Ransomware Evolution: Unveiling Patterns Using HDBSCAN*

Prajna Bhandary[1,*], Robert J. Joyce[1] and Charles Nicholas[1]

[1]University of Maryland, Baltimore County, 1000 Hilltop Cir,Maryland 21250

Abstract

This research presents an innovative approach to enhancing ransomware detection by leveraging Windows API calls and PE header information to develop precise signatures capable of identifying ransomware families. Our methodology introduces a novel application of hierarchical clustering using the HDBSCAN algorithm, in conjunction with the Jaccard similarity metric, to cluster ransomware into discrete families and generate corresponding signatures. This technique, to our knowledge, marks a pioneering effort in applying hierarchical density-based clustering to over 1.1 million malicious samples, specifically focusing on ransomware and using the clusters to automatically generate signatures.

We show that identifying unique Windows API function patterns within these clusters enables the differentiation and characterization of various ransomware families. Furthermore, we conducted a case study focusing on the distinctive function combinations within prominent ransomware families such as GandCrab, WannaCry, Cerber, Gotango, and CryptXXX, unveiling unique behaviors and API function usage patterns. Our scalable implementation demonstrates the ability to efficiently cluster large volumes of malicious files and automatically generate robust, actionable function signatures for each. Validation of these signatures on an independent malware dataset yielded a precision rate of 98.34% and specificity rate of 99.72%, affirming their effectiveness in detecting known ransomware families with minimal error. These findings underscore the potential of our methodology in bolstering cybersecurity defenses against the evolving landscape of ransomware threats.

Keywords

Ransomware, HDBSCAN, API call

1. Introduction

Ransomware has emerged as a formidable and destructive threat to individuals, organizations, and even governments worldwide. Designed to block access to files or a computer system until a sum of money is paid, this type of malicious software has continued to grow in both sophistication and impact, resulting in significant disruptions and financial losses. As of 2023, over 72% of businesses worldwide have been affected by ransomware attacks [1]. A Palo Alto Networks study found a 50% surge in ransomware attack announcements on leak sites in 2023, indicating a significant rise in ransomware incidents and underscoring the continually-changing ransomware threat landscape [2].

The dynamic nature of ransomware calls for additional research and analysis to understand their evolving patterns and to develop effective countermeasures. Ransomware actors continually update their malware to add new functionality and to evade detection by antivirus and other endpoint security products. New families of ransomware emerge frequently, and it is not uncommon for existing ransomware families to be “rebranded". Furthermore, it’s crucial to recognize that ransomware affiliates may change their tactics, switch between families, or even manage several families concurrently. This is done to avoid scrutiny and potentially target the same victims again under different guises [3]. Because of these factors, the naming of ransomware families and the tracking of ransomware campaigns over time is challenging and often a source of confusion in reporting.

To identify individual families of ransomware, the traditional approach has been antivirus signature detection. Antivirus companies deploy signatures that search for unique patterns (typically byte sequences) which occur within files belonging to a particular malware family [4]. Although they are

CAMLIS’24: Conference on Applied Machine Learning for Information Security, October 24–25, 2024, Arlington, VA

*Corresponding author.

prajnab1@umbc.edu (P. Bhandary); joyce8@umbc.edu (R. J. Joyce); nicholas@umbc.edu (C. Nicholas)

effective against known threats, antivirus signatures cannot accurately detect novel types of malware, and they may fail to detect older malware families which utilize common evasion techniques (e.g. polymorphism and packing). In these instances, antivirus products may fall back to heuristics and machine learning detection methods, but these methods cannot always reliably identify ransomware families.

The core of our analysis revolves around the examination of Windows API functions, which serve as indicators of ransomware behavior and intent. We observe that files within a given ransomware family tend to import a common set of functions which can uniquely identify files belonging to that family. Our work proposes a strategy for automatic ransomware signature generation which utilizes unique combinations of imported functions rather than traditional byte patterns. We note that, unlike traditional byte pattern signatures, our function signatures are robust against polymorphic evasion techniques since they are unaffected by code alterations.

Our methodology automatically identifies groups of malware belonging to the same family using the HDBSCAN hierarchical clustering algorithm, complemented by a custom distance metric based on function imports and PE metadata similarity[5]. We re-implemented large portions of an existing HDBSCAN library to support sparse matrices, allowing us to cluster over 600,000 ransomware files[6, 7].

This methodology enables the granular differentiation of ransomware families, laying the groundwork for the creation of precise and actionable signatures based on unique sets of imported functions. Our experiments show that our function signatures can accurately detect ransomware and distinguish between different ransomware families.

2. Background

We briefly review the terminology we use in this paper, including ransomware categories, Windows API functions, and PE header metadata.

2.1. Categories of Ransomware

Diverse ransomware families, each with unique characteristics and attack methodologies, have proliferated across the digital landscape. Understanding the distinctions among these families is crucial, as it enables cybersecurity professionals to develop targeted defense mechanisms. Familiarity with the operational nuances of various ransomware strains, such as their encryption tactics, propagation methods, and communication with command and control servers, provides essential insights necessary for effective threat detection and mitigation.

Ransomware can be broadly classified into several key types [8], [9], including File-encrypting ransomware (e.g. Cryptolocker, Wannacry, and Locky), which encrypt valuable files demanding a ransom for decryption keys; Locker ransomware (e.g. earlier versions of Petya and Satana), which deny user access to the infected device; Scareware (e.g. Reveton), which feign authority to extort victims; and Doxware, (e.g. Jigsaw), which threaten to leak sensitive information. Additionally, Ransomware as a Service (RaaS) platforms like GandCrab are examples of commercialization of these attacks, enabling a broader range of actors to deploy ransomware. Understanding these types aids in devising targeted defense strategies against the multifaceted ransomware threat landscape.

2.2. The Windows API

The Windows Application Programming Interface (API), also known as WinAPI, is Microsoft's core set of API functionality on Windows. WinAPI consists of numerous functions that allow developers to handle many low-level tasks, such as creating and managing windows and dialogues, processing user input, managing files and directories, and controlling various aspects of the system's operation. These functions are grouped into different libraries, such as USER32.DLL for user interface operations and KERNEL32.DLL for system operations. [10]

Table 1
Comparison of previous research works

Reference	# of malware	Method	Ransomware focused
Alazab et al. (2010) [11]	66,703	Static Analysis	-
Walker et al. (2023) [12]	4,533	Dynamic Analysis	-
Daeef et al. (2022) [13]	7,107	Dynamic Analysis	-
Mowri et al. (2022) [14]	1,550	Dynamic Analysis	✓
Anand et al. (2022) [15]	653	Dynamic Analysis	✓
Our work	627,298	Static Analysis	✓

The Windows API is important in malware analysis because it offers deep insights into how malware interacts with the Windows operating system. Malware utilizes these APIs to perform a variety of operations such as procuring system information, executing processes, and manipulating files or registry entries. By leveraging specific API calls, malware can maintain persistence, propagate, and evade detection, all while conducting its intended activities. Analysing API calls used by malware provides critical clues about its functionality and behaviour, helping malware analysts understand the malware's objectives, measure its impact on the infected system, and identify strategies for mitigation or removal.

2.3. The PE Header

The Portable Executable (PE) header plays a crucial role in malware analysis as it contains essential information about the executable file's structure and behavior. This header, specific to executable files for the Windows operating system, includes metadata about a file, such as its size, architecture, and the locations of its code and data segments. Malware analysts scrutinize the PE header to identify suspicious characteristics that might indicate malicious intent. For instance, anomalies in the section names, sizes, or the presence of unusual attributes can signal potential malware. Additionally, the PE header includes information about the file's dependencies e.g. the libraries it requires, which can reveal attempts to exploit particular vulnerabilities or perform malicious activities. By analyzing the PE header, experts can gain insight into the malware's functionality, origin, and potential impact, aiding in the development of effective detection and mitigation strategies.

3. Related Work

In the rapidly evolving field of malware analysis, leveraging API call sequences to understand and classify malware behavior has become increasingly prevalent. However, despite the advancements, there remains a gap in the application of unsupervised clustering methodologies to these sequences, especially on the scale of our dataset, which exceeds one million malicious files. Several studies have specifically focused on ransomware detection using API calls similar to our study; however, none have addressed it with the dataset size comparable to ours. Mowri et al. (2022) [14] harnessed supervised machine learning to achieve a high accuracy rate in classifying Crypto and Locker types of ransomware by extracting potential features through dynamic analysis. While their reported accuracy of 99.15% is impressive, the scope of their dataset, despite being larger than some, still did not approach the size of ours, potentially limiting the generalizability of their findings. Similarly, Anand et al. (2022) [15], focused on identifying key API calls through feature importance methods to build robust classification models. Their study employed both dynamic and static analysis techniques, highlighting the significance of API call analysis in detecting ransomware, yet their dataset was considerably smaller.

Broader malware analysis work using API calls includes Alazab et al. (2010) [11], which laid the groundwork in this domain by analyzing malware behavior through API call sequences. Despite its contributions, this study did not leverage machine learning techniques and was limited by a relatively small dataset. Walker et al. (2023) [12], performed dynamic analysis on malware samples and employed

cosine similarity to discern variants within malware families, addressing the issue of duplicate API calls. Nevertheless, these approaches did not incorporate unsupervised clustering, which might unveil more nuanced insights from the data. Furthermore, the work of Daeef et al. (2022) [13] highlights the critical role of API calls in malware classification, employing Jaccard similarity and visualization to extract meaningful patterns from API sequences. Their API sequences were extracted by executing malware in a sandbox. While this approach enhanced classification accuracy, it still did not leverage the potential of unsupervised learning nor did it analyze data at the scale we have.

Research has also explored the use of the HDBSCAN clustering technique. For instance, Azteni et al. (2018) [16] used HDBSCAN to cluster Android malware, utilizing features from APK file metadata and dynamic analysis. In similar fashion, Rahat et al. (2024) employed HDBSCAN to cluster IoT malware by analyzing static features, such as opcode sequences and control flow graphs, to effectively identify and categorize various malware families. In contrast, our research approach relies on static analysis alone, for the sake of scalability.

Expanding our horizons beyond API call clustering, research by Raff et al. (2020) [17] on clustering techniques for YARA rule generation emphasizes the versatility of unsupervised learning in cybersecurity applications. Their findings underscore the potential of such methods to improve threat detection mechanisms.

In Table 1, we summarize the previous works related to API call analysis and the dataset sizes used. Our work is distinguished by its scale and focus on ransomware. To our knowledge, no prior work has applied hierarchical density-based clustering to a dataset of ransomware of over 1.1 million malicious files. By integrating diverse clustering methodologies, our research aims to bridge the gap in the current literature, leveraging the scalability and robustness of unsupervised clustering techniques to analyze large volumes of malware data. This approach not only enhances the detection and classification of ransomware families but also provides actionable insights through the generation of robust function signatures.

4. Methodology

This section presents a detailed account of the methodology used to automatically generate signatures for ransomware. After extracting Windows API functions and PE metadata from a large corpus of malware, our methodology follows three major steps, as shown in Figure 1.

First, distances between pairs of files are measured as a function of the Jaccard similarity between imported functions and PE metadata. We hypothesize that PE metadata, although subject to spurious correlations, combined with imported API functions, provides a broader context of the malware's behavior and structure. Specifically, API calls are critical since ransomware, like any other malware, ultimately relies on these calls to execute its malicious activities. Despite the potential for evasion through dead code or runtime loading, our approach seeks to capture the core functionalities that ransomware cannot avoid.

Second, these distances are stored in a sparse matrix and clustered using HDBSCAN. Given the large size of our dataset, with over 1.1 million malicious files, we re-implemented significant portions of the HDBSCAN algorithm to support a sparse data format. This re-implementation allows us to handle the high dimensionality and sparsity of our data more efficiently than the original dense format.

Finally, a YARA rule is generated for each large cluster identified by HDBSCAN. The generated signatures match ransomware files which share a unique combination of imported functions. The YARA rule generation process is crucial for practical application, as it translates the clustering results into actionable detection rules. For each cluster, we identify common API functions that appear in a significant portion of the samples, specifically focusing on those functions that are not overly generic but rather indicative of ransomware behavior.

To clarify, the YARA rules include both "common" and "rare" functions within the cluster. Common functions are those present in more than 5% of the samples, serving as the core indicators for the rule. Conversely, functions imported by 5% or fewer of the malicious ransomware samples are considered to

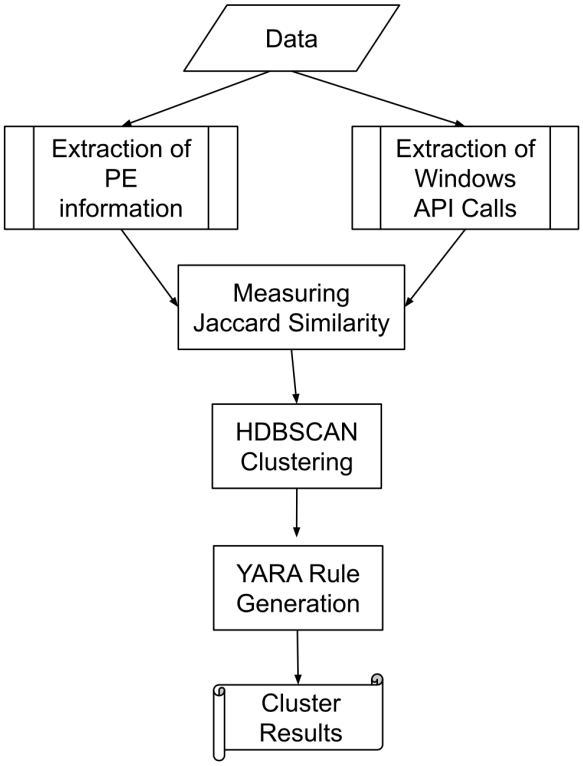

Figure 1: Workflow diagram

be “rare". We later show that this threshold, although initially an ad-hoc selection, permits an acceptable balance between detection precision and recall.

By integrating these steps, our methodology aims to leverage the robustness of unsupervised clustering techniques to analyze and categorize an unprecedented volume of malware data, ultimately enhancing our ability to detect and classify ransomware.

4.1. Dataset Preparation

We are using a subset of the Sophos-ReversingLabs 20 Million dataset (SoReL-20M) for this research [18]. SoReL-20M is a large-scale dataset consisting of pre-extracted features, metadata, and labels for approximately 20 million malicious files. The SoReL dataset is roughly balanced between malicious and benign files, and the malicious files are tagged according to 11 different behaviors. We selected the 1,152,354 malicious files with the “ransomware" tag from SoReL-20M for use in the following experiments. In addition, we queried the VirusTotal API to obtain antivirus scan results for each of these malicious files. These API queries were made in April 2022. We then ran AVClass on these VirusTotal scans to obtain malware family names for many of the files [19].

4.1.1. Extracting Unique Windows API Function Names

We began our methodology with the careful construction and cleaning of our training dataset.

SoReL-20M includes lists of functions imported by each file, and the name of the DLL from which each imported function originates. We identified 15,918 unique functions among the 1,152,354 ransomware files in SoReL-20M. Functions that correspond to unrelated libraries (not in the Windows API), as well as functions which were unidentifiable, too short, or had nonsensical names, were discarded. We then assigned a numeric value to each remaining function. After data cleaning, we were left with 3,464 unique Windows API functions.

4.1.2. Extracting PE Header Information

The raw metadata in SoReL-20M also contains parsed PE header metadata. We selected the following PE header fields: “Machine", “Characteristics", “MajorLinkerVersion", “MinorLinkerVersion", “MajorOperatingSystemVersion" , “MinorOperatingSystemVersion", “MajorImageVersion", “MinorImageVersion", “MajorSubsystemVersion" and “MinorSubsystemVersion". These PE header fields describe broad properties about the file, such as the minimum version of Windows it runs on, the version of the linker, and flags regarding whether debug, symbol, and relocation information was stripped from the file. These values were selected because they are less likely to be changed by minor alterations (e.g. fields representing offsets and sizes) or by packing (e.g. PE section fields).

4.1.3. Dataset Statistics

After extracting PE header metadata and imported functions from the 1,152,354 ransomware samples in SoReL, we discarded files which could not be used for our experiments. These included files for which no PE metadata could be extracted, files with fewer than 10 imported functions, and files which were not available on VirusTotal. After this process, our dataset for generating ransomware signatures contained 627,298 ransomware files.

4.2. Measuring Jaccard Similarity Between Files

During the next stage of our methodology, we computed the distance between pairs of potentially-similar files. The distance metric we developed is based on Jaccard similarity. Let A and B be two sets. Then, the Jaccard similarity of A and B is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Let M_i be the set of imports for file i and let P_i be the set of selected PE metadata fields for file i. Then, define $dist(i,j)$ as:

$$dist(i,j) = 1 - \frac{J(M_i,M_j)+J(P_i,P_j)}{2} + \epsilon$$

More informally, $dist(i,j)$ is one minus the average of the Jaccard similarity between the sets of imported functions and the Jaccard similarity between the sets of PE metadata, plus a small ϵ term. This ϵ term is necessary for our implementation, where non-sparse elements must be non-zero and positive. Our custom distance metric requires overlap in both the imported functions and the selected PE metadata fields in order for a pair of files to be considered similar.

Let X be a distance matrix, where $X_{ij} \in X = dist(i,j)$. Due to the size of our ransomware dataset, a dense matrix of shape 627,298 x 627,298 could not be stored in memory. For the same reason, it would also not be practical to compute the distances between every pair of files. Because of this, our approach is selective in choosing pairs of files for which we compute distances. From our set of 3,464 observed Windows API functions, we identified “rare functions" which were imported by fewer than 5% of the ransomware samples. Of these, we only computed the distance between pairs of files which shared at least one of these “rare functions". This serves two purposes: it significantly reduces the number of comparisons and it ensures that resulting signatures are not simply combinations of common Windows API functions.

We used Compressed Sparse Row (CSR) matrices in our implementation for storing the distance matrix X in a sparse format.

As a dense matrix, X would have required ≈1.4TB of memory with a 32-bit floating point representation, but our sparse implementation required just 5.24 MB - more than a 250,000× savings. For scalability purposes, our implementation supports parallelized distance computations. Using 32 cores on an AMD Threadripper 3970X CPU, we computed X in 21.64 minutes.

4.3. Application of HDBSCAN

Once distances for all eligible pairs of files have been computed, the sparse distance matrix X has encoded a network of the functional and structural overlaps between a massive collection of ransomware files. We observe dense, inter-connected regions within this network where all files are members of the same families. Our approach uses the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) to extract these dense regions as clusters [20, 6].

4.4. Adapting HDBSCAN for Sparse Matrices

Unfortunately, the Python hdbscan library only supports dense matrices. It was necessary for us to re-implement major portions of HDBSCAN ourselves in order to cluster the ransomware data. Below, we describe our adaptations to the HDBSCAN algorithm.

4.4.1. HDBSCAN Mutual Reachability Distance

The initial step of HDBSCAN involves determining the "core distance" $core_k(i)$ for each data point, given by the distance to the k^{th} nearest point. HDBSCAN uses the core distance to approximate the density of a region around a given data point. We selected $k = 2$ in our approach.

Once the core distances are established, our implementation calculates a second metric, the mutual reachability distance, between each pair of points i, j for which $X_{i,j} \neq 0$. The mutual reachability distance $reach(i, j)$ is defined as follows in [6]:

$$reach(i, j) = max\{core_k(i), core_k(j), dist(i, j)\}$$

Mutual reachability in HDBSCAN considers the core distances of both points and the actual distance between them, artificially increasing the distances between points that are not in dense regions. These mutual reachability distances are then efficiently stored in a second sparse matrix Z.

4.4.2. HDBSCAN Robust Single Linkage

Each mutual reachability distance $Z_{ij} \in Z$ can be thought of as a weighted edge in a graph between nodes i and j. However, Z may not represent a fully-connected graph. Our implementation identifies the connected components of Z and computes the minimum spanning tree (MST) of each connected component. Each MST yields a backbone of connectivity based on the shortest possible mutual reachability distances between points in the connected component.

The edges of each MST are then sorted from highest to lowest distance. This sorting is preparatory for the next step, where a single linkage tree is derived from the MST. The single linkage tree represents hierarchical clustering, where clusters at one level are joined to form clusters at the next level, based on the closest mutual reachability distance.

Finally, our implementation uses the Python hdbscan library to condense the single linkage tree, which involves trimming branches that do not contribute to cluster stability. Cluster stability is an assessment of the strength and persistence of clusters over different scales of distance. This condensed tree represents the final output of HDBSCAN, delineating stable clusters that are robust across different data densities and separations.

4.4.3. Hyperparameter Selection

A series of experiments were conducted to empirically select key hyperparameters, namely the minimum number of samples per cluster (min_samples) and the 'k' value for core distance calculations. These experiments were important in identifying the optimal settings that yielded the most coherent and interpretable clustering results.

```
import pe

rule cluster_10 {
  meta:
    hash1 = "2f746653089765de4158b6dbda..."
    hash2 = "a2c12d3f0af37c9a5769dcb681..."
    hash3 = "3a1af65e3362d6371590099a5b..."
    ...
  condition:
    pe.imports("version.dll",
    "GetFileVersionInfoSizeA") and
    pe.imports("version.dll",
    "GetFileVersionInfoA") and
    pe.imports("version.dll",
    "VerQueryValueA") and
    ...
}
```

Listing 1: Example generated YARA rule

4.5. YARA Rule Signature Generation

The final stage of our methodology is the generation of function signatures for each cluster which contains at least 10 files. We do this using YARA rules. Although YARA is typically used for matching byte patterns in files, YARA rules can be written to match files based on solely their imported functions using the YARA PE module.

The process of generating a function signature begins by analyzing each cluster's aggregated data and identifying a core set of Windows API functions shared by each of the files in the cluster. In order to generate a rule, all files in the cluster must import at least six common functions, with one of those being a "rare function" that is imported by no more than 5% of the ransomware samples in our dataset. Then, our implementation formats each function that is common to all files in the cluster (and the name of the DLL from which the function was imported) into the "condition" section of a YARA rule. Metadata, such as the hashes of each file in the cluster, is formatted into the "meta" section of the rule. An example generated function signature is shown in Listing 1.

5. Experimental Analysis and Results

In this section, we present a detailed analysis of our methodology's cluster formation and signature generation. In addition, we discuss the insights derived from testing our generated signatures on further ransomware data.

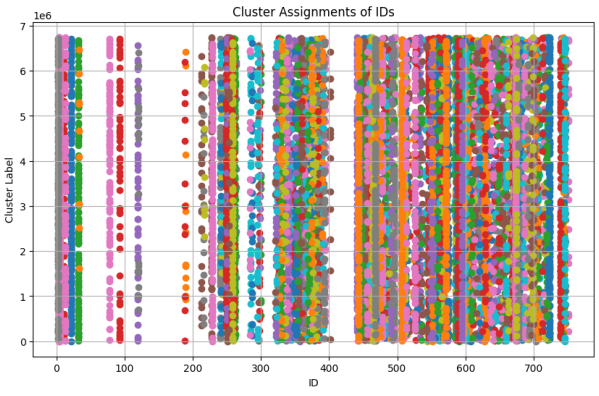

Figure 2: Distribution of all samples with their cluster.

5.1. Cluster Validation

Clustering our dataset of 627,298 ransomware files yielded a total of 751 clusters with size 10 or greater. Figure 2 shows the cluster to which each sample belongs. We manually investigated each of the 751 clusters. Of these, 386 clusters predominantly featured various types of known ransomware. The remaining clusters either had incor-

rect SoReL-20M tags (and after further investigation we found that these clusters instead were better categorized as RATs, PUPs, trojans, file infectors, or worms) or were groups of generic ransomware without a known family name.

We accepted the most common AVClass label as the family name of the cluster, and Figure 3 shows the distribution of clusters with known ransomware families [19]. Most of the clusters we identified were uniformly labeled by AVClass or had only minor variance in family labeling. In some cases, these were due to false positives in our methodology, and in others they were due erroneous AVClass outputs.

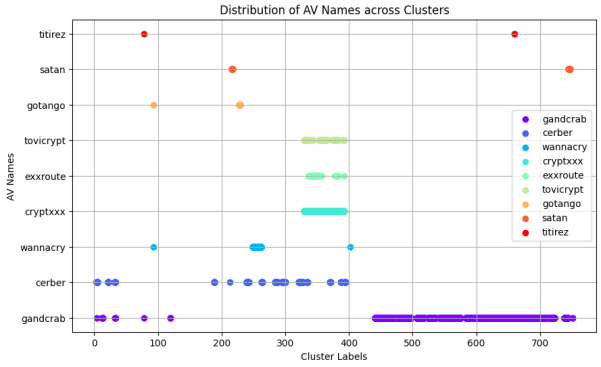

Figure 3: Distribution of ransomware and the cluster they belong.

We note that approximately 267 clusters among the clusters featuring ransomware were identified as containing solely Gandcrab ransomware, and these clusters tended to be larger in size, as shown in Figure 3. Furthermore, our study detected clusters containing CryptXXX, Exxroute, and Tovicrypt, which appear to be variants of the same malware. Besides these, we also identified other frequently-occurring ransomware families including Cerber, Wannacry, Satan, Titirez, and Gotango.

5.2. Discussion on Ransomware Family Clusters

Let us discuss the ransomware families we identified in more detail. The fundamental behavior of ransomware remains consistent: once it gains access to a system, it encrypts the system or files and demands a ransom. However, our experiments have revealed that each ransomware variant employs distinct sets of Windows API functions to infiltrate and encrypt files.

5.2.1. Ransomware Function Groupings

Considering the approach of Alazab et al, [11] to group API functions into distinct behaviors, we organized the frequently-occurring Windows API function calls identified across all ransomware clusters into five function types. Each group of functions is related to a specific behavior common to most ransomware. We describe each observed set of behaviors below, and more details are available in Table 4 in Appendix A.

- Function Type 1: **System and Process** Ransomware uses these functions to manipulate system resources and processes, allowing it to control system behavior and execute malicious actions discreetly.
- Function Type 2: **File and Library Interaction** This category of functions enable ransomware to interact with files and to dynamically load functions from external libraries, facilitating tasks such as file encryption and enabling access to functionality not listed in the import address table.
- Function Type 3: **GUI and Window Functions** Ransomware employ these functions to display ransom notes and manipulate window behavior to pressure users into paying the ransom.
- Function Type 4: **Shell and COM Operations** Ransomware can leverage these functions to execute system commands, access system resources, and exploit vulnerabilities through interactions with COM objects, enabling deeper infiltration and control of the system.
- Function Type 5: **Utility Functions** This category provides ransomware with tools for memory management, string manipulation, and configuration tweaks, enhancing its capabilities in data manipulation, evasion, and persistence.

5.2.2. Ransomware Family Investigations

In addition to observing these function groupings across all ransomware families in our study, we also noticed trends in the prevalence of certain functions within each family. We measured the frequency with which each Windows API function was imported by each of the five most common ransomware families in our dataset. As we show in Figures 4-8, there are clear differences in how frequently certain functions are imported by specific families. We provide a survey of the five most common families in our study, describing which behaviors their commonly-imported Windows API functions may enable them to perform.

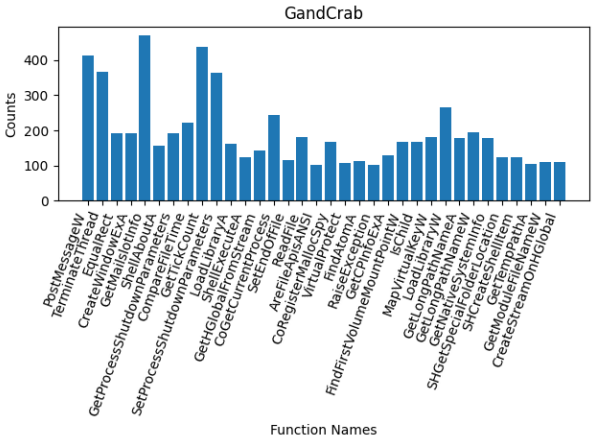

Figure 4: Function counts in Gandcrab files

GandCrab The unique imports utilized by GandCrab are illustrated in Figure 4. An analysis of the API calls indicates a predominance of behaviors 1, 2 and 5. Functions like "GetMailSlotInfo", "PostMessageW", "TerminateThread", "SetProcessShutdownParameters", and "GetTickCount" are most frequent within these clusters. This pattern suggests that GandCrab ransomware aims to access files and directories and establish a connection to its command and control server to encrypt the entire machine.

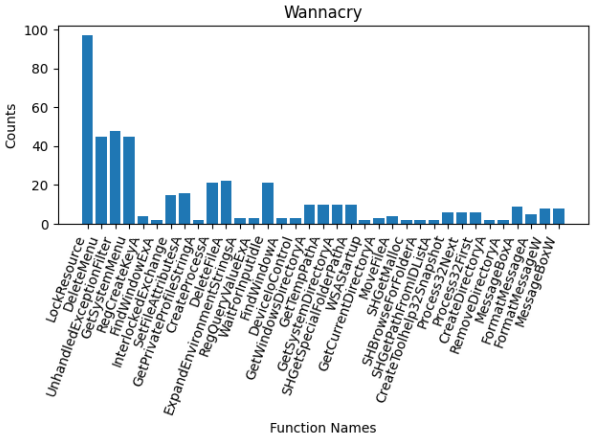

Figure 5: Function counts in WannaCry files

Wannacry Figure 5 reveals that Wannacry ransomware also employs behaviors 2 and 3. However, Wannacry distinguishes itself by utilizing additional API functions such as "LockResource", "GetSystemMenu", "InterlockedExchange", "DeleteFileA", "RemoveDirectoryA", and "FindWindowA", which are not as common in other ransomware families. WannaCry appears more likely to take destructive actions than other ransomware, and open-source reporting corroborates our findings - it deletes and overwrites files on the desktop and in the User folder. [21]

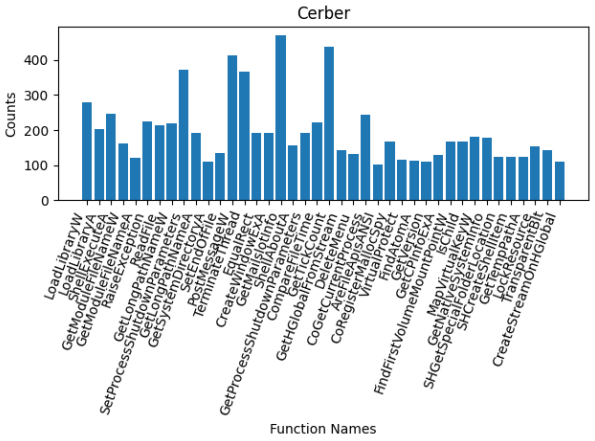

Figure 6: Function counts in Cerber files

Cerber As depicted in Figure 6, Cerber exhibits characteristics common to both GandCrab and Wannacry. The observed frequency of all function names is consistently high, which suggests they exhibit a wide variety of behaviors.

Gotango Figure 7 displays the API functions that Gotango ransomware exploits, including behaviors 2 and 4. Gotango engages in File and Library Management operations, using functions like “DeleteFileA" and “GetSystemDirectoryA", while showing a heightened focus on Shell and COM operations. Functions such as "Process32Next" and "Process32First" allow the malware to check if processes of interest are currently running.

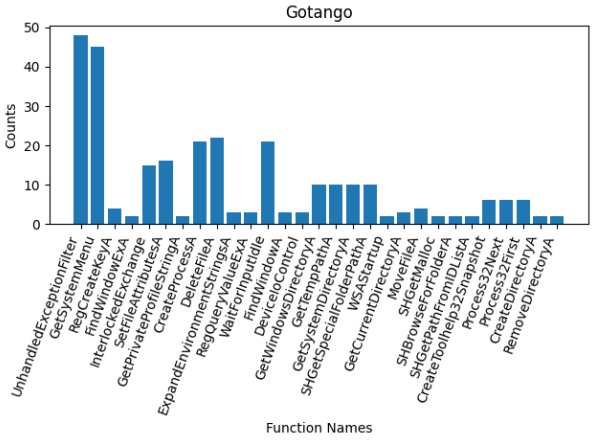

Figure 7: Function counts in Gotangto files

CryptXXX The API calls leveraged by CryptXXXpredominantly exhibit behaviors 3 and 5. This ransomware variant utilizes string-related functions such as “strCmpLogicalW", “StrStrNW", “StrCmpW", and “IsBadStringPtrA", among others. Additionally, it engages functions like “CryptVerifySignatureW" and makes extensive use of graphics control commands, including “TransparentBlt" and “AlphaBlend", suggesting a diverse operational approach.

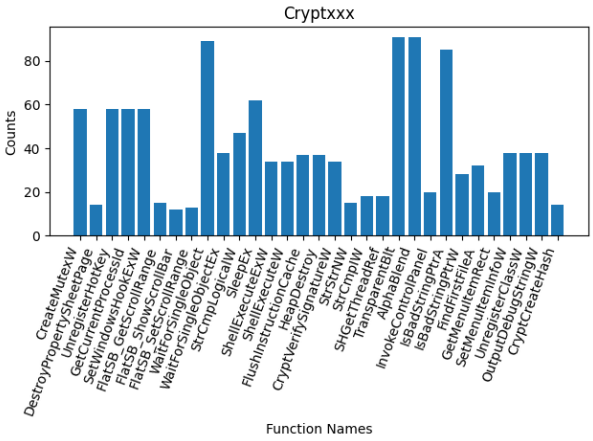

Figure 8: Function counts in CryptXXX files

5.3. Function Signature Validation

We employed the MalDICT-Behavior dataset to verify the accuracy of our function signature generation approach [22]. MalDICT-Behavior includes 105,849 malicious files tagged as ransomware. We queried the VirusTotal API for these files between February and April 2023, and we used AVClass to obtain malware family labels for them [19]. MalDICT-Behavior does not exclusively contain Windows PE files. We selected the 386 generated signatures corresponding to ransomware with known family names (Gandcrab, Cerber, WannaCry, CryptXXX, Exxroute, Tovicrypt, Gotango, Satan and Titirez).

Then, we scanned the 105,849 ransomware files selected from MalDICT-Behavior using these 386 signatures. A total of 6667 unique files were matched, and there were 25,912 total signature matches. This is an average of 3.88 signature matches per detected file, due to multiple signatures generated for most families. As we later discuss in more detail, it was very rare for files to be detected by signatures for different families.

To measure the performance of the scans, we identified 62,308 Windows PE files with a valid Import Address Table in our dataset of 105,849 files. Of these, 55,701 had AVClass labels. Because it was possible for a file to be detected by multiple signatures, we strictly defined the conditions that must be met in order to achieve a true positive:

- True Positive (TP): The file is detected by at least one signature, and the AVClass family is correct (for all signatures, if there are multiple detections).
- False Positive (FP): The file is detected by a signature, but the AVClass family does not match the family the signature is meant to detect.
- False Negative (FN): The file belongs to one of the families of interest, but no signature detects it.
- True Negative (TN): The file does not belong to any of the families of interest, and no signature matches it.

Table 2
Contingency Table Results

	Predictive	
Actual	**Positive**	**Negative**
Positive	6558	9505
Negative	109	39529

Table 3
Classifier Performance Metrics

Measurement	Value
Precision (PPV)	0.9834
Recall (TPR)	0.4082
Specificity (TNR)	0.9972
F1 Score	0.5765
False Positive Rate (FPR)	0.0027
False Negative Rate (FNR)	0.5918
Positive Predictive Value (PPV)	0.9834
Negative Predictive Value (NPV)	0.8053

Table 2 is the resulting contingency table for the 55,701 scanned PE files with AVClass labels. We computed multiple metrics to evaluate the performance of our generated signatures, shown in Table 3.

5.4. Discussion

In our analysis, we identified just 116 false positives, and only seven files were matched by signatures for different families. Four files were matched by both Gandcrab and Titirez signatures and three files were matched by both CryptXXX and Tovicrypt signatures. Since CryptXXX and Tovicrypt are variants, these false positives are not unexpected.

Based on the metrics in Table 3, we can conclude that our approach demonstrates satisfactory overall performance in identifying known ransomware families. The high precision (98.34%) and specificity (99.72%) values suggest that our signatures have a very low false positive rate. Due to the extreme class imbalance of malware to benign files, antivirus products place high emphasis on this quality [23].

Although our signature generation approach prioritizes minimizing false positives, users have the option of changing multiple hyper-parameters which would increase recall at the expense of lowered precision. Additionally, we believe that some of the false negatives produced by our generated signature can be explained by our selected evaluation dataset. The SOREL-20M dataset used for generating signatures only includes malware captured between 2017 and 2019 [18]. However, the MalDICT dataset includes malware first captured between 2006 and 2023. There are many ransomware families in MalDICT which do not appear in SOREL, and multiple ransomware families from our case study have been under active development since 2019.

6. Conclusion

We have demonstrated a method for generating signatures for known ransomware families using unique combinations of Windows API functions. Unlike traditional byte pattern signatures, our function signatures are unaffected polymorphism and other common evasion techniques which alter the malware's code. We created a scalable implementation which can efficiently cluster hundreds of thousands of malicious files and automatically generate a robust and actionable function signature for each.

We performed a case study on the unique function combinations within the well-known GandCrab, Wannacry, Cerber, Gotango, and CryptXXX families, identifying unique behaviors and API function

usage patterns. Each ransomware family's distinct approach to system infiltration and file manipulation was highlighted, providing insights into their operational mechanisms and objectives.

Ultimately, our contributions underscore the significance of meticulous experimental analysis in understanding ransomware's evolving threats. Furthermore, although the scope of our study was limited to ransomware, our function signature generation method can be applied to any Windows executable malware. The implications of this research will help broaden understanding of malware trends and offer a robust framework for automatically identifying and classifying these complex and ever-evolving threats.

7. Future Work

The current study has demonstrated promising results in using HDBSCAN for clustering ransomware samples and generating YARA rules. However, there are several areas for further investigation to enhance the robustness and applicability of our methodology.

While our signature generation method has shown good accuracy in detecting known ransomware families, the detection error rate for new malware, especially in broad classes such as trojans and backdoors, remains high. This indicates that while our approach is effective within the specific domain of ransomware, its generalizability to other types of malware needs improvement. Future work will focus on refining our algorithm to reduce this error rate, potentially through the incorporation of more diverse features and advanced machine learning techniques.

In addition, benchmarking our methodology against existing state-of-the-art approaches is essential for placing our findings within the broader context of ransomware detection. Future work will involve such benchmarking to validate our approach and highlight areas for further refinement. By addressing these key areas, future research will build upon the foundation laid by our current study, enhancing the robustness, accuracy, and comprehensiveness of ransomware detection methodologies.

References

[1] A. Petrosyan, Global firms targeted by ransomware 2023, 2024. URL: https://www.statista.com/statistics/204457/businesses-ransomware-attack-rate/.

[2] D. Santos, Ransomware Retrospective 2024: Unit 42 Leak Site Analysis, 2024. URL: https://unit42.paloaltonetworks.com/unit-42-ransomware-leak-site-data-analysis/.

[3] C. Team, Ransomware Hit $1 Billion in 2023, 2024. URL: https://www.chainalysis.com/blog/ransomware-2024/.

[4] M. Botacin, F. D. Domingues, F. Ceschin, R. Machnicki, M. A. Zanata Alves, P. L. De Geus, A. Grégio, AntiViruses under the microscope: A hands-on perspective, Computers & Security 112 (2022) 102500. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167404821003242. doi:10.1016/j.cose.2021.102500.

[5] L. McInnes, J. Healy, S. Astels, hdbscan: Hierarchical density based clustering, The Journal of Open Source Software 2 (2017) 205.

[6] hdbscan, The hdbscan Clustering Library — hdbscan 0.8.1 documentation (2016). URL: https://hdbscan.readthedocs.io/en/latest/, available online:https://hdbscan.readthedocs.io/en/latest/.

[7] R. Campello, D. Moulavi, J. Sander, Density-based clustering based on hierarchical density estimates, volume 7819, 2013, pp. 160–172. doi:10.1007/978-3-642-37456-2_14.

[8] Crowdstrike, 5 most common types of ransomware - crowdstrike, 2023. URL: https://www.crowdstrike.com/cybersecurity-101/ransomware/types-of-ransomware/, available online:https://www.crowdstrike.com/cybersecurity-101/ransomware/types-of-ransomware/.

[9] Kaspersky, Ransomware Attacks and Types – How Encryption Trojans Differ, 2023. URL: https://www.kaspersky.com/resource-center/threats/ransomware-attacks-and-types, section: Resource Center.

[10] GrantMeStrength, Windows API index - Win32 apps, 2023. URL: https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list.

[11] M. Alazab, S. Venkataraman, P. Watters, Towards Understanding Malware Behaviour by the Extraction of API Calls, in: 2010 Second Cybercrime and Trustworthy Computing Workshop, IEEE, Ballarat, Australia, 2010, pp. 52–59. URL: http://ieeexplore.ieee.org/document/5615097/. doi:10.1109/CTC.2010.8.

[12] A. Walker, R. M. Shukla, T. Das, S. Sengupta, Runs in the Family: Malware Family Variants Identification through API Sequence and Frequency Analysis, in: 2023 International Conference on Multimedia Computing, Networking and Applications (MCNA), IEEE, Valencia, Spain, 2023, pp. 55–61. URL: https://ieeexplore.ieee.org/document/10185752/. doi:10.1109/MCNA59361.2023.10185752.

[13] A. Y. Daeef, A. Al-Naji, J. Chahl, Features Engineering for Malware Family Classification Based API Call, Computers 11 (2022) 160. URL: https://www.mdpi.com/2073-431X/11/11/160. doi:10.3390/computers11110160.

[14] R. A. Mowri, M. Siddula, K. Roy, Application of Explainable Machine Learning in Detecting and Classifying Ransomware Families Based on API Call Analysis, 2022. URL: http://arxiv.org/abs/2210.11235, arXiv:2210.11235 [cs].

[15] P. Mohan Anand, P. Sai Charan, S. K. Shukla, A Comprehensive API Call Analysis for Detecting Windows-Based Ransomware, in: 2022 IEEE International Conference on Cyber Security and Resilience (CSR), IEEE, Rhodes, Greece, 2022, pp. 337–344. URL: https://ieeexplore.ieee.org/document/9850320/. doi:10.1109/CSR54599.2022.9850320.

[16] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero, A. Tonda, Countering android malware: A scalable semi-supervised approach for family-signature generation, IEEE Access 6 (2018) 59540–59556. doi:10.1109/ACCESS.2018.2874502.

[17] E. Raff, R. Zak, G. L. Munoz, W. Fleming, H. S. Anderson, B. Filar, C. Nicholas, J. Holt, Automatic Yara Rule Generation Using Biclustering, in: Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security, 2020, pp. 71–82. URL: http://arxiv.org/abs/2009.03779. doi:10.1145/3411508.3421372, arXiv:2009.03779 [cs, stat].

[18] R. Harang, E. M. Rudd, SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection, 2020. URL: http://arxiv.org/abs/2012.07634, arXiv:2012.07634 [cs].

[19] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, AVclass: A Tool for Massive Malware Labeling, in: F. Monrose, M. Dacier, G. Blanc, J. Garcia-Alfaro (Eds.), Research in Attacks, Intrusions, and Defenses, volume 9854, Springer International Publishing, Cham, 2016, pp. 230–253. URL: http://link.springer.com/10.1007/978-3-319-45719-2_11. doi:10.1007/978-3-319-45719-2_11, series Title: Lecture Notes in Computer Science.

[20] L. McInnes, J. Healy, S. Astels, Parameter selection for hdbscan— hdbscan 0.8.1 documentation, 2016. URL: https://hdbscan.readthedocs.io/en/latest/parameter_selection.html#min-samples-label, available online: https://hdbscan.readthedocs.io/en/latest/parameter_selection.html#min-samples-label.

[21] C. Team, How Can Disk Drill Help with Wannacry Ransomware Attack?, 2017. URL: https://www.cleverfiles.com/help/wannacry-recovery.html.

[22] R. J. Joyce, E. Raff, C. Nicholas, J. Holt, MalDICT: Benchmark Datasets on Malware Behaviors, Platforms, Exploitation, and Packers, 2023. URL: http://arxiv.org/abs/2310.11706, arXiv:2310.11706 [cs].

[23] A. T. Nguyen, E. Raff, C. Nicholas, J. Holt, Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints, 2021. URL: http://arxiv.org/abs/2108.04081, arXiv:2108.04081 [cs].

A. Appendix

Table 4
API Calls Categorization Examples

Function Behavior	Windows API Calls
Behavior 1: System and Process Functions	GetTickCount (438), SetProcessShutdownParameters (371), TerminateThread (365), CompareFileTime (221), GetProcessShutdownParameters (192), GetNativeSystemInfo (178), VirtualProtect (115), GetVersion (109)
Behavior 2: File and Library Management	LoadLibraryW (279), LoadLibraryA (203), LockResource (154), GetHGlobalFromStream (143), GetModuleFileNameW (162), GetModuleFileNameA (121), GetSystemDirectoryA (109), CreateStreamOnHGlobal (109)
Behavior 3: GUI and Window Management	PostMessageW (413), CreateWindowExA (193), EqualRect (193), ShellAboutA (156), TransparentBlt (142)
Behavior 4: Shell and COM Operations	ShellExecuteW (6436), ShellExecuteA (246), CoGetCurrentProcess (244), CoRegisterMallocSpy (167), SHGetSpecialFolderLocation (125), SHCreateShellItem (125)
Behavior 5: Utility Functions	GetMailslotInfo (470) GetLongPathNameW (218), GetLongPathNameA (192), InitCommonControls (187), MapVirtualKeyW (182), IsChild (167), FindFirstVolumeMountPointW (167), GetCPInfoExA (128), GetTempPathA (124), AreFileApisANSI (101), RaiseException (225)