim, S., and Kang, B. B. (2021). Ai-hydra: Advanced hybrid approach using random forest and deep learning for malware classification. *Information Sciences*, 546:420–435.