DAKOTA STATE
UNIVERSITY

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Philosophy degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: Mar Castro Student ID: 101009287

Dissertation Title:
Malware Pattern of Life Analysis

Graduate Office Verification: DocuSigned by: Abby Chowning F44C8D9E621C417... Date: 11/20/2023

Dissertation Chair/Co-Chair: DocuSigned by: Austin O'Brien E94723CA282F45C... Date: 11/20/2023
Print Name: Austin O'Brien

Dissertation Chair/Co-Chair:__________ Date: __________
Print Name: __________

Committee Member: DocuSigned by: Yong Wang 70AB505BC7B649E... Date: 11/20/2023
Print Name: Yong Wang

Committee Member: DocuSigned by: Cherie Noteboom 897E5127036A49C... Date: 11/21/2023
Print Name: Cherie Noteboom

Committee Member:__________ Date: __________
Print Name: __________

Committee Member:__________ Date: __________
Print Name: __________

Submit Form Through Docusign Only
or to Office of Graduate Studies
Dakota State University

MALWARE PATTERN OF LIFE ANALYSIS

A dissertation submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Cyber Operations

Fall 2023

By

Mar Castro

Dissertation Committee:

Dissertation chair: Dr. Austin O'Brien

Committee member: Dr. Cherie Noteboom

Committee member: Dr. Yong Wang

DAKOTA STATE UNIVERSITY®

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Philosophy in Cyber Operations degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: _Mar Castro_________________________

Dissertation Title: _____Malware Pattern of Life Analysis______

Dissertation Chair/Co-Chair:______Dr. Austin O'Brien_____ Date: _______________

Committee member:__Dr. Cherie Noteboom__________________ Date: _______________

Committee member:______Dr. Yong Wang__________ Date: _______________

ABSTRACT

Many malware classifications include viruses, worms, trojans, ransomware, bots, adware, spyware, rootkits, file-less downloaders, malvertising, and many more. Each type may share unique behavioral characteristics with its methods of operations (MO), a pattern of behavior so distinctive that it could be recognized as having the same creator. The research shows the extraction of malware methods of operation using the step-by-step process of Artificial-Based Intelligence (ABI) with built-in Density-based spatial clustering of applications with noise (DBSCAN) machine learning to quantify the actions for their similarities, differences, baseline behaviors, and anomalies. The collected data of the research is from the ransomware sample repositories of Malware Bazaar and Virus Share, totaling 1300 live malicious codes ingested into the CAPEv2 malware sandbox, allowing the capture of traces of static, dynamic, and network behavior features. The ransomware features have shown significant activity of varying identified functions used in encryption, file application programming interface (API), and network function calls. During the machine learning categorization phase, there are eight identified clusters that have similar and different features regarding function-call sequencing events and file access manipulation for dropping file notes and writing encryption. Having compared all the clusters using a “supervenn” pictorial diagram, the characteristics of the static and dynamic behavior of the ransomware give the initial baselines for comparison with other variants that may have been added to the collected data for intelligence gathering. The findings provide a novel practical approach for intelligence gathering to address ransomware or any other malware variants’ activity patterns to discern similarities, anomalies, and differences between malware actions under study.

DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

____

Mar Castro

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Malware has been around for years. Both the industry and research community have developed, used, and researched different algorithms, including machine learning (ML), for malware detection to prevent them from being introduced into computer systems. However, they have not delved into their pattern of life to drill down to dynamic events of malware's methods of operations (MMO). Exposing malware's characteristics and habits allows the communities to dissect and discover the specific behavioral representation of data to its granular event level. Therefore, the study aims to establish the quantitative analysis of the current baseline, similarities, differences, and anomalies of static and behavior features to the level of detail in its procedural structural behavior. To accomplish the research, the Artificial-Based Intelligence (ABI) process and the DBSCAN machine learning (ML) tool, a significant novel research methodological profiling, paves the way to discover malware operations. The outline for the chapter provides a discussion of the background of the problem, the statement of the problem, motivation, research questions, and objectives.

A. Background of the Problem

Malware identification and classification extensively use research on features from static, dynamic, or heuristic analysis to identify the patterns of whether the malware is either malware or not. The industry and research community only focus on detection by establishing the pattern. However, the previous research failed to investigate what makes malware unique and identify its detailed processes and static and behavioral characteristics. The researchers did not present the malware's drill-down pieces of evidence or artifacts. Each type of malware family processes, methods, and tools must be examined to understand the more profound attributes of a particular type of malware. Since malware becomes sophisticated year after year from its normal behavioral activities, it is necessary to develop a pattern of life (PoL) to gather and discover new features. PoL is a set of behavior, activities, habits, and movements within the network and computing systems associated with particular malware over a given period. It is a technique to establish an entity's uniqueness if it is similar to other observed entities within the malware family. This malicious software could be parsed and analyzed by selecting a feature engineering baseline technique used by machine learning. Because of the technological advancement of the past decades, multiple sources of data to explore are easy tasks to collect, store, and analyze. This data collection mimics the Artificial-Based Intelligence (ABI) methodology, where data are compiled, dissected, and integrated from different sources for intelligence discovery. "*Intelligence discovery* is the ability to select, manipulate, and correlate data from multiple sources to identify information relevant to ongoing operations and requirements. Discovery is about better organizing and using the data that we already know. It is also about finding previously hidden patterns and anomalies—former Secretary Donald Rumsfeld's "unknown unknowns" [4]. After the collections, Machine Learning (ML) technique is used to facilitate the work to identify the level of intrusion of activities into the infective systems that might have occurred when the malware deviates from its habitual behavior. Therefore, the research establishes the pattern of life, allowing us to uniformly understand malware static and dynamic activities in the computing environment for their similarities, differences, variances, and anomalies.

B. Statement of the Problem and Motivation

Malware has its unique way of infiltrating the systems. They continue to evolve or change as our technology advances and the creativity of the design of the malware improves, rendering them challenging to detect. Regardless of their evolution to progress into an advanced state of change, each malware family may share the same baseline characteristics, which describe the effective methods of operations. Malware is about achieving a malicious goal by exploiting vulnerabilities in computer systems. The goals of these attacks can range from stealing confidential data, disrupting the system's operation, or destroying it. The kind of behavior must be probed and explored in detail to uncover the manner of its internal working. To discover the behavioral traits, the researcher conducts the pattern of life's behavior, the intelligence-gathering technique used by the military or law enforcement, to track the objects' behavior regarding their observed frequency and types of activities. The types of information gathered could indicate threats or criminal activities.

The research malware community has not fully explored the pattern of life events of the malware family. Therefore, it is a novel research endeavor to establish its standard baselines to discover patterns and anomalies and see malware's profound details despite its family structure. The density of types of malware has been increasing since early 2000. Sifting through the malware's frequency of events, event sequencing, and the Application Programming Interface (API) creates a natural set of patterns. There are many possibilities for pattern activities in the systems for feature discovery. The general taxonomy below describes the drill-down specification of the type of malware to its different variants.

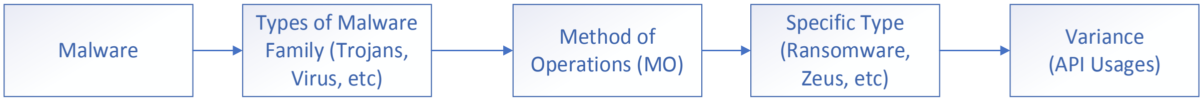

Malware
Types of Malware Family (Trojans, Virus, etc)
Method of Operations (MO)
Specific Type (Ransomware, Zeus, etc)
Variance (API Usages)

Figure A. General Malware Taxonomy

Each type of malware differs from one another. However, the research aims to dive into the specific one-type pattern of life to establish baselines of methods because it is conceptually similar to malware activities of the same family classification. Figure A depicts the fundamentally hierarchical structure of malware taxonomy from types to methods to the specific type and its variance. The ransomware is the target of malware to drill into, providing an in-dept look at the content.

C. Research Questions

The paper intends to reveal patterns and identify a change to establish a decision advantage to gather intelligence on the activities of the malware creators. Patterns could be perceived by identifying the pattern of a unit. As described by Platas in this web article [4],

> *"We need to understand not just the individual elements within this pattern unit but also how the pattern unit is repeated. If you see only* AB*, you don't have enough evidence to identify the pattern. But if you see the* AB *unit repeating, as in* ABABAB*, then you can be confident of your judgment [6]".*

The pattern repetition concept of malware activities running in the system could be perceived and reported through extensive analysis of malware features to establish patterns. As indicated by the Artificial-Based Intelligence (ABI) methodology, data collection derives from many sources [3]. For example, the research will gather static, behavioral, and network features to be compiled and analyzed. The use of machine learning analyzes patterns to facilitate and recognize patterns with outliers or deviations. The method enables a complete picture of family-related malware deployed in the wild west of the internet to show correlations between two malware's likeness and differences. The multiple sources of data observation allow the strength of several features to compensate for the lack of strengths of another feature. The past researchers not only did not delve into the detailed events of the malware activities but focused on the malware's patterns PE format, string patterns, opcode, memory process, configuration settings, API system calls, network, and bytecode to determine its likelihood of being malware. The missing research to pivot into is the distinctness of the features involved depending upon the malware type. Because of that reason, the three research questions need to address the following questions to examine the malware in depth from static, dynamic, and network points of view.

1. What are the feature similarities and differences (anomalies) of the malware feature activities collected over the years?
2. How do we identify the constraints needed to develop a common baseline, a hidden data pattern, for each malware classification or cluster type?
3. How can we determine if the variant types of malware in the wild were the original copies of other types of malware?

The questions would unify and improve the correct baseline identification of the clusters of malware deployed in the wild by studying the differences and similarities from its normal behavior, providing a basis for an accurate and robust pattern of life (PoL). The research novel endeavor will enable the formalization of the malware pattern of life of each type of malware classification.

D. Objectives

The research aims to demonstrate the novel idea of quality intelligence gathering on a specific malware activity by exploring and establishing its natural-occurring pattern of life. The study will sift through thousands of malware of the same type to extract insights. Capev2, derived from Cuckoo Sandbox, captures features that collect static, behavior, and network data, making ABI methodology collection processes easier. ABI would allow the integration of multiple sources to be parsed, examined, and analyzed for intelligence gathering to discover life patterns. Using the ABI method is not new because it has been applied in the military intelligence community [3]. The similarities and differences of the malware features, anomalies, or activities running within the computing systems may make it possible to give us origins, derivatives, or deviations from other source codes by establishing the baseline for each type of malware activity. The research drills more profoundly into the micro-level of a specific malware community; the study compares, contrasts, and understands standard methods of behavior, along with their variance. " Micro-level theories provide explanations limited to small slices of time, space, or numbers of people, such as Goffman's theory of facework, which explains how people engage in rituals during face-to-face interactions. Meso-level theories link the micro and macro levels. These are theories of organizations, social movements, or communities, such as Collins's theory of control in organizations. Macro-level theories explain larger aggregates, such as social institutions, cultural systems, and whole societies. Macro-level theories explain larger aggregates, such as social institutions, cultural systems, and whole societies" [7]. The research approach will build up the general pictorial patterns of a specific malware family and an anomaly. The scope is to see the patterns of malware profile methods and characteristics' behavior to assess features and determine their capabilities. The process below is the step-by-step process of the ABI section, which will guide the research to quantify malware traits [4].

> *1. Discovery (features collected) – the features of malware activities are available in static, behavioral, and network sources. They are quality selected, manipulated, and correlated to identify information needed for past operations and future events. In this paper, the initial research is to compile three sources for study.*
>
> *2. Assessment (quantification) – examining the aggregated data is essential to discovering statistical events needed to establish a pattern of activities. Here, data is synthesized to assess better the existing event pattern of the malware, which has been collected over the years.*
>
> *3. Explanation (similar events) – the discovered data pattern may relate to other types of malware; it may signifies other events' discovery to tell us that other malware may exhibit the same characteristics, rendering possible identical approaches.*
>
> *4. Anticipation (possible forecast) – the deviation of the malware activities may give future states of activities of the malware creator. As we gather and compile more data to integrate into the existing pile from the collected malware, we may discover other actions of malware creators. Has the malware creator created or modified a form of malware with the same goal of preventing anti-analysis?*
>
> *5. Delivery is the final phase for reports that summarize the over activities and patterns of the type of malware chosen to be analyzed.*

These phases provide profiling, identification, and comparison of malware activities statistically and dynamically that hinge upon the malware sequences, frequencies, and event data parameter activities. Thus, a feasible fingerprint pattern of life and its anomalous activities is created. The sequence search is similar to the published research in Finding Patterns in Biological Sequences stating that "Finding Patterns in Biological Sequences where proteins where all of the sequences are aligned to identify conserved regions which are used to generate models that represent ancient conserved regions" [2]. This is called the phylogenetic relationships between entities. However, the paper focuses on several merged inputs for analysis as part of the ABI approach. The result of the study becomes valuable to the malware research practitioner, Incident Response Team (IRT) of Defense Industrial Base (DIB) as part of NIST 3.6 and related to software antivirus providing the process and procedural framework intelligence gathering. Using the ABI object process creates integrated and coherent visualization of the picture behavioral events of a particular type of malware to provide actional intelligence for the malware analyst of interested organizations.

CHAPTER 2

LITERATURE REVIEW

The literature review is used to discover other works that might not have been foreseen during the research in order not to replicate the works of others. It is essential that during the research phase, nothing on this project has rehashed existing work but instead adds to the existing body of knowledge of malware research activities. The existence of malware may have been around since the inception of the computer age. Only now that malware has become either a nuisance or even become destructive. Researchers have produced many algorithms, from customized novel algorithms to using existing machine learning (ML) application programming interfaces (API) for detection. No research papers, however, ever introduced the drill-down detail of what the malware is made from, rendering to be the research gap. Forty-three published papers have delved into this subject to identify malware using input data, customized algorithms, and machine learning for detection, as mentioned in Appendix B. The literature focuses on data patterns using specific input features for machine learning for parsing, scanning, and identifying whether the malware found is positive or negative. The input data listed in Appendix A are classified as PE Format, Function Name/String, Bytecode: Input Scale, ASM (OPCODE), Memory/Process, Log/Config Settings, API: System Calls, and Network Traffic/Packets. The literature research comprises probing the 11 publishers between 2008 and 2021, including Springer, Science Direct, IEEE, ACM Digital Library, Academic Conferences International Limited, MDPI, Research Gate, IOScience, Standford site, ndupress, and Sage Publications. Figure B shows that IEEE has the most extensive peer-reviewed source for the references. The explored keywords are malware, ransomware, Artificial-Based Intelligence, density-based spatial clustering of applications with noise, DBSCAN, and machine learning.

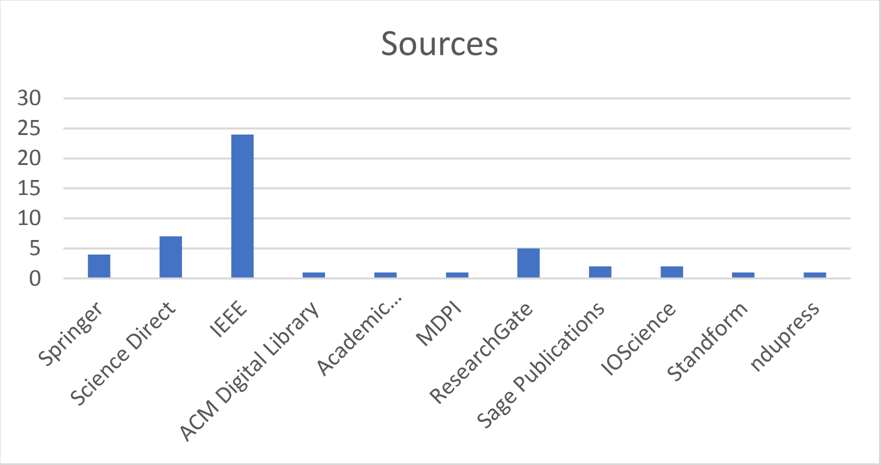


Figure B: Reference Source Statistics

Since there is no published research to expose and describe the malware characteristics, the study demonstrates to the research community a more general review of the drill-down data to its specific forms and patterns to better identify the type of malware for profiling. It shows the malware's API usage and sequence of call events to establish baselines, anomalies, similarities, differences, and intentions of the type of malware. For that reason, Artificial-Based Intelligence (ABI) using DBSCAN machine learning are used to establish the pattern of life with multiple input. The combination of these tools makes the research of the paper unique. The scikit-learn library makes the DBSCAN machine-learning algorithm for use [60]. However, it has never been used to analyze malware to establish the pattern of life. The research malware community was never practically used in the industry, field, or combination to analyze malware. Under certain circumstances, the research introduces the Pattern of Life (PoL) and Artificial-Based Intelligence (ABI) to demonstrate the academic aspect of the literature into practice to identify malware characteristics. The literature review listed in the section Appendices has summarized the related published papers as follows:

- **The pattern of Life (PoL), DBSCAN, and Activity-Based Intelligence (ABI):** Gross uses PoL and ABI extensively to analyze complex behavior as stated," POL analysis methods are particularly important to understand when attempting to track complex human" [1]. The research analysis of Gross is similar to this project in that the end goal is the same. Its task is to detect anomalies that deviate from normal behavior. However, it has not used DBSCAN machine learning to compare data. The paper only introduces the theoretical aspect of implementing ABI using the pattern of life for detection. In another paper published by Atwood from National Defense University Press, ABI has not been used extensively in practice but only in the theoretical aspect of using the Artificial-Based (ABI) process in military environmental settings [3]. It is likely that despite ABI's old concept of gathering intelligence, its potential applications have not been explored effectively due to its expensive operations. Nevertheless, it would be used for the research to explore its undeveloped capabilities.

- **Feature Engineering (FE).** Appendix A provides the general category of the features used for feature engineering (FE) by different literature authors. Its purpose is to extract information from Portable Executable (PE), Strings, Functions, ASM OPCODE, Bytecode, Memory Processes, and Microarchitectural events (Hardware Level). The features used are further explained in Appendix B feature engineering column, which derives from references 10 to 43 of the references section. The research topic under study uses mainly API dynamic behavior, the DLL from static analysis of the portable executable (PE), and the network behavior of the malware. These are the main features used by the CAPEv2 malware sandbox being studied and included for behavioral clustering and comparison analysis.
- **Machine Learning/Algorithm:** Published literature uses different algorithms or machine learnings, listed in Appendix B of column Technique (Algorithm/Machine Learning (ML)). DBSCAN has never been used or applied to cluster malware for its differences, similarities, and outliers in a practical manner. One mentioned that it had been used only to compare the differences of DBSCAN against other clustering algorithms [30]. In other words, it has never been used, tested, or tried out by anyone for actual malware feature analysis in real applications, so there is no way how it performs or even if it works.

- **Tools Used:** Appendix B [reference 26, 30, 33] identified three pieces of literature that use Cuckoo malware sandbox, a baseline software used by CAPEv2, which has more upgraded and improved malware analysis detection. It allows the researcher to have more accurate results by detecting granular levels of behavior in smaller quantities than in different conditions than it could before. The unstructured data, JSON format, the output of the malware static and behavioral analysis are dumped to a directory where the research project parse and dissect the contents to convert it to a comma delimited format, the preferred format data for machine learning used by scikit-learn.
- **Feature Engineering (FE), Level of Scan (LS), Algorithm:** The FE, LS, and Algorithm/Machine learning (A/ML) are the malware detection summary from references 10 to 43. Feature Engineering (FE) columns are used as an input with the corresponding algorithm and level of scans to predict whether the malware is identified as positive or negative. The reviewed literature has not identified DBSCAN machine learning to classify the positive or negative malware. In other words, DBSCAN was never put into practice to analyze malware. However, it was used for galaxy clustering by Zhang as stated, "*DBSCAN algorithm is the most effective and accurate algorithm. By comparing with the correct Figure, we can find that the DBSCAN algorithm can accurately identify all classes and eliminate noise interference to a certain extent, which is impossible to be achieved by the KMeans algorithm and the Decision Tree algorithm*" [5]. Based on the data results of this article, the data points are labeled for this type of DBSCAN analysis because it has mentioned the accuracy of the data. The accuracy estimate is only available if the data is labeled. In addition, Yang has also used DBSCAN combined with a Genetic Algorithm (GA) for fault detection of gas-insulated switchgear (GIS) [6]. However, this research project attempts to cluster known malware, such as ransomware, for its similarities, differences, and outliers analysis. Its intention is not to predict whether the data is found to be malware or not nor to be used whether the malicious codes are within the range of possible malware. The research design compares the ransomware belonging to different clusters for intelligence gathering, as mentioned in the ABI process.

In conclusion, the research literature emphasizes the use of Artificial-Based Intelligence (ABI), the effectiveness of using Use Density-Based Spatial Clustering of Applications with Noise (DBSCAN) for clustering, or the pattern of life (PoL) to detect anomalies. Nevertheless, the research gap is that the combination of ABI and DBSCAN for pattern life drill-down identification behavior was never emphasized in detail in analyzing the similarities and differences to reveal their character traits of a specific type of malware, along with the analysis of the sequence of events of the application programming interface (API). In addition, ABI methodology and DBSCAN, as described by the literature, only used the technique of using them, but they never used how to apply the concept in a practical manner. As a result, the research is a novelty attempt to reveal and apply the effectiveness of using the combination of both ABI and DBSCAN against ransomware to expose its intent behavior as part of intelligence gathering.

CHAPTER 3

3.1 Introduction

Chapter 3 introduces the research methods to describe the different parts of the study. It provides information on the malware data downloaded from the community research repository, the use of observing the malicious code using CAPEv2 malware sandbox, and the techniques and tools for extracting static, behavioral, and network data. The researcher further describes the chosen research design using Artificial-Based Intelligence (ABI), the purpose of the study, and the reasons for the design. In addition, the researcher discusses the methods used in analyzing data, the limitations, further discussions of the techniques, and the ethical considerations.

3.2 Research Design

Quantitative objective emphasizes measurements and statistical, mathematical, or numerical analysis of the collected data. Its goal is to collect information from an existing source, in this case, the malware source from an open or commercial malware provider, such as Malware Bazaar or VirusShare. The analysis results are transformed into numerical data, visualization of graphs, table data, and suitable charts corresponding to the measured quantity under study, which determines the relationships between two or more variables and features of the malware under study to establish baselines, similarities, differences, and anomalies. The study undergoes several phases of tasks to explain and drill down into the malware variant of the same malware type, such as ransomware, as the target samples. The Figure C diagram, the methodology, gives the groundwork for five-stepped from the collection, data analysis, feature extraction, and running machine learning using Artificial-Based Intelligence (ABI) methodology data that shows the step-by-phased analysis to establish and discover the pattern of life, along with the deviation and anomalies.

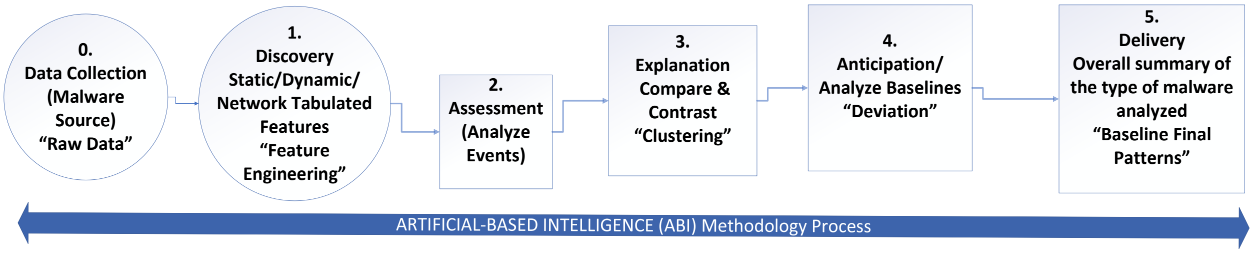


Figure C. ABI Methodology Life Cycle

0. Collect malware data – all malware belongs to the same type of classification. It is the critical phase of any research project because it involves data gathering from the respective sources; in this case, it is from a scientific malware research repository. It does not involve Personal Identifiable Information (PII) or Personal Health Information (PHI). Therefore, there are no privacy issues that were impacted during the research study.
1. Feature extraction of features – the characteristic of each malware is stored in the central database or comma-delimited format to get ingested into machine learning. The important features are identified and meaning to be used as input for machine learning. A subject matter expert (SME) or knowledgeable of the sample target is an essential ingredient to understanding the characteristical features to improve the models' accuracy, reduce the problem's complexity, and make the model more interpretable.
2. Feature analysis – the collected features are converted into a comma-delimited format, which is ingested into the chosen machine learning. Each data point sample is grouped or categorized into its own cluster that has similar features. Identifying the clustered data points gives the patterns of data that give way to predict how similar the malware is.
3. Descriptive analysis – the data is compared and differentiated, showing the summarized points. One of the most important sections of the research phase is statistical analysis, which is the foundation of answering the research questions. It helps to detect similarities among features, making it further to run other statistical analyses.

4. Anticipation phase – provides direction and narrows down the scope of data investigation after the malware clustering that helps to establish answering the three research questions.
5. It is an overall summary of the findings that conclude the results.

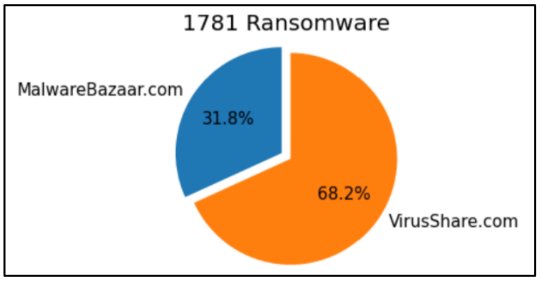
There is a lack of practical research on using this process. Each successive event phase is a form of data filtering and massaging to get the finalized results of the pattern of life. Each phase is a modular design with a plugged-play process or algorithm to improve intelligence gathering. The quantitative analysis approach of post-positivism, as stated by Crestwell," This worldview is sometimes called positivist/postpositivist research, empirical research science, and postpositivism. This last term is called post-positivism because it represents the thinking after positivism and recognizing that we cannot be positive about our claims of knowledge when studying the behavior and actions of humans" [7]. For that reason, the knowledge observed from the extracted malware features can be objective without absolute certainty. The quantitative analysis uses the idea of post-positivism and by merging into the Artificial-Based Intelligence (ABI) methodology process using the DBSCAN machine learning tool.

An ABI plus the use of DBSCAN is the process to quantify the research study and is appropriate since the study aims to examine the multisource feature collections of static, behavioral, and network data malware activities to quantify the malware features. As stated by Atwood, " Today's focus on single-source exploitation in an environment of multisource data availability clearly hinders analysts from understanding and conveying the overall meaning of the integrated results" [3]. The tasks analyze the activity of multiple occurrences of essential characteristics of malware to explain the event frequencies, event sequences, and other possible data event activities in the Windows operating systems in their natural settings. The overall results give the results of descriptive and correlational designs of the variables that measure different types of malware variables describing frequencies and events, along with correlations.

3.3 Research Methodology

The type of data to collect for this method is an observational method where ransomware is individually run into Windows 10 Operating System (OS) and is observed by the CAPEv2 malware sandbox. The operating system is rebuilt and purged automatically for each ransomware to run by itself to prevent interference from other malicious code in the system. In addition, the system's default security baselines set by Microsoft were turned off, for example, Antivirus (AV), memory integrity, and Windows Firewall (F/W) to allow the malware to communicate externally. The required settings would allow the malware behavioral data to run as part of the collection process and be able to objectively, logically, statistically, and unbiased measurements. The following section will follow the framework of the research design using the Artificial-Based Intelligence methodology for data collection, feature discovery, assessment of data, explanation, possible Anticipation, and the delivery of the final reports.

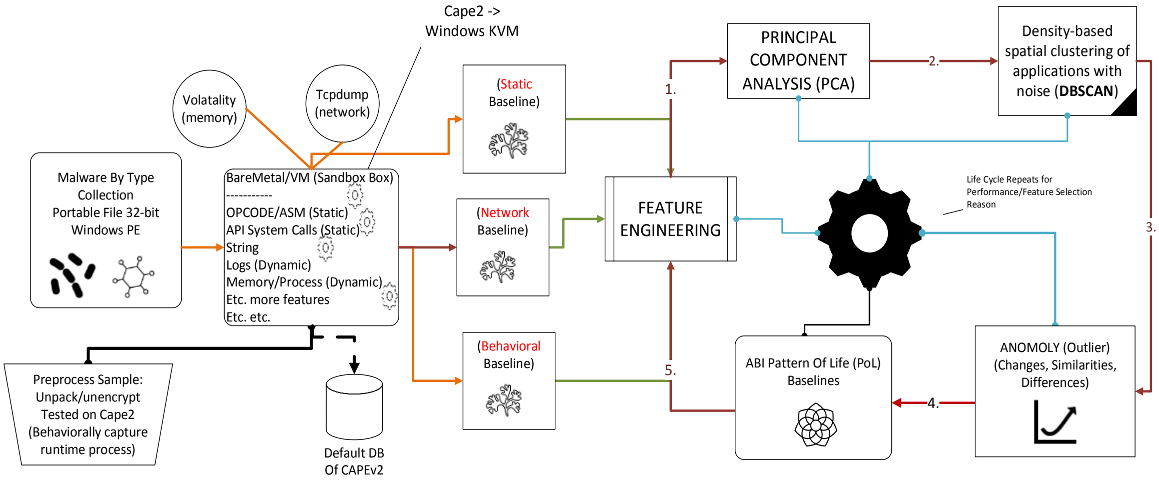
3.3.0 Phase 0: Data Collection



There is over 21k population collected from authoritative data sources, 1781 of which are scanned with YARA scripts, which consists of 510 confirmed ransomware from Malware Bazaar [52] and 1271 confirmed ransomware from Virus Share [53] repositories. The ransomware samples were ingested into the CAPEv2 sandbox to capture the activities of the process of the malicious software. The rest of the ransomware that failed to identify by Yara scripts was still tested to run into the malware sandbox. However, these portable executables (PE) failed to run because the error states that it is Russian, Chinese, Spanish, Iran, or other languages quoting “Unconventional language used in binary resources.” As a result, they were excluded from the study because the main goal of the data is to collect confirmed reliable ransomware data as input parameters for machine learning clustering.

3.3.0.1 Instrument (Hardware/Software)

Since malware is dangerous, it tends to compromise the network and the connected machines, and the lab environment is isolated from the rest of the systems. As a result, the architectural design diagram described in Figure D shows the general overview of the lab environment from collection data, analyzation, feature engineering, descriptive analysis, and the final results. The primary concern with component functionality is to ensure its structurally sound environmental laboratory accomplishes the goals of parsing and dissecting all the features of a specific type of study malware without affecting and infecting essential files.


Cape2 -> Windows KVM
Volatility (memory)
Tcpdump (network)
Malware By Type Collection Portable File 32-bit Windows PE
BareMetal/VM (Sandbox Box)

OPCODE/ASM (Static)
API System Calls (Static)
String
Logs (Dynamic)
Memory/Process (Dynamic)
Etc. more features
Etc. etc.
Preprocess Sample: Unpack/unencrypt Tested on Cape2 (Behaviorally capture runtime process)
Default DB Of CAPEv2
(Static Baseline)
(Network Baseline)
(Behavioral Baseline)
1.
FEATURE ENGINEERING
5.
PRINCIPAL COMPONENT ANALYSIS (PCA)
2.
Density-based spatial clustering of applications with noise (DBSCAN)
3.
Life Cycle Repeats for Performance/Feature Selection Reason
ABI Pattern Of Life (PoL) Baselines
4.
ANOMOLY (Outlier) (Changes, Similarities, Differences)

Figure D: Malware Extraction Life Cycle

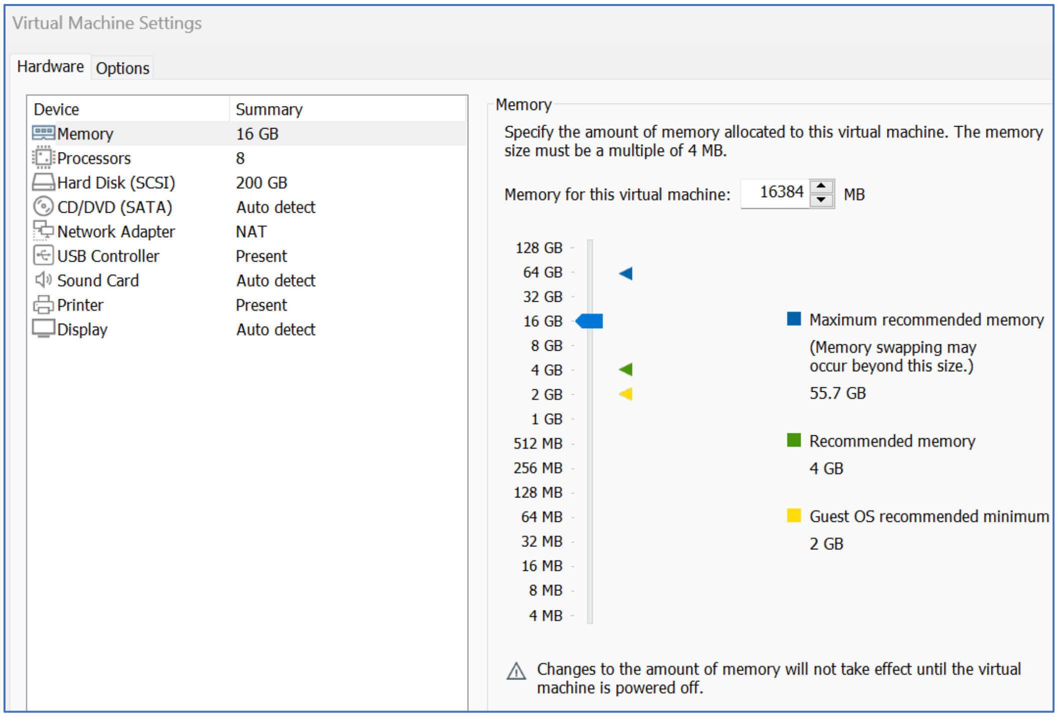
The diagram starts with the data collection, mainly portable executable (PE) file samples from Virus Share and Malware Bazaar malware repositories. The Sandbox, CAPEv2, analyzes the ransomware after the completed run and extracts the static, network, and behavioral processes. The following steps show the detailed settings and configurations:

Step 1: Yara Scripts (Confirmed Ransomware Data)

The YARA scripts are used to filter out malicious portable executable data by searching for patterns within the malware data. It allows identifying of quick, suspicious data requiring investigation of ransomware, the target samples. Under certain circumstances, they are used to isolate data that is deemed to be ransomware. The YARA scripts being used are from different sources, namely 3vangel1st, bartblaze, BinaryAlert, f0wl, Open-Source-YARA-rules, Neo23x0, Petya, Reversing labs, TJN yara repo, crime_wannacry, and Yara rules project [54-65]. They are combined into one, removing duplicates. After running the ransomware in the malware sandbox labs, there are roughly 1300 unlabeled rows of data that run successfully with static, dynamic, and network with fewer null values**.** As a result, 1300 will be used for this research giving us the completed run on the CAPEv2, an **in-house** installed malware sandbox used as an instrument for data extraction.

Step 2: Virtual Machine Configuration

A virtual machine (VM) is a safer way to analyze and study malware behavior than running in a virtual machine because it enables wiping and recreating the VM at any time. It can provide an isolated environment for the malware to trigger their behavioral actions within the isolated systems that can be controlled and intercepted. The processor's configuration must be set with the virtualization engine enabled for Virtualize Intel VT-x/EPT or AMD-V/RVI to enable the Windows 10 Virtual Machine (VM) to run inside of the VMWare virtual machine. The Hard Disk is set to 200 GB to accommodate 2000 live viruses for processing, parsing, and collection. Figure E shows the virtual machine's configuration for the lab.



Virtual Machine Settings

Hardware Options

Device	Summary
Memory	16 GB
Processors	8
Hard Disk (SCSI)	200 GB
CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 16384 MB

🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
55.7 GB

🟩 Recommended memory
4 GB

🟨 Guest OS recommended minimum
2 GB

⚠ Changes to the amount of memory will not take effect until the virtual machine is powered off.

Figure E: VMware Workstation Configuration Settings

Step 3: CAPEv2 Malware Sandbox

The sandbox software is downloaded from the CAPEv2 website [51] and installed on Ubuntu Linux version 20.04 using VMware Workstation 16 with the configuration diagram below in Figure F. The malware sandbox deploys ransomware one at a time to Windows 10 running in Kernel-based Virtual Machine (KVM). After each deployment, the ransomware's collection processes and other dynamic behavior are collected to a temporary directory where the output JSON files are stored. To ensure that malware runs without any restrictions or blockage, the security settings of the Windows 10 running in KVM, such as Windows Defender, Anti-Virus, Firewall (F/W), and device security, are disabled.

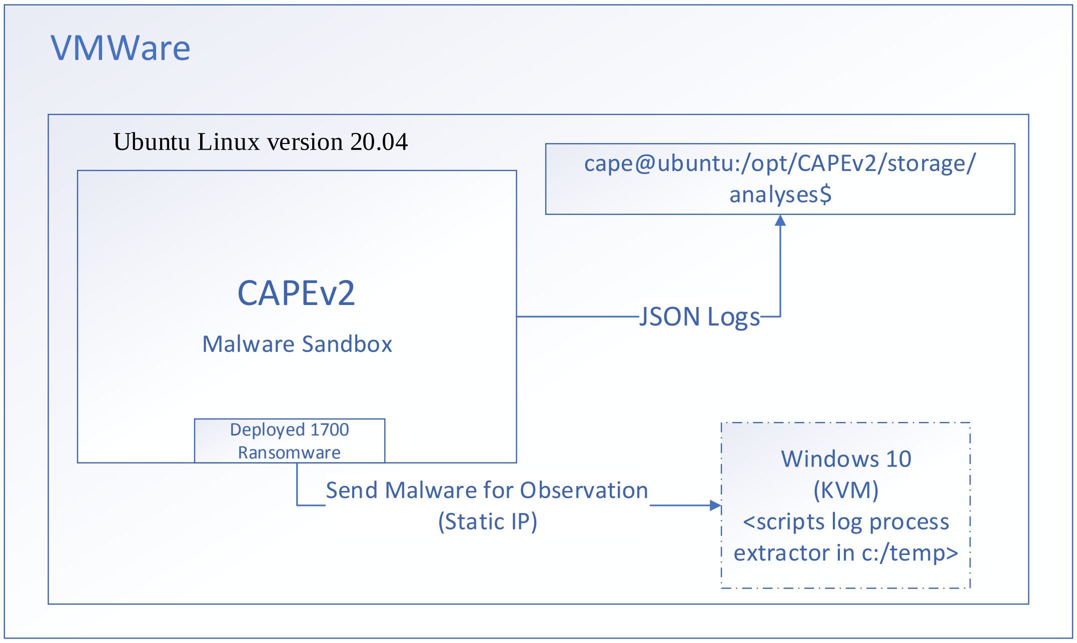
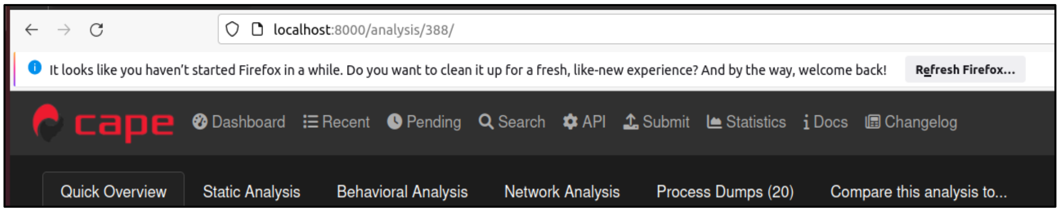

VMWare
Ubuntu Linux version 20.04
cape@ubuntu:/opt/CAPEv2/storage/analyses$
CAPEv2
Malware Sandbox
JSON Logs
Deployed 1700 Ransomware
Send Malware for Observation (Static IP)
Windows 10 (KVM) <scripts log process extractor in c:/temp>

Figure F: VMWare Malware Sandbox Setup

Figure F is the general high-level diagram of the malware sandbox configuration. Windows 10 is configured to have 8 GB RAM with at least 80 GB allocated space to trick the malware as if it were running on a real machine. That is, the environmental design setup should appear to be a natural environment making the malware believe it is a physical machine.

Step 4: Extraction and Conversion

The raw data output of the malware run within the sandbox, CAPEv2, is in JSON format. Using the domain knowledge of the malware characteristics, several categories can be parsed or to conduct data mining, namely Static, Behavioral, Network, Dropped Files, Process Dumps, and Payloads. Behavioral data is the application programming interface (API) that manipulates the files written to the systems, including the dropped files and the process of the executable running. Static is the executable portal mapping of the malware. It contains the name generic name of the malware, the Dynamic Link Library (DLL), object code, and other data structures that encapsulate Windows portable executable (PE) information. Finally, the network is a means to communicate using HTTP requests. Figure E is the snapshot of the menu in which ransomware runs successfully. The "Dropped Files", "Process Dumps", and "Payloads" are populated with the indication of number one (1). Having 1700 malicious software to ingest in the CAPEv2 malware sandbox takes at least 10 to 11 weeks.



localhost:8000/analysis/388/

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Figure G: CAPEv2 Menu Malware Sandbox Sample # 388

The data representation of each report is stored in JSON format, which is named report.json by the CAPEv2 malware sandbox. It is stored in the default installation, usually in /opt/CAPEv2/storage/analyses/reports directory. The portion of the JSON output is formatted to comma-delimited files format, which is required for the machine learning to ingest. The JSON formatting is written in a customized Python program, as shown in **Figure H** Unified Modeling Language (UML) Diagram. The 1700 ransomware of JSON extracting and formatting takes three weeks for conversation to comma-delimited files. The Python program reads the location of the JSON logs output by the malware sandbox one at a time. It converts it to a Python dictionary and dumps the file comma-delimited to an external file using append mode.

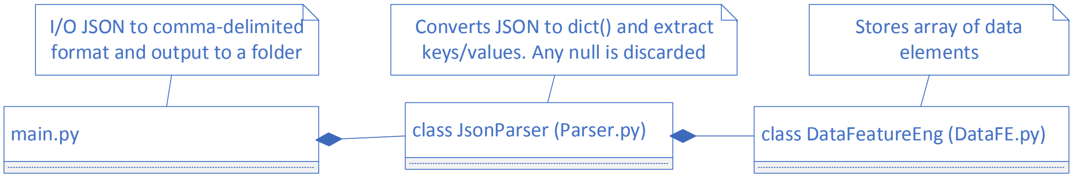
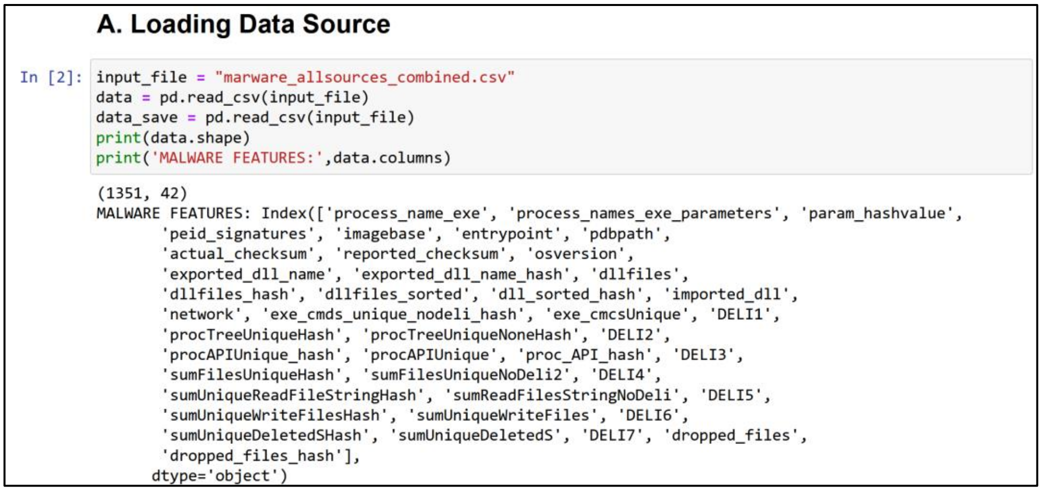

I/O JSON to comma-delimited format and output to a folder
Converts JSON to dict() and extract keys/values. Any null is discarded
Stores array of data elements
main.py
class JsonParser (Parser.py)
class DataFeatureEng (DataFE.py)

Figure H: Modularized/Customized Python Program UML Design

Figure H shows the primary function controlling the JsonParser class object to parse and formats the JSON format. Each JSON output is stored in a Python dictionary to probe the contents to prepare for skipping nulls and to extract meaningful data, such as static, dynamic, and network data. The final output is stored in a comma-delimited format.

Step 4: Results of Conversion

Snippet A snapshot shows the snippets of the code of “Jupyter Notebook,” showing the input of the CSV file as an input for the machine learning. It displays the raw data of comma-delimited columns extracted from JSON files outputted from the CAPEv2 malware sandbox. The 1351 out of 1700 result is the total number of ransomware eliminating “null” row data, capturing relevant features based on the domain knowledge, and capturing with low missing data. The DELI* columns are designed to separate malware features from static, dynamic, and network behavior, as described in the column section, which needed to be further analyzed for feature engineering. Some column features identified in Figure I have textual descriptions captured during the observation and may require to be further broken down into more feature extraction.



A. Loading Data Source

```
In [2]: input_file = "marware_allsources_combined.csv"
        data = pd.read_csv(input_file)
        data_save = pd.read_csv(input_file)
        print(data.shape)
        print('MALWARE FEATURES:',data.columns)

        (1351, 42)
        MALWARE FEATURES: Index(['process_name_exe', 'process_names_exe_parameters', 'param_hashvalue',
               'peid_signatures', 'imagebase', 'entrypoint', 'pdbpath',
               'actual_checksum', 'reported_checksum', 'osversion',
               'exported_dll_name', 'exported_dll_name_hash', 'dllfiles',
               'dllfiles_hash', 'dllfiles_sorted', 'dll_sorted_hash', 'imported_dll',
               'network', 'exe_cmds_unique_nodeli_hash', 'exe_cmcsUnique', 'DELI1',
               'procTreeUniqueHash', 'procTreeUniqueNoneHash', 'DELI2',
               'procAPIUnique_hash', 'procAPIUnique', 'proc_API_hash', 'DELI3',
               'sumFilesUniqueHash', 'sumFilesUniqueNoDeli2', 'DELI4',
               'sumUniqueReadFileStringHash', 'sumReadFilesStringNoDeli', 'DELI5',
               'sumUniqueWriteFilesHash', 'sumUniqueWriteFiles', 'DELI6',
               'sumUniqueDeletedSHash', 'sumUniqueDeletedS', 'DELI7', 'dropped_files',
               'dropped_files_hash'],
              dtype='object')
```

Snippet A: Jupyter Notebook Displaying Raw Column Data

Column Sections:

- **Network feature:** Malware tends to communicate **using the Windows** network services to send or receive data off the internet. The feature is essential to provide descriptive analysis. The column is named the “network” feature. It describes the network traffic behavior of malware in terms of application programming interfaces (API) that attempt to open sockets to connect to the network.
- **Static Features**. The static analysis of the portable executable involves the code of a malicious program without being executed. It can be used to identify imported dynamic link library (DLL) being used, encrypted strings, and codes being used. The research attempts to gain preliminary insight into the behavior of malicious programs. The following features are as follows:
  - process_name_exe, process_names_exe_parameters, param_hashvalue, peid_signatures, imagebase, entrypoint, pdbpath, actual_checksum, reported_checksum, osversion, exported_dll_name, exported_dll_name_hash, dllfiles, dllfiles_hash, dllfiles_sorted, dll_sorted_hash, imported_dll

- **Dynamic Features**: Dynamic analysis is the behavioral process or application programming interfaces (API) captured while observing system malware. Its characteristics can dynamically change without the user's knowledge by interacting with the victims' computer systems. The following features are captured as follows: namely, exe_cmcsUnique, procTreeUniqueHash, procTreeUniqueNoneHash, procAPIUnique_hash, procAPIUnique, proc_API_hash, sumFilesUniqueHash, sumFilesUniqueNoDeli2, sumUniqueReadFileStringHash, sumReadFilesStringNoDeli, sumUniqueWriteFilesHash, sumUniqueWriteFiles, sumUniqueDeletedSHash, sumUniqueDeletedS,dropped_files, dropped_files_hash

In conclusion, the data collection takes an extensive period for analysis. It requires domain knowledge of the malware characteristics to represent the data set better. The first step of the evaluation stage is understanding the data before getting ingested into machine learning. The collection phase, in summary, has gone through finding specific malware repositories for samples, ingesting into the CAPEv2 malware sandbox, and converting the JSON output format into comma-delimited files. During the conversation, some features with fewer null values are extracted to represent the data model.

3.3.1 Phase I: Discovery

The collection of data by the CAPEv2 malware sandbox targets static, dynamic, and network run-time output. These are the general categories of multiple data sources during the observation phase that are filtered, parsed, and included in the samples. ABI states, "Intelligence discovery is the ability to select, manipulate, and correlate data from multiple sources to identify information relevant to ongoing operations and requirements "[3]. As a result, the objective is to perform feature engineering, a process to filter data through feature selection, transformation, construction, and extraction. It is vital and the first step to select relevant features of interest from the dataset. The selections portray strongly related features; any weak relations are discarded. Having quality data to analyze gives easy quantification of data patterns and tests the research questions. The following steps of discovery are performed by using feature engineering used for this research project.

Step 1: Data Cleanup: The data source from phase one is the first exploratory selection process to be parsed using cleaned data and extensive feature engineering. Feature engineering (FE) requires the selection and manipulation of the raw data to be transformed into features that need to be ingested into machine learning. The Python DataFrame is a two-dimensional data structure that stores data. It has an extensive library of functions to drop or add column features which are used for data cleanup, which is essential before loading into the machine learning.

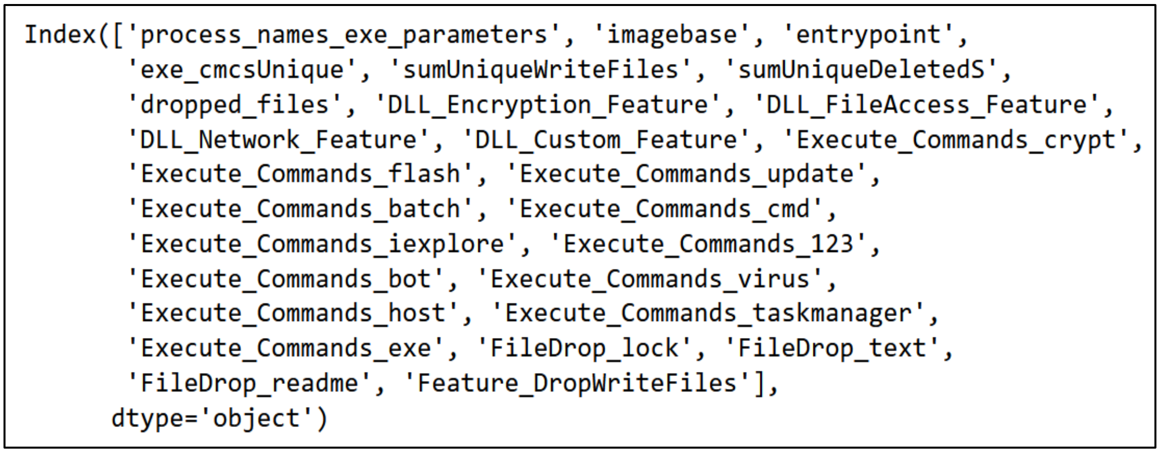
- **Unique Feature**: Any columns with unique features, for example, the generic name “process_name_exe” of the malware, are dropped. The malware repository generates the ransomware being used. It does not specify what type of ransomware.
- **Dropped Null Values**: Null values are designated as unknown or missing data. They are only bits and pieces of information. Several features that were extracted from the raw data have null values. The column features of the sample data are dropped if they have at least 50% of missing values. As part of the domain knowledge required for the project, the belief is that some ransomware features may not have used the features due to changes in functionalities. There are no imputation techniques used to fill in the null data, except having to mark it as zero to indicate that feature is not used.

Step 2: Feature Extraction: The process takes the extracts of features from the current data that are useful, and in turn, another feature is created. As shown below in Table X, the query containing the keyword, for example, ‘Crypt’ on DLL columns, the feature is created having each row whether with the designation of 1(true) or 0 (false). The resulting feature is listed in Table 1.

<table>
<tr><td>DLL Columns</td></tr>
<tr><td>DataFrame['dllfiles'].str.contains('Crypt', case=False, regex=True)
DataFrame['dllfiles'].str.contains('SHELL', case=False, regex=True)
DataFrame['dllfiles'].str.contains('wsock|WS2_32|MSWSOCK|WININET|netapi32|WINHTTP|Mswsock', case=False, regex=True)
DataFrame['dllfiles'].str.contains('sohutool', case=False, regex=True)</td></tr>
<tr><td>Executable</td></tr>
<tr><td>DataFrame ['exe_cmcsUnique'].str.contains('crypt', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('Install', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('update', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('batch', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('cmd', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('iexplore', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('123|321', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('bot', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('virus', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('host', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('task', case=False, regex=True)
DataFrame ['exe_cmcsUnique'].str.contains('exe', case=False, regex=True)</td></tr>
<tr><td>Dropped Files</td></tr>
<tr><td>DataFrame['dropped_files'].str.contains('lock', case=False, regex=True)
DataFrame['dropped_files'].str.contains('text|files', case=False, regex=True)
DataFrame['dropped_files'].str.contains('read', case=False, regex=True)
DataFrame ['sumUniqueWriteFiles'].str.contains('msg|hack|crypt|hello|notice|readme|news|EnCiPhErEd|EnCrYpTeD|ENCODED|lock|password', case=False, regex=True)</td></tr>
<tr><td>API Files</td></tr>
<tr><td>DataFrame ['procAPIUnique'].str.contains('write', case=False, regex=True)
DataFrame ['procAPIUnique'].str.contains('read', case=False, regex=True)
DataFrame ['procAPIUnique'].str.contains('open', case=False, regex=True)
DataFrame ['procAPIUnique'].str.contains('delete', case=False, regex=True)
DataFrame ['procAPIUnique'].str.contains('crypt', case=False, regex=True)
DataFrame ['procAPIUnique'].str.contains('http|socket|connect|send|recv|GetAdaptersAddresses|bind', case=False, regex=True)</td></tr>
</table>

Table 1: Feature Creation/Engineering by Feature Extraction

Step 4: After the exploratory data research, the identified features from the original features and created features with the least number of null values on each column are included as the final features as shown in Table 2.



```
Index(['process_names_exe_parameters', 'imagebase', 'entrypoint',
       'exe_cmcsUnique', 'sumUniqueWriteFiles', 'sumUniqueDeletedS',
       'dropped_files', 'DLL_Encryption_Feature', 'DLL_FileAccess_Feature',
       'DLL_Network_Feature', 'DLL_Custom_Feature', 'Execute_Commands_crypt',
       'Execute_Commands_flash', 'Execute_Commands_update',
       'Execute_Commands_batch', 'Execute_Commands_cmd',
       'Execute_Commands_iexplore', 'Execute_Commands_123',
       'Execute_Commands_bot', 'Execute_Commands_virus',
       'Execute_Commands_host', 'Execute_Commands_taskmanager',
       'Execute_Commands_exe', 'FileDrop_lock', 'FileDrop_text',
       'FileDrop_readme', 'Feature_DropWriteFiles'],
      dtype='object')
```

Table 2: Feature Selection Output

Column definitions identify the usefulness of the data. They would be used as the parameter input for clustering because they are deemed to be meaningful data related to ransomware's general characteristics. Most of the data does not have null values and is mostly related to ransomware's general characteristics. The identified data is the essential and the most granular piece of static, behavioral, and network parameters. They are chosen because of their knowledge of ransomware. As a result, the recognized data are to be included in the process, such as the data described in Table 2.

- **Process names with parameters** (process_names_exe_parameters) are parameters being passed to the malware before they are set to run in the Windows environment. Many programs run differently with different parameters. One must run the Windows DOS command prompt window to accomplish this. Each observation has shown using different parameters.
- **Imagebase** (imagebase) – is the portable executable (PE) hash identifier of the malware to ensure that the collected malware is unique. The value of the variable specifies the preferred address where the Windows executable should be mapped to memory.

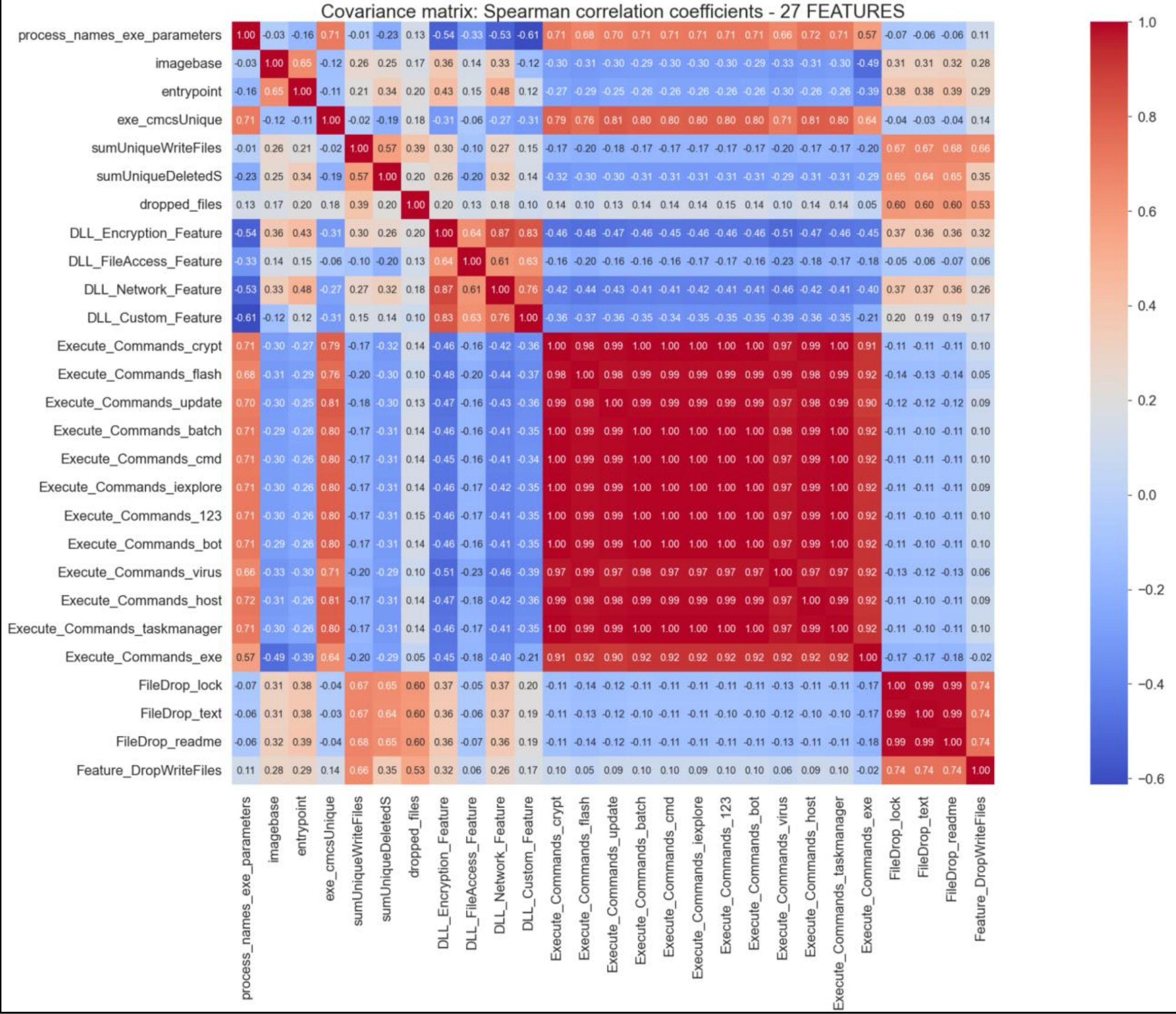
- **Entry Point** (entry_point) is the starting point memory address where malware attempts to execute. For example, the malware starts with 0x004ee3c0 memory to execute. It is relative to the "Imagebase" address. The address of the entry point is the address where the portable executable loader will start its execution.
- **Execute Unique Commands** (exe_cmcsUnique) – the different executable files or commands the malware launches (i.e., update.bat, fondue.exe). These are the executable files run by the malware. An executable file (EXE file) contains an encoded sequence of instructions that computer systems can execute, whether by user clicks or called by the other program.
- **File Manipulation** (sumUniqueWriteFiles, sumUniqueDeletedS) shows files created (written) or deleted by the malware.
- **DLL** (Dynamic Link Library) – the columns DLL_Encryption_Feature, DLL_FileAccess_Feature, DLL_Network_Feature, and DLL_Custom_Feature are the libraries used and shared by many applications running in the Windows systems.
- **Execution Files** (Execute_Commands_crypt,Execute_Commands_flash, Execute_Commands_update, Execute_Commands_batch, Execute_Commands_cmd, Execute_Commands_iexplore, Execute_Commands_123, Execute_Commands_bot, Execute_Commands_virus, Execute_Commands_host, Execute_Commands_taskmanager, Execute_Commands_exe) are the processes executable features as part of feature engineering. These are the most common executable features captured during the observation.
- **Dropped Files** (FileDrop_lock, FileDrop_text, FileDrop_readme, Feature_DropWriteFiles) are the files dropped by the malware in different file extensions. It is either in *.lock, *.text, or *.readme file extensions)

Step 5: Heatmap Correlation

Heatmap correlation, a form of a graphical two-dimensional representation of data, displays between variables as a color-code matrix. The chart shows how the variables are closely related to visualize the strength of relationships between two variables and excluding the variables with low correlation.

Using a Spearman correlation coefficient shows the correlations of each feature in the dataset. Each feature is listed on both axes, and the relationships with other variables are displayed from blue to red color. The correlation ranges from -1.0 (blue) to +1.0 (red). It determines whether there is a linear or nonlinear relationship between variables. In this research, features with more than 50% (.50), from moderate to strong positive correlation, will be the target variables as feature selections. There are 1300 observations (samples) with twenty-seven identified features, the data variables from each sample. Figure I, the Heatmap, displays the value in each cell used to gauge the strength of the relationship and the direction of the relationship between the two variables. That is, each feature is listed on both axes. The correlation could be positive or negative (Weak +/-0.0-0.40, Moderate +/-0.40-0.70, Strong +/-0.70-1.0). As the color becomes darker in red, they are more highly correlated. The heatmap values are changed to the standard scale from -1 to 1.

The closer to the 1.00 value, the higher the correlation; it is said to have a positive relationship between the two variables. Any value with -1.00 is said to have a negative relationship, and any values with 0 are said not to correlate. The matrix's diagonal elements contain the variables' variances, while the off-diagonal elements contain the covariances between all possible variables. The covariances shown in Figure I indicate the correlations between variables. The figure has also demonstrated the existence of multicollinearity. That is when features, the input variables, correlate highly with one or more of the other features. Having a high correlation affects the performance of any classification or regression model because it skews the output, but the research uses clustering modeling. The dataset is ingested into Principal Component Analysis (PCA) to combine all the highly correlated variables into an uncorrelated variable. Then, the PCA output is used as input for DBSCAN. Theoretically, one could apply PCA to the samples or the features (dimensions). This research uses features because they are smaller than the samples.



Covariance matrix: Spearman correlation coefficients - 27 FEATURES

	process_names_exe_parameters	imagebase	entrypoint	exe_cmcsUnique	sumUniqueWriteFiles	sumUniqueDeletedS	dropped_files	DLL_Encryption_Feature	DLL_FileAccess_Feature	DLL_Network_Feature	DLL_Custom_Feature	Execute_Commands_crypt	Execute_Commands_flash	Execute_Commands_update	Execute_Commands_batch	Execute_Commands_cmd	Execute_Commands_iexplore	Execute_Commands_123	Execute_Commands_bot	Execute_Commands_virus	Execute_Commands_host	Execute_Commands_taskmanager	Execute_Commands_exe	FileDrop_lock	FileDrop_text	FileDrop_readme	Feature_DropWriteFiles
process_names_exe_parameters	1.00	-0.03	-0.16	0.71	-0.01	-0.23	0.13	-0.54	-0.33	-0.53	-0.61	0.71	0.68	0.70	0.71	0.71	0.71	0.71	0.71	0.66	0.72	0.71	0.57	-0.07	-0.06	-0.06	0.11
imagebase	-0.03	1.00	0.65	-0.12	0.26	0.25	0.17	0.36	0.14	0.33	-0.12	-0.30	-0.31	-0.30	-0.29	-0.30	-0.30	-0.30	-0.29	-0.33	-0.31	-0.30	-0.49	0.31	0.31	0.32	0.28
entrypoint	-0.16	0.65	1.00	-0.11	0.21	0.34	0.20	0.43	0.15	0.48	0.12	-0.27	-0.29	-0.25	-0.26	-0.26	-0.26	-0.26	-0.26	-0.30	-0.26	-0.26	-0.39	0.38	0.38	0.39	0.29
exe_cmcsUnique	0.71	-0.12	-0.11	1.00	-0.02	-0.19	0.18	-0.31	-0.06	-0.27	-0.31	0.79	0.76	0.81	0.80	0.80	0.80	0.80	0.80	0.71	0.81	0.80	0.64	-0.04	-0.03	-0.04	0.14
sumUniqueWriteFiles	-0.01	0.26	0.21	-0.02	1.00	0.57	0.39	0.30	-0.10	0.27	0.15	-0.17	-0.20	-0.18	-0.17	-0.17	-0.17	-0.17	-0.17	-0.20	-0.17	-0.17	-0.20	0.67	0.67	0.68	0.66
sumUniqueDeletedS	-0.23	0.25	0.34	-0.19	0.57	1.00	0.20	0.26	-0.20	0.32	0.14	-0.32	-0.30	-0.30	-0.31	-0.31	-0.31	-0.31	-0.31	-0.29	-0.31	-0.31	-0.29	0.65	0.64	0.65	0.35
dropped_files	0.13	0.17	0.20	0.18	0.39	0.20	1.00	0.20	0.13	0.18	0.10	0.14	0.10	0.13	0.14	0.14	0.14	0.15	0.14	0.10	0.14	0.14	0.05	0.60	0.60	0.60	0.53
DLL_Encryption_Feature	-0.54	0.36	0.43	-0.31	0.30	0.26	0.20	1.00	0.64	0.87	0.83	-0.46	-0.48	-0.47	-0.46	-0.45	-0.46	-0.46	-0.46	-0.51	-0.47	-0.46	-0.45	0.37	0.36	0.36	0.32
DLL_FileAccess_Feature	-0.33	0.14	0.15	-0.06	-0.10	-0.20	0.13	0.64	1.00	0.61	0.63	-0.16	-0.20	-0.16	-0.16	-0.16	-0.17	-0.17	-0.16	-0.23	-0.18	-0.17	-0.18	-0.05	-0.06	-0.07	0.06
DLL_Network_Feature	-0.53	0.33	0.48	-0.27	0.27	0.32	0.18	0.87	0.61	1.00	0.76	-0.42	-0.44	-0.43	-0.41	-0.41	-0.42	-0.41	-0.41	-0.46	-0.42	-0.41	-0.40	0.37	0.37	0.36	0.26
DLL_Custom_Feature	-0.61	-0.12	0.12	-0.31	0.15	0.14	0.10	0.83	0.63	0.76	1.00	-0.36	-0.37	-0.36	-0.35	-0.34	-0.35	-0.35	-0.35	-0.39	-0.36	-0.35	-0.21	0.20	0.19	0.19	0.17
Execute_Commands_crypt	0.71	-0.30	-0.27	0.79	-0.17	-0.32	0.14	-0.46	-0.16	-0.42	-0.36	1.00	0.98	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.91	-0.11	-0.11	-0.11	0.10
Execute_Commands_flash	0.68	-0.31	-0.29	0.76	-0.20	-0.30	0.10	-0.48	-0.20	-0.44	-0.37	0.98	1.00	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.92	-0.14	-0.13	-0.14	0.05
Execute_Commands_update	0.70	-0.30	-0.25	0.81	-0.18	-0.30	0.13	-0.47	-0.16	-0.43	-0.36	0.99	0.98	1.00	0.99	0.99	0.99	0.99	0.99	0.97	0.98	0.99	0.90	-0.12	-0.12	-0.12	0.09
Execute_Commands_batch	0.71	-0.29	-0.26	0.80	-0.17	-0.31	0.14	-0.46	-0.16	-0.41	-0.35	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.98	0.99	1.00	0.92	-0.11	-0.10	-0.11	0.10
Execute_Commands_cmd	0.71	-0.30	-0.26	0.80	-0.17	-0.31	0.14	-0.45	-0.16	-0.41	-0.34	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.92	-0.11	-0.11	-0.11	0.10
Execute_Commands_iexplore	0.71	-0.30	-0.26	0.80	-0.17	-0.31	0.14	-0.46	-0.17	-0.42	-0.35	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.92	-0.11	-0.11	-0.11	0.09
Execute_Commands_123	0.71	-0.30	-0.26	0.80	-0.17	-0.31	0.15	-0.46	-0.17	-0.41	-0.35	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.92	-0.11	-0.10	-0.11	0.10
Execute_Commands_bot	0.71	-0.29	-0.26	0.80	-0.17	-0.31	0.14	-0.46	-0.16	-0.41	-0.35	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.92	-0.11	-0.10	-0.11	0.10
Execute_Commands_virus	0.66	-0.33	-0.30	0.71	-0.20	-0.29	0.10	-0.51	-0.23	-0.46	-0.39	0.97	0.99	0.97	0.98	0.97	0.97	0.97	0.97	1.00	0.97	0.97	0.92	-0.13	-0.12	-0.13	0.06
Execute_Commands_host	0.72	-0.31	-0.26	0.81	-0.17	-0.31	0.14	-0.47	-0.18	-0.42	-0.36	0.99	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.97	1.00	0.99	0.92	-0.11	-0.10	-0.11	0.09
Execute_Commands_taskmanager	0.71	-0.30	-0.26	0.80	-0.17	-0.31	0.14	-0.46	-0.17	-0.41	-0.35	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	0.92	-0.11	-0.10	-0.11	0.10
Execute_Commands_exe	0.57	-0.49	-0.39	0.64	-0.20	-0.29	0.05	-0.45	-0.18	-0.40	-0.21	0.91	0.92	0.90	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	1.00	-0.17	-0.17	-0.18	-0.02
FileDrop_lock	-0.07	0.31	0.38	-0.04	0.67	0.65	0.60	0.37	-0.05	0.37	0.20	-0.11	-0.14	-0.12	-0.11	-0.11	-0.11	-0.11	-0.11	-0.13	-0.11	-0.11	-0.17	1.00	0.99	0.99	0.74
FileDrop_text	-0.06	0.31	0.38	-0.03	0.67	0.64	0.60	0.36	-0.06	0.37	0.19	-0.11	-0.13	-0.12	-0.10	-0.11	-0.11	-0.10	-0.10	-0.12	-0.10	-0.10	-0.17	0.99	1.00	0.99	0.74
FileDrop_readme	-0.06	0.32	0.39	-0.04	0.68	0.65	0.60	0.36	-0.07	0.36	0.19	-0.11	-0.14	-0.12	-0.11	-0.11	-0.11	-0.11	-0.11	-0.13	-0.11	-0.11	-0.18	0.99	0.99	1.00	0.74
Feature_DropWriteFiles	0.11	0.28	0.29	0.14	0.66	0.35	0.53	0.32	0.06	0.26	0.17	0.10	0.05	0.09	0.10	0.10	0.09	0.10	0.10	0.06	0.09	0.10	-0.02	0.74	0.74	0.74	1.00

Figure I: Heatmap (Multicollinearity)

In conclusion, Figure I displays the summary of all correlations between all the possible pairs of variables. It has shown many multi-collinearity. Any paired variables (positive or negative pairs) that decrease or increase together are identified as correlated. The heatmap shown under study is the primary variable of interest across two axes. The closer the value to 1.0, the higher the correlation. Since the heat map contains multiple dimensions, Principal Component Analysis (PCA) is used to reduce the dimensionality of column features to the next phase of ABI.

3.3.2 Phase II: Assessment

The assessment starts with categorizing the target data by focusing on and drilling down each malware, the ransomware in question, by examining its capabilities and features. It detects any modifications and groups all the malware similar to the other malware. As stated by Atwood, "*Intelligence assessment* is the ability to provide a focused examination of data and information about an object or an event, to classify and categorize it, and to assess its reliability and credibility in order to create estimates of capabilities and impacts." [3]. Examining data requires looking into the purpose of each feature to be included in the study as part of the feature engineering phase. There are many features to sift through, with at least 27 features or dimensions. Consequently, a Principal Component Analysis (PCA) is used to reduce the dimensions followed by Unsupervised Density-based spatial clustering of application with noise (DBSCAN), which is precomputed by running a Silhouette Score.

3.3.2.1 Principal Component Analysis [Step 1]

The first step requires studying and parsing the combined three sources (static, behavioral, network) for feature engineering. CAPEv2 Sandbox provided data set behaviors after ingesting a large malware sample. The associated datasets (selected 27 features) are reduced or condensed while preserving meaningful data. Principal Component Analysis (PCA), an unsupervised learning method, is one technique used to reduce the features into a smaller set of new composite dimensions. PCA is used to explore data for analysis, giving us an excellent data summary using a limited number of principal components. Reducing our dataset dimensions can find new variables from the original datasets and solve eigenvalue and eigenvector problems. From Figure, I, any identified features with 0.50 correlations are included. Having to analyze 27 variables is too large for helpful analysis; anything beyond the three dimensions is difficult to understand. As a result, the principal component analysis is configured to output from 27 original features to a reduced three feature variables of a data set while preserving as much information as possible.

Using the Scikit-learn PCA(), part of the machine learning library for Python programming language, there is a way to project the best number of principal components; it is a type of hyperparameter tuning process selecting the optimal value for the hyperparameter n_components variable. In other words, we select the smallest number of components that hold at least 80% of the total variance, the recommended value Watkins [8]. In addition, based on the interpretation of Cangelosi, PCA should keep the most variance between 80% and 90% for straightforward interpretation [61]. Figure I, a scree plot useful visual aid representation for determining the number of principal components, shows *"explained variance"* across components and informs about an individual and cumulative explained variance for each component. It is a graphical representation of the variation of each principal component. The explained variance ratio is the percentage of variance explained by each selected component. The number of components to include in the model is adding the explained variance ratio of each component until we reach at least 80% percent to avoid overfitting.

```
Number of Features:  27
How many components algorithm has selected:  5
Total Variance for each bar:  [0.52294824 0.20165593 0.09273911 0.05369307 0.02995287]
Cumulated In Progression:  [0.52294824 0.72460417 0.81734328 0.87103635 0.90098922]
Total Variance Explained: 90.1
(1147, 5)
```

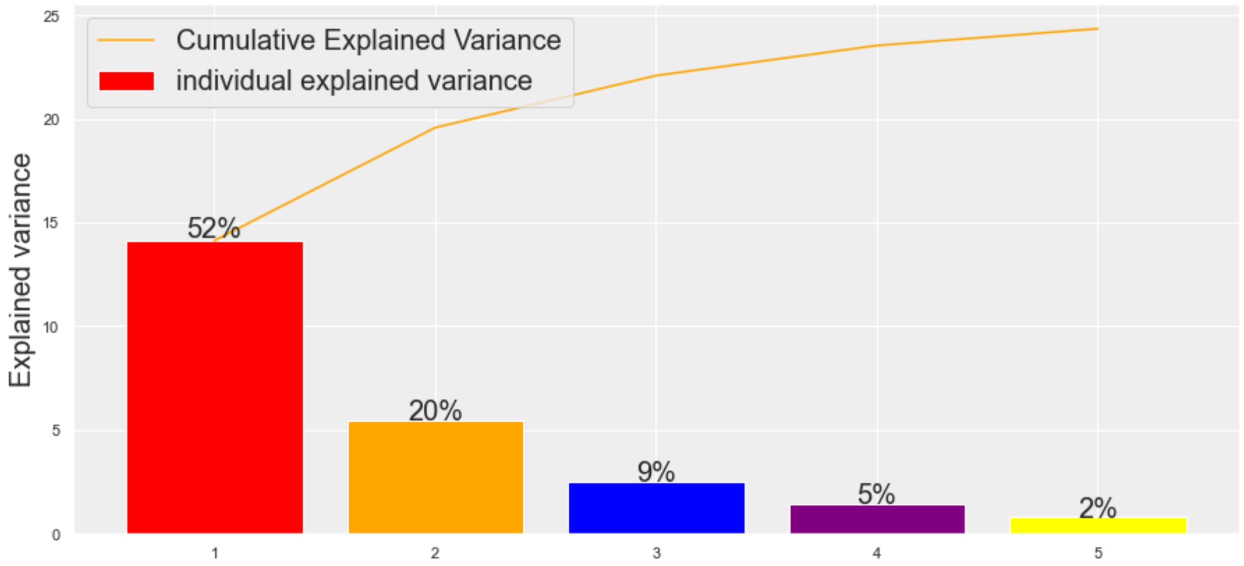


Figure J: 81.73 Cumulative Variance

The first three bar graphs, the eigenvalues, shows PCA 1 [red], PCA 2 [orange], and PCA 3 [blue]) are added to 81.73%. Added all the projected PCAs, the Total variance is the sum of all variances of individual principal components. The values, 0.52294824 0.20165593 0.09273911], equal the eigenvalues of the covariance; it's stored in "PCA.explained_variance_." These 3 PCAs are used as the cut-off value for the 80% threshold as shown in the curved line (a cumulative explained variance) – the dimensionality reduction applications represent all the data features. PCA 1 has the most considerable explained variance. It is the most significant absolute value contributing to more specific features; it accounts for 50% of the variation. As a result, we have twenty-seven features reduced to 3, the PCA(s); they would be used to represent the data accurately. An alternative method of showing Figure J is the Scree Plot Figure K. It shows the explained variance ratio plot. The screen plot below shows the eigenvalues from the largest to the smallest. The ideal or acceptable pattern for PCA is the steeped curve, followed by a bend and straight line. In this case, PCA 3 is chosen.

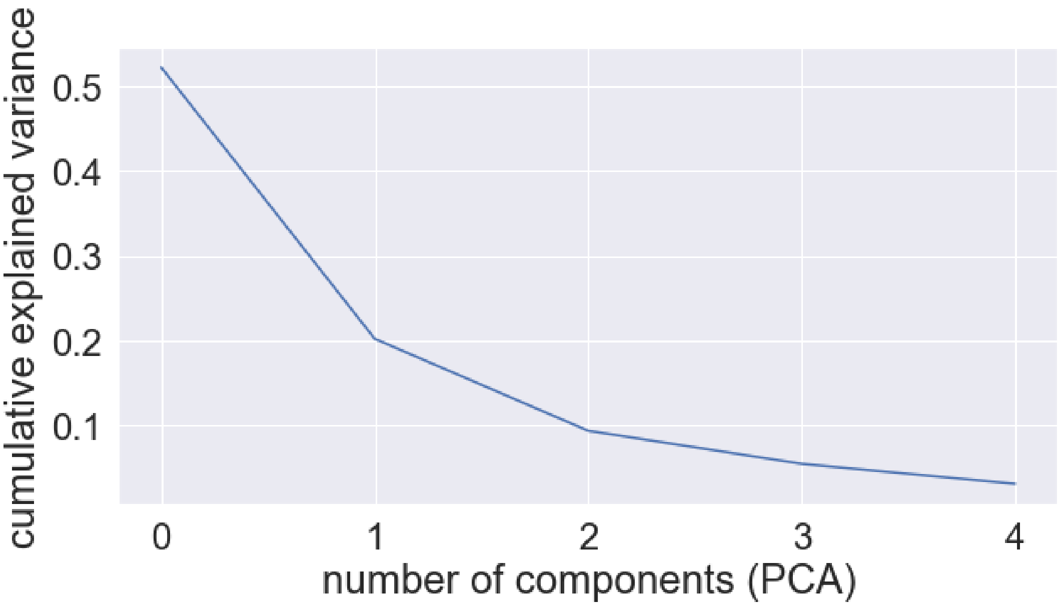


Figure K: Project Principal Component Analysis Alternative View

3.2.2.2 Silhouette Scoring and DBSCAN [Step 2]

The purpose of this phase is to categorize the data using the PCA as the input. It is ingested by the Unsupervised Density-based spatial clustering of application with noise (**DBSCAN)** clustering algorithm. The collected ransomware data is **unlabeled data** clustering using the **Euclidean** distance measurement. The DBSCAN has to make up new labels for the data based on what it sees. The clusters give the baselines to show similarities among malware of the same type deployed by different authors. Finding clusters of data signifies an association of the malware that most likely comes from the exact origin of the source code, which the malware author may share. The more they are similar, the more they belong to a group or a cluster. "Clustering is partitioning the data into groups that are similar as possible given a set of data Objects," as stated by Bushra and G. Yi [9]. It would be highly effective and beneficial to use DBSCAN to improve clustering quality. With thousands of malware samples collected, it would tell us the number of baselines or commonalities, giving us a possible clue that they may have derived from the source code origin.

The graph in Figure L, the Image before applying DBSCAN, derives from Figure K Principal Component Analysis (PCA), where feature engineering was used to reduce the 27 features. Section 3.2.2.2.1 shows to calculate the silhouette scoring, which would be applied to DBSCAN as shown in Figure K of section 3.2.2.2.2.

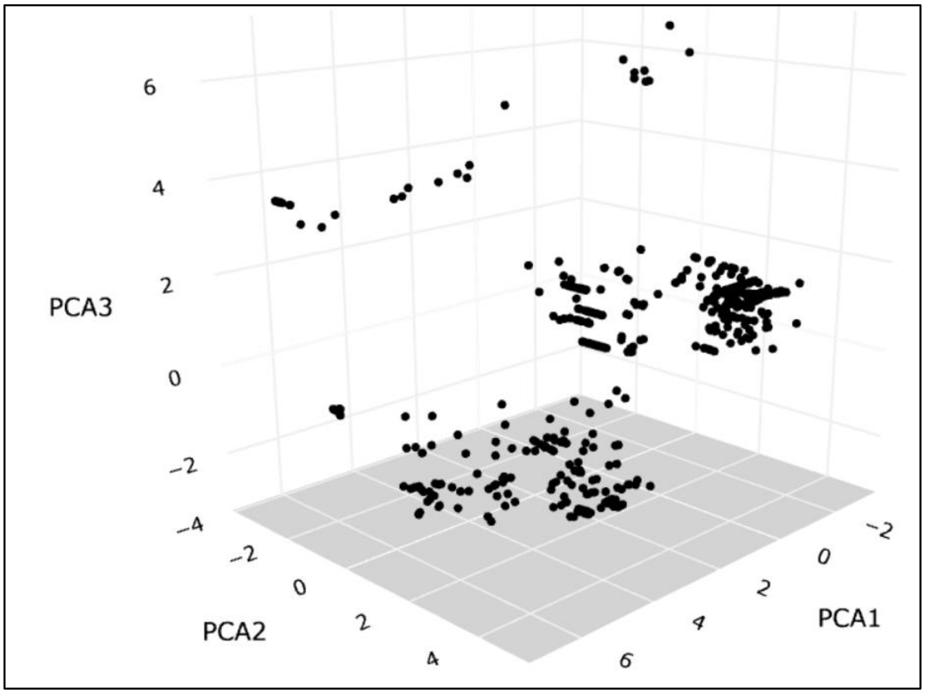
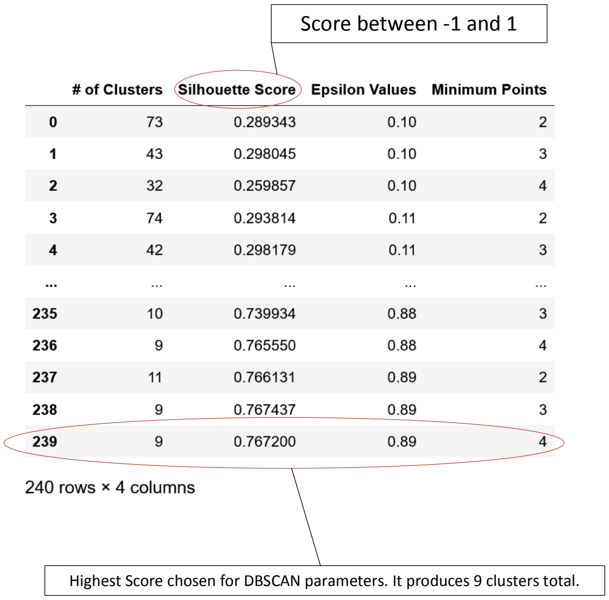


Figure L: PCA Graph Feature Engineering

3.2.2.2.1 Calculating Silhouette Score Technique (Internal Measures)

Two measuring methods are introduced to validate the accuracy of Silhouette scoring techniques, which are used to find the maximum number of clusters in the dataset under study. The first is the Silhouette Scoring technique, a graphical representation technique to measure cluster separability. It is a metric for how good the clustering or how well each object has been classified. It indicates how many clusters are in a given dataset. The scoring value ranges from -1 (low value) to 1 (high value), as shown in Table 3 Silhouette Score column. It demonstrates that the high value indicates how similar or cohesive an object is to its own cluster compared to other separated clusters. The second one is the elbow technique used to determine the appropriate epsilon. They are used side by side to find and compare the optimal clusters.

Seeing Table 3 shows the iteration of epsilon values ranging. The scoring metrics below determine the appropriate epsilon level and the minimum number of points to get the correct clusters. The epsilon values are iterated, ranging from 0.10 to 0.89. Looking at the table below, the best epsilon value and the minimum are 0.89 and 4, respectively; it generates 8 clusters plus outliers totaling 9. **The data came from line # 239 below, showing one of the optimal numbers of clusters; it is the simulated precomputed value of distance metrics;** it gives the least number of outliers. Only Silhouette Score is used for measurement because Inertia Score is only to be used with DBSCAN with a spherical shape; it is not applicable at this point.



Score between -1 and 1

	# of Clusters	Silhouette Score	Epsilon Values	Minimum Points
0	73	0.289343	0.10	2
1	43	0.298045	0.10	3
2	32	0.259857	0.10	4
3	74	0.293814	0.11	2
4	42	0.298179	0.11	3
...	...	...	...	...
235	10	0.739934	0.88	3
236	9	0.765550	0.88	4
237	11	0.766131	0.89	2
238	9	0.767437	0.89	3
239	9	0.767200	0.89	4

240 rows × 4 columns

Highest Score chosen for DBSCAN parameters. It produces 9 clusters total.

Table 3: Silhouette Score

There are several approaches or techniques to visualize Silhouette Scoring to determine the appropriate epsilon values and minimum points by using the elbow approach. The research found 0.89 epsilon values and four minimum points is a table representation to the graphical Figure M as shown below (8|4), using different level hyperparameters. The silhouette score is 0.790138 meaning the clusters are well apart from each other as the silhouette scoring is closer to 1.

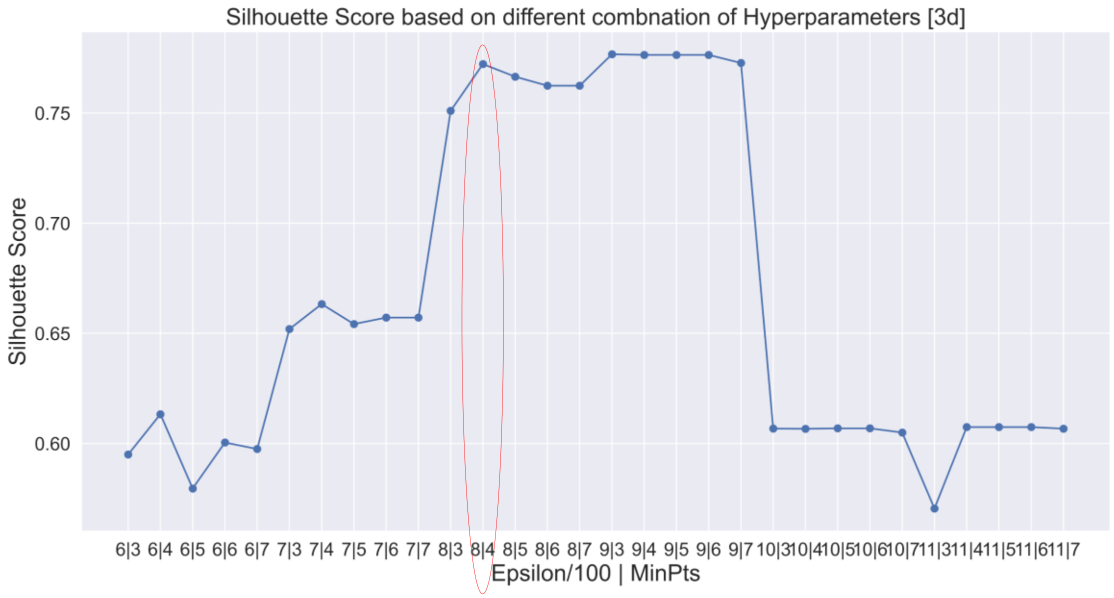


Figure M: Silhouette Scoring using Hyperparameters

Finally, Silhouette scoring is compared to the elbow method, as shown in Figure N. The goal is to find the elbow in the plot, which shows the point where the number of clusters increases, shown in a circle. Comparing this to the Silhoutette scoring, it is determined that the epsilon is around .80. That entails that the scoring is measurably accurate.

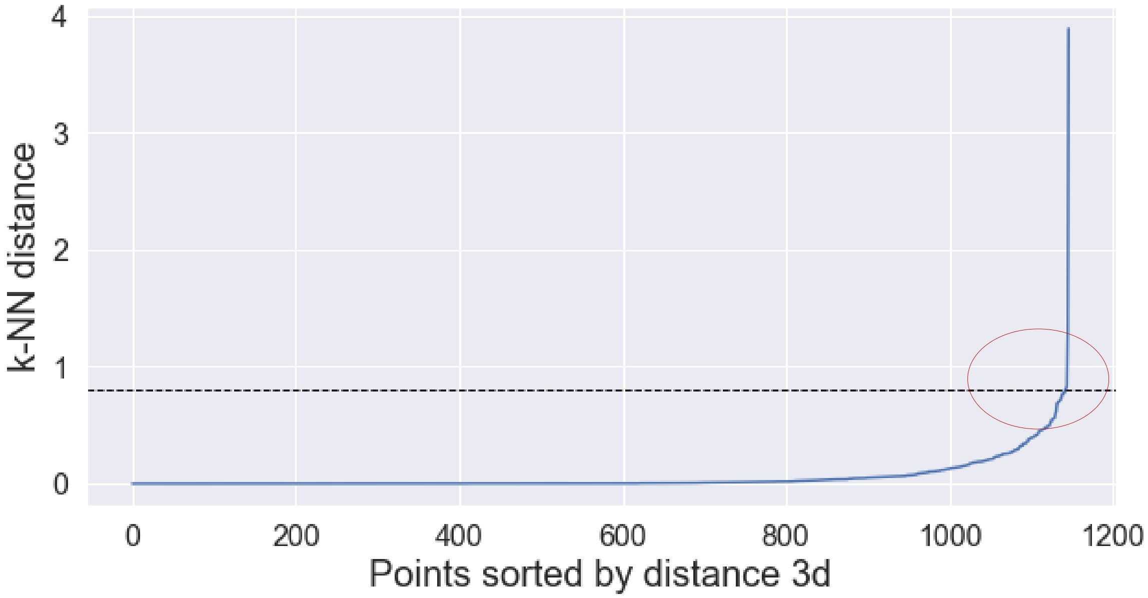


Figure N: Elbow Technique To Find Optimum Epsilon

3.2.2.2.2 Applying Silhouette Scoring to DBSCAN

DBSCAN is a density-based clustering algorithm in which dense regions in space are separated from other regions by lower density. Using the clusters would show some degrees of similarities among ransomware or any malware. The advantage of the DBSCAN machine learning algorithm is that it does not require specifying the number of clusters in the dataset. Using this algorithm requires two parameters, namely epsilon (eps) and minimum points (minPoints), which could be calculated from Figure M and Table 3 Silhouette scoring. Epsilon is the radius of the circle, a close point that should be considered as part of the cluster. The minimum (minPoints) is the number of data points required inside of the circle to form a dense region; for instance, if the parameter of the minPoints is set to 4, then the algorithm needs at least 4 points to form a dense region.

Before applying the clustering algorithm, one has to determine the correct epsilon level and minimum points. Using Table 3 shows the simulated combination of it, resulting in the number of clustering output results. Since the project requires three principal components, Figure O, the DBSCAN results, shows that after running DBSCAN using epsilon .89 and a minimum of 4 points precomputed values with the default metric parameter of "Euclidean," which calculates the distance between instances in a feature array. Each object clustered within the same group is more similar than those in the other groups. The diagram below shows eight clustered groups; there are 14 individual outliers, data points that do not belong to any groups. The resulting clusters are calculated using Silhouette Coefficient only because the malware data [ransomware] are not labeled data. Note that the collected malware is unlabeled, rendering the truth labels unknown. As a result, the only evaluation for DBSCAN is the model results; in this case, the Silhouette Coefficient is the only metric to use, which works well in classifying clusters resulting in globular clusters. The typical metrics identified by sci-kit-learn DBSCAN, such as Homogeneity, Completeness, V-measure, Adjusted Rand Index, and Adjusted Mutual Information, could only be used if the DBSCAN is used with "true labels." In addition, Density-based Clustering Validation (DBCV) is unnecessary because it only applies to non-globular clusters.

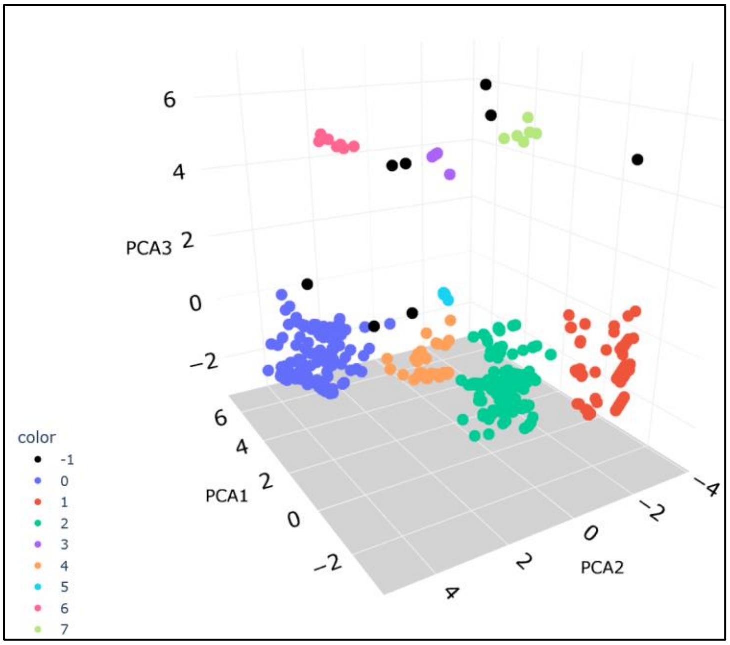


Figure O: DBSCAN using Silhouette Coefficient

The clustering results portrayed in Figure N (Cluster Cardinality) have followed the segmentation process of using Silhouette scoring metrics as the input data for DBSCAN, which uses Euclidean distance metrics. Choosing the correct features with the correct number of PCAs resulted in showing no overlapping clusters with the least number of outliers, which is 8 (-1 label on the x-axis). The clustering data shows the similarities of the internal characteristics. According to Li, "Clustering analysis refers to the analysis process of grouping a set of physical or abstract objects into multiple classes composed of similar objects. Its goal is to classify the data according to the similarity of the data's internal characteristics and reveal the data's internal natural structure. In short, clustering refers to grouping abstract objects or physical object sets so that the similarity of objects in a group is large. In contrast, the difference between different groups is large "[63]. Under certain circumstances, Figure P shows the cardinality where each cluster shows a degree of similarity compared to other ransomware belonging to other clusters.

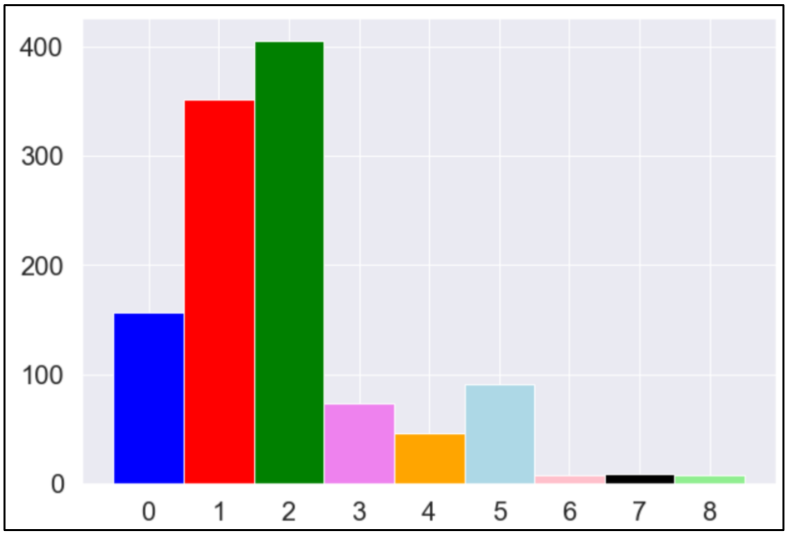


Figure P: Clustering Cardinality

3.3.3 Phase III: Explanation (Descriptive Data Analysis)

The descriptive analysis investigates or describes the summary of the data points in such a way as to show patterns and insights into the static, behavioral, and network data. It is one of the essential steps before conducting statistical because it substantially affects data analysis or predictive analysis for their completeness. It gives the insight distribution of data to detect similarities, differences, or outliers to identify associations among the features captured during the observation phase of the malware behavior. Under certain circumstances, bar graphs, table data, line graphs, or supervene diagrams will be used to represent the collective statistics for comparison.

Using the results of the DBSCAN, where eight (8) clustered observations are identified, this phase's task is to drill down the examinations of each cluster by identifying the baselines of each significant feature. It includes the similarities, differences, and anomalies of the ransomware features using descriptive statistics regarding its frequencies, percentage, and mode summary. They are essential because they provide absolute numbers that map to the Charts and Graphs of our analysis for the intelligence explanation as stated, "*Intelligence explanation* is the ability to examine events and derive knowledge and insights from interrelated data in order to create causal descriptions and propose significance in greater contexts" [3]. The data provides a descriptive broader narrative for the research questions.

Ransomware is a kind of malware that is designed to deny users or organizations access to files on their computer systems. It encrypts files and demands a ransom payment for the decryption keys. The known domain knowledge of ransomware characteristics is essential to target the statistical analysis of Windows Portable Executable (PE) encryption, file application programming interface (API), and dropped files. Thus, the description of the research focuses on profiling these features to provide descriptive statistics, precise API frequencies, and sequence API analyses to answer the research questions quantitatively. Below are the descriptive statistical data gathered from the ransomware traits, along with the interpretations of each line graph and by using a supervenn diagram [66]. Line graphs for each cluster are used to compare trends and patterns of the features used by the ransomware author. It helps to decipher connections between clusters. The statistical descriptions show the step of the drill-down data that makes up the APIs responsible for file access, like dropping and deleting files and data encryption usage. The following drill-down data delves into different categories of ransomware characteristics from static, dynamic, and network behavior that reveals its method of operations for statistical usage, plus a short summary of the directional data results.

Drill Down Data: General Features (Comparison of captured 27 features)

Application Programming Interfaces (API), text documents, and command executions are the most prevalent behaviors captured during the observation phase during ransomware runs. These features are extracted and transformed from raw data in JSON format of CAPEv2 malware sandbox. The chosen observations derive from the feature discovery process of using domain knowledge. All these features are used to improve the quality of the results from data analysis as the result of machine learning. Figure O describes the up and down slopes of the line segments of each cluster; it shows the ransomware features' changes, trends, and patterns. The line graphs describe the number of elements (static, dynamic, and network categories) that are mainly used for all the ransomware. The label features derive from Figure I of Feature Engineering (FE), showing the general trend differences and similarities.

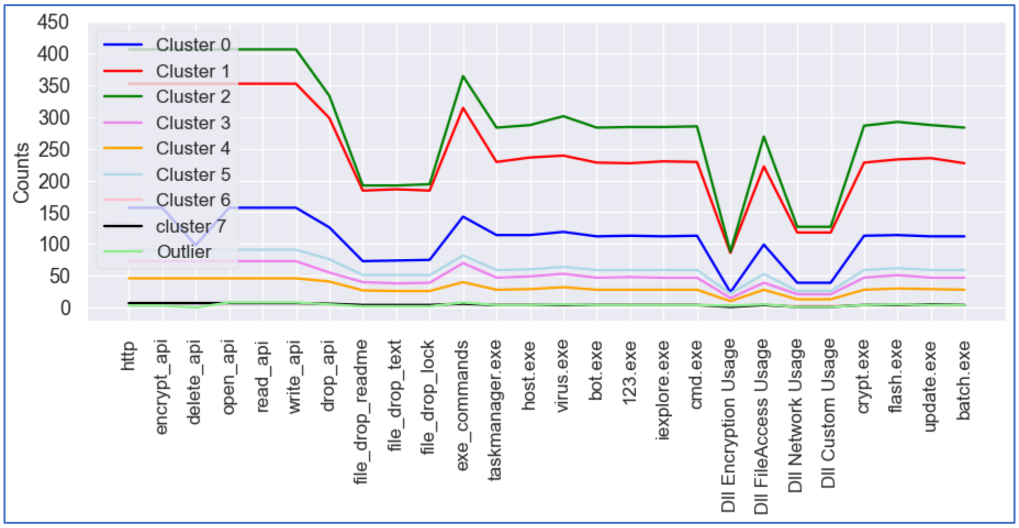


Figure Q: Processes/Files/Static/Dynamic Features

As defined by the Artificial-Based Intelligence (ABI) quantitative analysis, Cluster 0 (zero) **deleted "API"** functions that are less used than the rest of the clusters. It does not conform to the pattern set forth by other groups. This research discovered that all the samples might have been derived from an identical source copy of the malware. Figure M line graph derives from the statistical number of occurrences from Table 3. Lines 1, 3, 4, and 5 have the most events signifying that most activities of reading, opening, encrypting, and writing files.

	General Features	**Cl0**	**CL1**	**CL2**	**CL3**	**CL4**	**CL5**	**CL6**	**CL7**	**Outlier**
0	http	157	352	406	73	46	91	7	7	3
1	encrypt_api	157	352	406	73	46	91	7	7	3
2	delete_api	98	352	406	73	46	91	7	7	0
3	open_api	157	352	406	73	46	91	7	7	8
4	read_api	157	352	406	73	46	91	7	7	8
5	write_api	157	352	406	73	46	91	7	7	8
6	drop_api	126	298	333	55	41	76	5	6	5
7	file_drop_readme	73	184	192	40	27	51	4	4	2
8	file_drop_text	74	186	192	38	26	51	4	4	2
9	file_drop_lock	75	184	194	39	26	51	4	4	2
10	exe_commands	143	314	364	70	40	82	7	6	8
11	taskmanager.exe	114	229	283	47	28	59	4	4	4
12	host.exe	114	236	287	49	29	60	4	4	4
13	virus.exe	119	239	301	53	32	64	5	4	5
14	bot.exe	112	228	283	47	28	59	4	4	4
15	123.exe	113	227	284	48	28	59	4	4	4
16	iexplore.exe	112	230	284	47	28	59	4	4	4
17	cmd.exe	113	229	285	47	28	59	4	4	4
18	Dll Encryption Usage	24	86	88	15	10	22	3	1	3
19	Dll FileAccess Usage	99	222	269	39	28	53	4	4	5
20	Dll Network Usage	39	118	127	21	13	26	3	1	1
21	Dll Custom Usage	39	118	127	21	13	26	3	1	1
22	crypt.exe	113	228	286	47	28	59	4	4	4
23	flash.exe	114	233	292	51	30	62	5	4	5
24	update.exe	112	235	287	47	29	59	4	5	4
25	batch.exe	112	227	283	47	28	59	4	4	0

Table 4: General Features Statistical Occurens Per Cluster

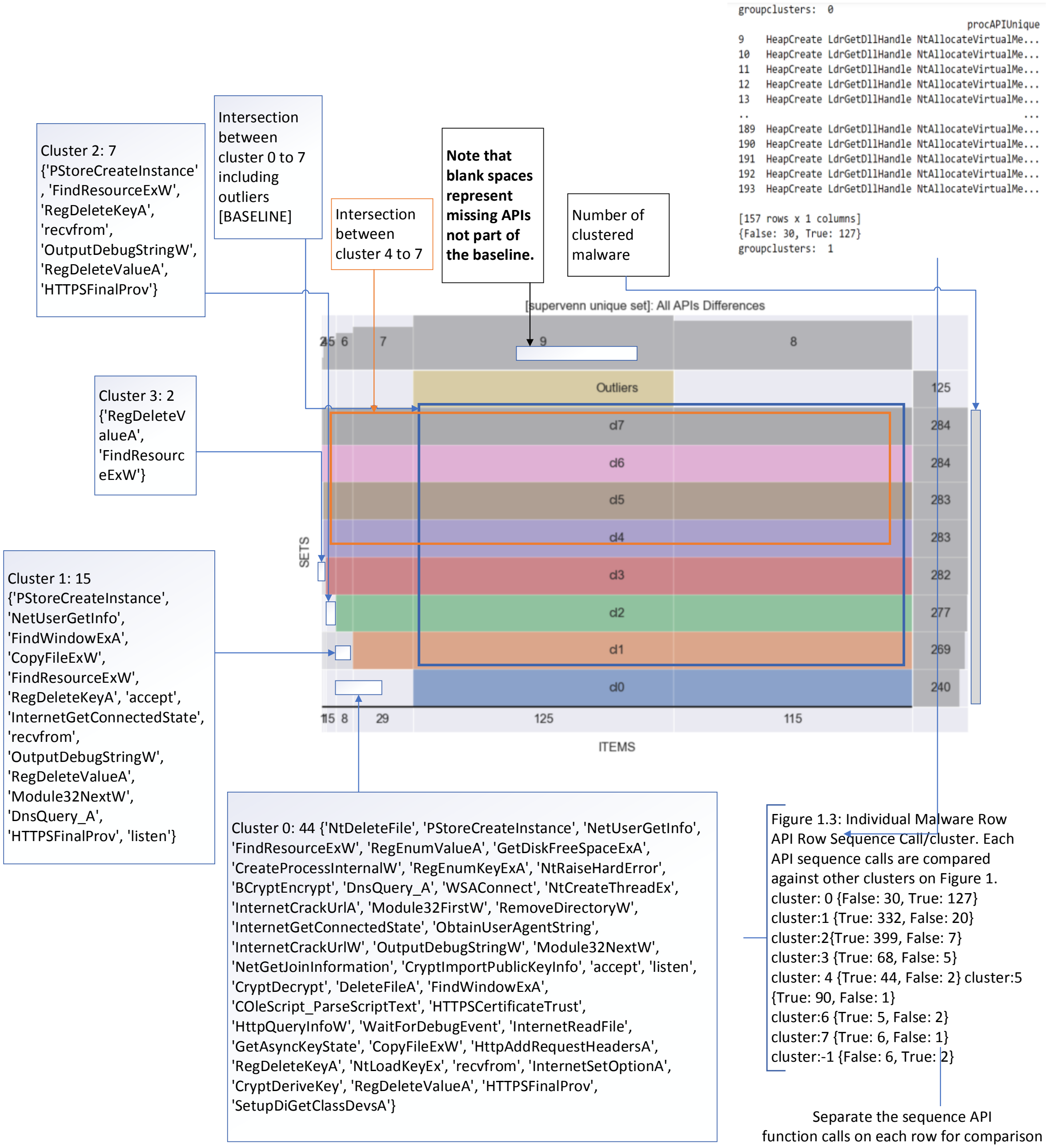
Table 5 shows the definitions and description of the malware activities and the executive summary of the static, network, and dynamic processes captured during the observation. It emphasizes executable, HTTP, file API, DLL, and dropped files.

Label	Description
EXE	The features with *.exe are the system's captured processes. They are the most common portable executables launched by the ransomware after they are deployed in Windows 10 environment.
HTTP	A hypertext transfer protocol captures features during the scan; however, there is hardly any Internet Protocol (IP). The network column was excluded from further analysis due to 90% of the null values.
STATIC ENCRYPTION/FILE	Encryption and file access are the most used DLL features in static analysis.
DYNAMIC ENCRYPTION/FILE	There are several APIs captured by CAPEv2 virtual machine; there are delete, encrypt, read, write, and drop Application Programming Interface (API). "crypt" is a utility program used for encryption. Notice that encrypts_api is a calling function that the utility program might have called.
DROPPED FILES	The most common dropped files have an extension of *.txt, *. readme, and *.lock. They carry messages for the victims to read and instructions.

Table 5: Figure O Label Executive Summary

Drill Down Data: All Invoked APIs [General Comparison] By Cluster

One of the most representative characteristics of malware behavior in detecting malware is the Application Programming Interface (API). It reveals the most intrinsic behavior of the malware motive of operations and its sequence behavior. It’s the way a program interacts with one another, either with the built-in Windows API or the customized API. API calls for a message sent to service the request asking other APIs to provide any services or information. Figure N depicts a general overview of all API calls. Since it is impossible to create more than three sets of Venn diagrams, "**supervenn"** is used to depict the general relationship of each cluster. The orange color is the "Intersection between clusters 4 and 7", which is the final baseline for all ransomware API calls currently; they have identical unique API calls. Cluster zero to cluster three differs from clusters 4 to 7, the total baseline calls, some of which do not exist in some clusters. There are multiple calls, yet the diagram represents only the class APIs; they do not convey the number of calls because it would not fit into the diagram. Reading the "**supervenn**" diagram, all the white spaces are APIs not included in the baseline APIs. For example, "Cluster 1" has 15 API calls not included in the baseline. This diagram will not be discussed in detail, but we will drill down on it in the following section. Figure R shows individuals that have similar characteristics. "True" means the rows are the same, while the "False" findings have no matching rows. It could be inferred that cluster 0 has the least API calls, which might have been the initial baseline.



```
groupclusters:  0
                                          procAPIUnique
9    HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
10   HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
11   HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
12   HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
13   HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
..                                                ...
189  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
190  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
191  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
192  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...
193  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...

[157 rows x 1 columns]
{False: 30, True: 127}
groupclusters:  1
```

Cluster 2: 7 {'PStoreCreateInstance', 'FindResourceExW', 'RegDeleteKeyA', 'recvfrom', 'OutputDebugStringW', 'RegDeleteValueA', 'HTTPSFinalProv'}

Intersection between cluster 0 to 7 including outliers [BASELINE]

Intersection between cluster 4 to 7

Note that blank spaces represent missing APIs not part of the baseline.

Number of clustered malware

Cluster 3: 2 {'RegDeleteValueA', 'FindResourceExW'}

Cluster 1: 15 {'PStoreCreateInstance', 'NetUserGetInfo', 'FindWindowExA', 'CopyFileExW', 'FindResourceExW', 'RegDeleteKeyA', 'accept', 'InternetGetConnectedState', 'recvfrom', 'OutputDebugStringW', 'RegDeleteValueA', 'Module32NextW', 'DnsQuery_A', 'HTTPSFinalProv', 'listen'}

Cluster 0: 44 {'NtDeleteFile', 'PStoreCreateInstance', 'NetUserGetInfo', 'FindResourceExW', 'RegEnumValueA', 'GetDiskFreeSpaceExA', 'CreateProcessInternalW', 'RegEnumKeyExA', 'NtRaiseHardError', 'BCryptEncrypt', 'DnsQuery_A', 'WSAConnect', 'NtCreateThreadEx', 'InternetCrackUrlA', 'Module32FirstW', 'RemoveDirectoryW', 'InternetGetConnectedState', 'ObtainUserAgentString', 'InternetCrackUrlW', 'OutputDebugStringW', 'Module32NextW', 'NetGetJoinInformation', 'CryptImportPublicKeyInfo', 'accept', 'listen', 'CryptDecrypt', 'DeleteFileA', 'FindWindowExA', 'COleScript_ParseScriptText', 'HTTPSCertificateTrust', 'HttpQueryInfoW', 'WaitForDebugEvent', 'InternetReadFile', 'GetAsyncKeyState', 'CopyFileExW', 'HttpAddRequestHeadersA', 'RegDeleteKeyA', 'NtLoadKeyEx', 'recvfrom', 'InternetSetOptionA', 'CryptDeriveKey', 'RegDeleteValueA', 'HTTPSFinalProv', 'SetupDiGetClassDevsA'}

Figure 1.3: Individual Malware Row API Row Sequence Call/cluster. Each API sequence calls are compared against other clusters on Figure 1.
cluster: 0 {False: 30, True: 127}
cluster:1 {True: 332, False: 20}
cluster:2{True: 399, False: 7}
cluster:3 {True: 68, False: 5}
cluster: 4 {True: 44, False: 2} cluster:5 {True: 90, False: 1}
cluster:6 {True: 5, False: 2}
cluster:7 {True: 6, False: 1}
cluster:-1 {False: 6, True: 2}

Separate the sequence API function calls on each row for comparison

Figure R: Application Programming Interface (API) Comparison

Drill Down Data: Encryption API Drill Down [Comparison]

Encryption is the process of converting original data, known as plaintext, into another format, which is known as ciphertext. Individuals and companies protect sensitive data by encryption at rest and in transit from hacking. To encrypt a unique encryption key is needed to decrypt the encrypted data. It is essential for organizations or any entities to protect their data. However, bad actors who are either one person, a group, or an organized crime, could use encryption to encrypt files or the whole systems' victims to prevent access to their computer systems or important files and demand ransom for their return. Ransomware, malicious software, is a form of malware used by bad actors. They encrypt victims' system files to lock by making them unreadable or by locking the computer system directly. As part of the ABI profiling process, it is essential to identify all the most common methods used for encryption. For the study, the dynamic application programming interfaces (API) usages for encryption are collected for visualization. Figure S displays 20 dynamic API encryptions that are identified on all 8 clusters during the observation phase.

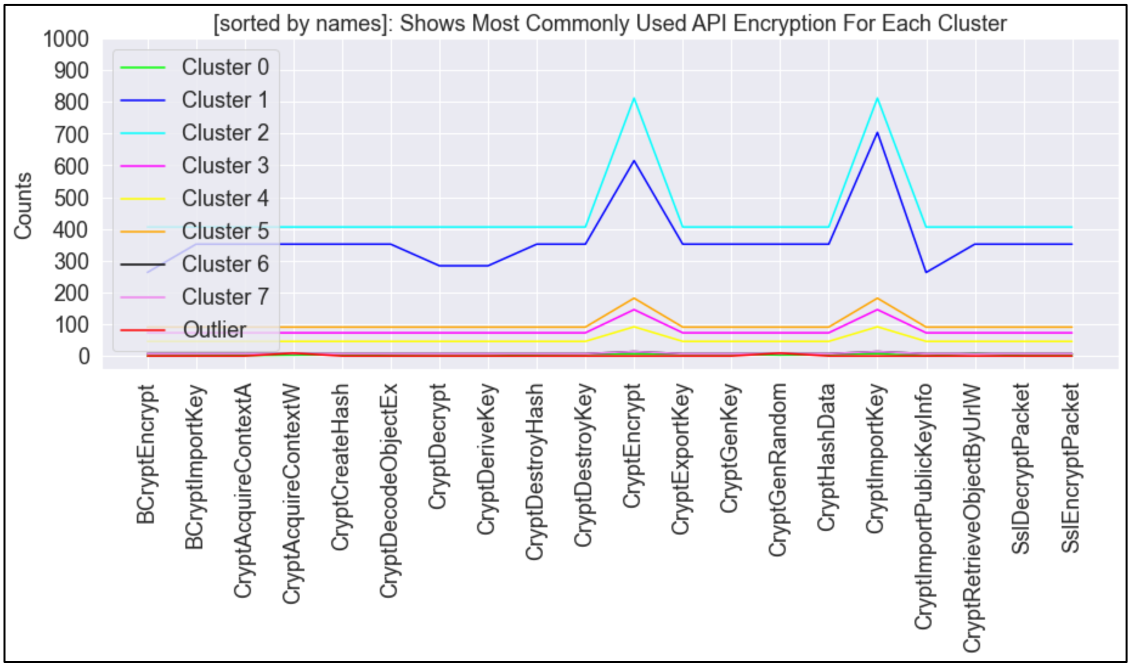


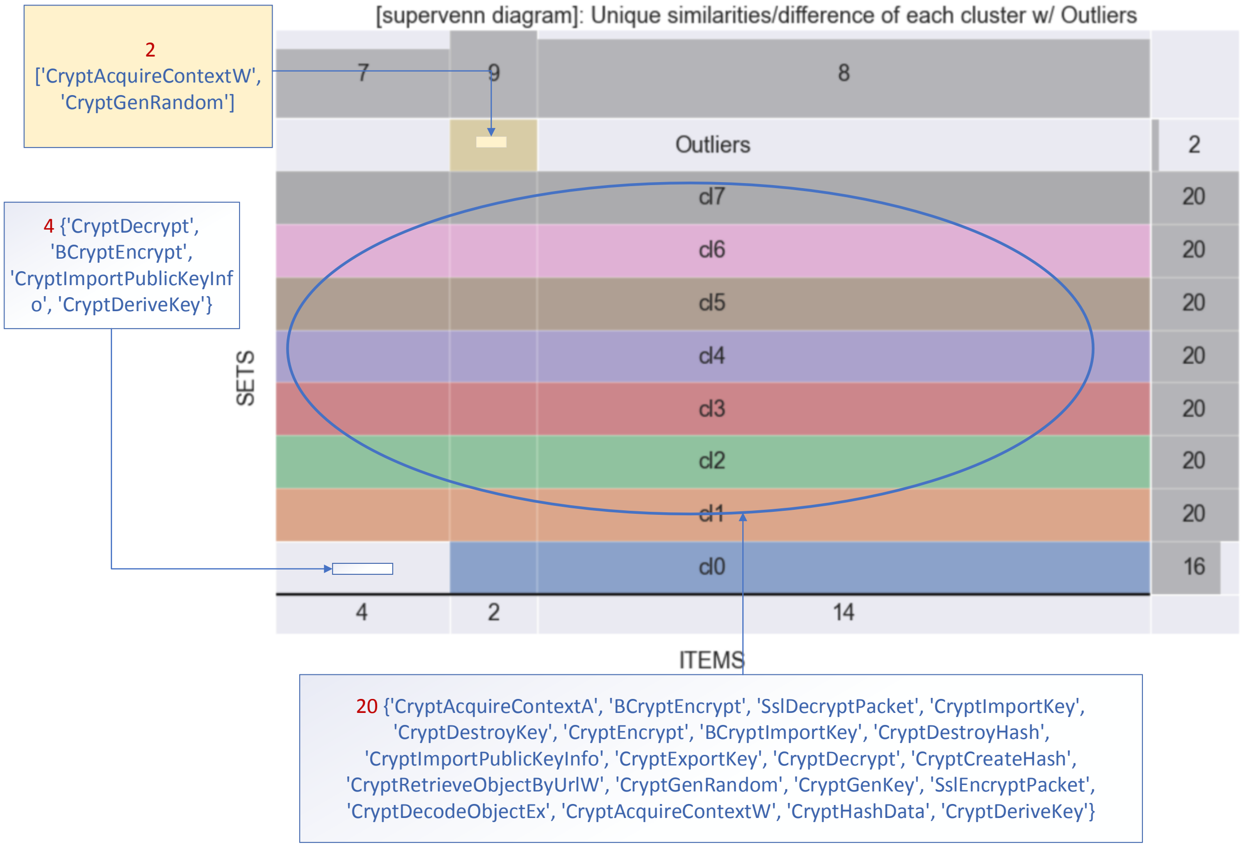
Figure S: API Encryption Names

The most notable and commonly used are "CrypExportKey" and "CrypImpKey." The pattern dictates that they are the primary API tool to lock files or the systems. In addition, based on Figure P, there are 20 identified unique encryption API functions used in clusters 1 to 7. Cluster 0 is the difference between 4 APIs missing from the rest of the clusters. In other words, individual malware uses all API encryption baselines, the identified unique ransomware signatures, to lock the victims' machines. Finally, the outliers only use two encryption APIs, namely CryptAcquireContextW and CryptGenRandom, as shown in Table 6.

	Encryption API	**Cl0**	**CL1**	**CL2**	**CL3**	**CL4**	**CL5**	**CL6**	**CL7**	**Outlier**
0	BCryptEncrypt	0	263	406	73	46	91	7	7	0
1	BCryptImportKey	121	352	406	73	46	91	7	7	0
2	CryptAcquireContextA	127	352	406	73	46	91	7	7	0
3	CryptAcquireContextW	157	352	406	73	46	91	7	7	3
4	CryptCreateHash	128	352	406	73	46	91	7	7	0
5	CryptDecodeObjectEx	121	352	406	73	46	91	7	7	0
6	CryptDecrypt	0	284	406	73	46	91	7	7	0
7	CryptDeriveKey	0	284	406	73	46	91	7	7	0
8	CryptDestroyHash	128	352	406	73	46	91	7	7	0
9	CryptDestroyKey	150	352	406	73	46	91	7	7	0
10	CryptEncrypt	**139**	**615**	**812**	**146**	**92**	**182**	**14**	**14**	**0**
11	CryptExportKey	150	352	406	73	46	91	7	7	0
12	CryptGenKey	150	352	406	73	46	91	7	7	0
13	CryptGenRandom	157	352	406	73	46	91	7	7	3
14	CryptHashData	128	352	406	73	46	91	7	7	0
15	CryptImportKey	**260**	**704**	**812**	**146**	**92**	**182**	**14**	**14**	**0**
16	CryptImportPublicKeyInfo	0	263	406	73	46	91	7	7	0
17	CryptRetrieveObjectByUrlW	45	352	406	73	46	91	7	7	0
18	SslDecryptPacket	121	352	406	73	46	91	7	7	0
19	SslEncryptPacket	121	352	406	73	46	91	7	7	0

Table 6. Encryption API Statistics Per Cluster

Comparing each cluster's unique encryption API features, it is discovered that cluster 1 to cluster seven demonstrate they have utilized the same encryption API as shown in Figure T. It indicates the general API baseline source code used for ransomware. The outlier of the supervene diagram shows some similarities among the clusters with some key differences; it has missing APIs from the baseline indicated in Figure R number 4. Based on ABI, It is the starting point for comparisons among different ransomware deployed in the wild.



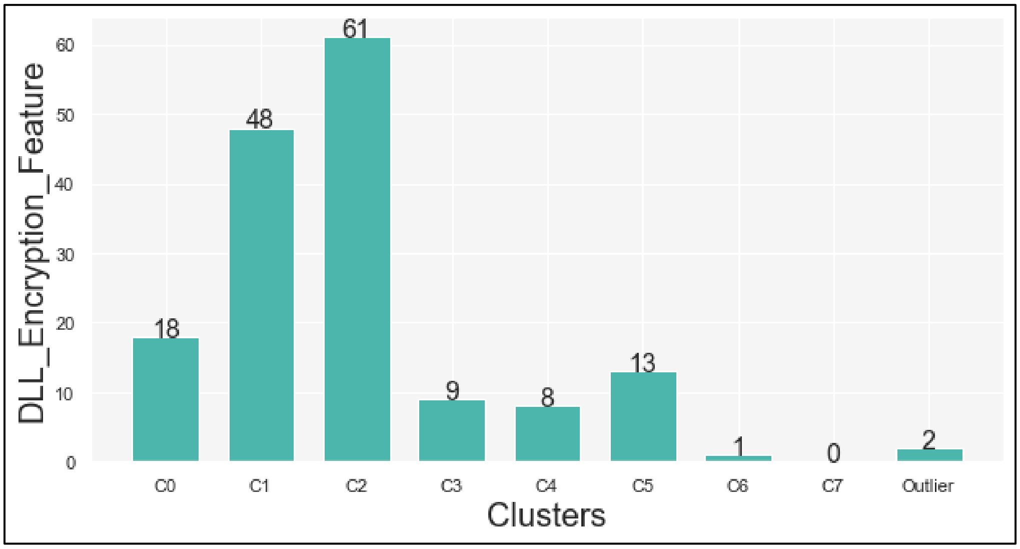
2 ['CryptAcquireContextW', 'CryptGenRandom']

4 {'CryptDecrypt', 'BCryptEncrypt', 'CryptImportPublicKeyInfo', 'CryptDeriveKey'}

20 {'CryptAcquireContextA', 'BCryptEncrypt', 'SslDecryptPacket', 'CryptImportKey', 'CryptDestroyKey', 'CryptEncrypt', 'BCryptImportKey', 'CryptDestroyHash', 'CryptImportPublicKeyInfo', 'CryptExportKey', 'CryptDecrypt', 'CryptCreateHash', 'CryptRetrieveObjectByUrlW', 'CryptGenRandom', 'CryptGenKey', 'SslEncryptPacket', 'CryptDecodeObjectEx', 'CryptAcquireContextW', 'CryptHashData', 'CryptDeriveKey'}

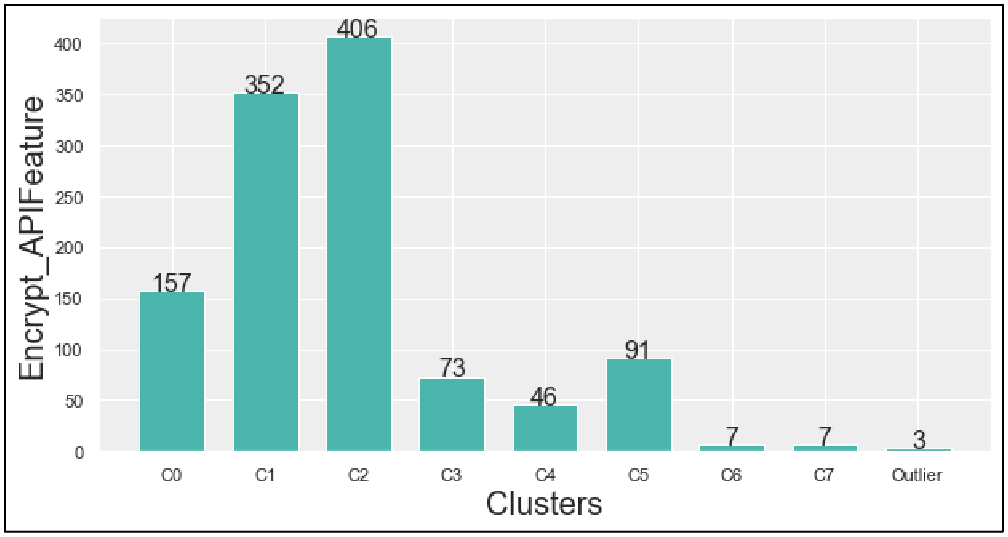
Figure T: Dynamic API Encryption Comparison

The Dynamic Link Library (DLL) static portable executable analysis on Bar Chart 1 shows the clusters used by CRYPT32.DLL. The DLL is a Microsoft module of the Windows Operating System (OS) implementing certificate and cryptographic messaging functions. Different versions of MS Windows come with different capabilities, but the research uses Windows 10. CRYPT32.DLL is in the C:\Windows\SysWOW64\crypt32.dll file, and there are 207 other DLL files in the system32 directory that are statically linked to this file. In those words, the ransomware is a 32-bit system running in the subsystem of the 64-bit Window environment that uses the DLL.



Bar Chart 1: Static Analysis Dynamic Link Library (DLL)

Finally, Bar Chart 2 shows the number of dynamically invoked cryptographic function processes for each cluster, uniquely identified in Figure T, showing the unique baselines for each cluster. This chart shows the number of times the encryption APIs were called, indicating the change in the program of the ransomware functionality.



Bar Chart 2: Dynamic Encryption API Class Per Cluster

Drill Down Data: File Function API Usage

Microsoft Windows comes with API file and directory management, an input/output device that can delete, move, create, download, and drop files or directories. They are heavily used by ransomware, which can create files to encrypt victims' files, and dropped files contain messages for the victims to read. That is why the malware needs access to the file DLL functions to perform its destructive tasks. Figure U, derived from Table 7, identifies all API file functions on each cluster for file window manipulation regarding reading, creating, copying, moving, and writing files or directories. These are ransomware's main file function characteristics to create or drop files. Cluster 2 does not conform to the pattern of the rest of the clusters, with minimal slope changes; however, the CopyFileExW function, which copies an existing file to a new one, has dipped sharply.

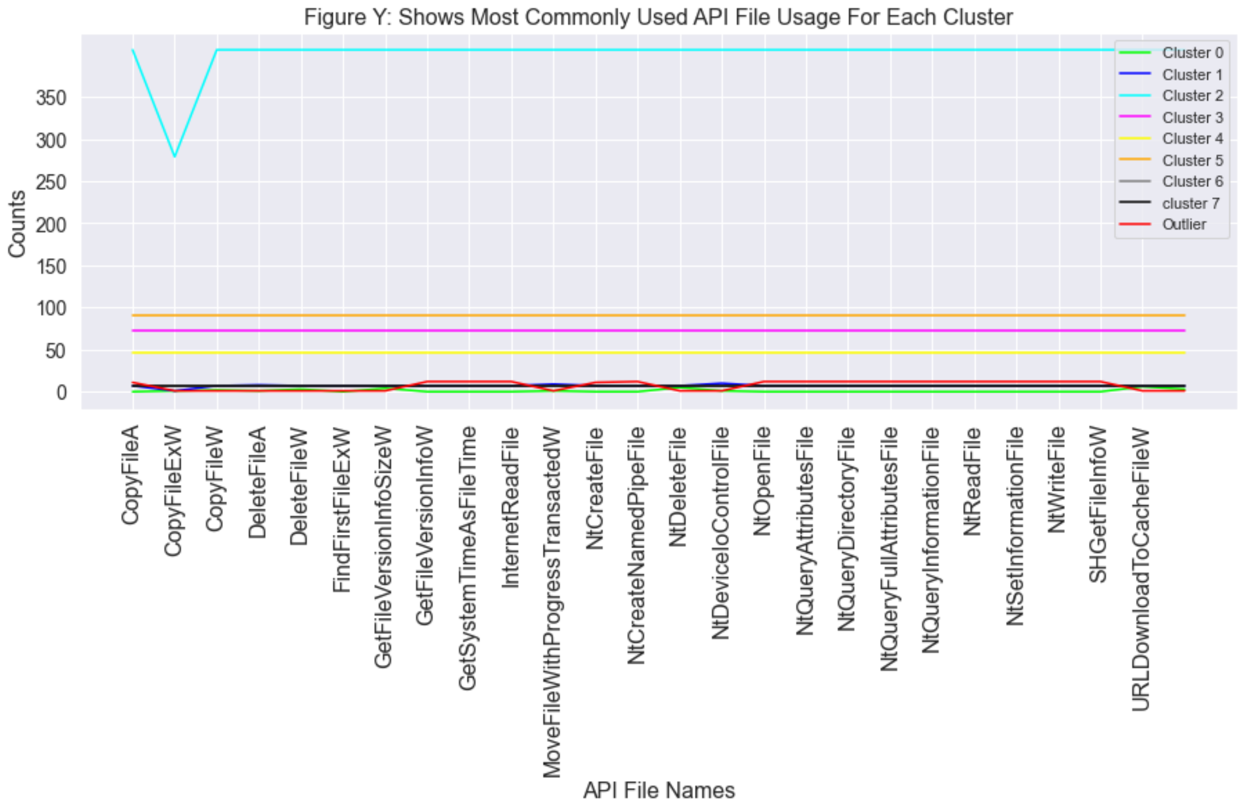
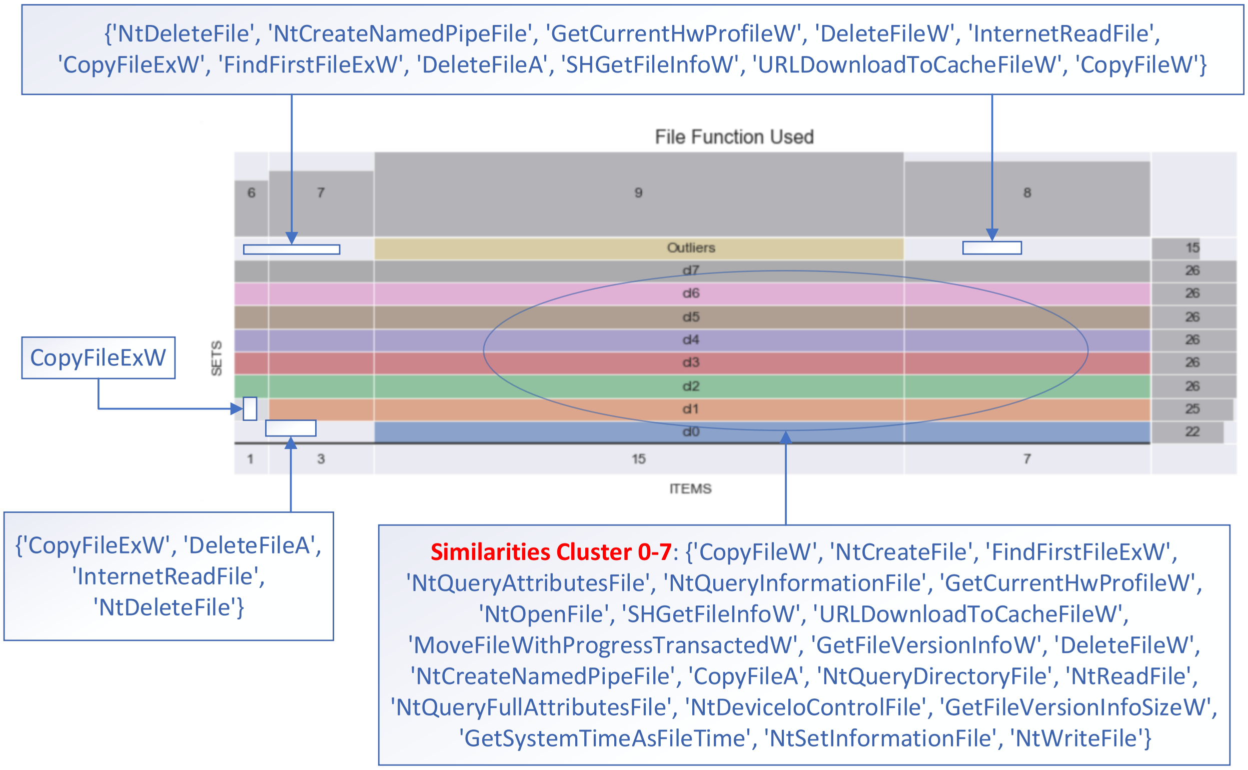


Figure U: API File Names

	API_FILENAMES	0	1	2	3	4	5	6	7	Outlier
0	CopyFileA	157	352	406	73	46	91	7	7	1
1	CopyFileExW	0	0	279	73	46	91	7	7	0
2	CopyFileW	139	352	406	73	46	91	7	7	0
3	DeleteFileA	0	272	406	73	46	91	7	7	0
4	DeleteFileW	98	352	406	73	46	91	7	7	0
5	FindFirstFileExW	157	352	406	73	46	91	7	7	0
6	GetFileVersionInfoSizeW	157	352	406	73	46	91	7	7	8
7	GetFileVersionInfoW	157	352	406	73	46	91	7	7	8
8	GetSystemTimeAsFileTime	157	352	406	73	46	91	7	7	8
9	InternetReadFile	0	144	406	73	46	91	7	7	0
10	MoveFileWithProgressTransactedW	157	352	406	73	46	91	7	7	1
11	NtCreateFile	157	352	406	73	46	91	7	7	8
12	NtCreateNamedPipeFile	150	352	406	73	46	91	7	7	0
13	NtDeleteFile	0	172	406	73	46	91	7	7	0
14	NtDeviceIoControlFile	157	352	406	73	46	91	7	7	8
15	NtOpenFile	157	352	406	73	46	91	7	7	8
16	NtQueryAttributesFile	157	352	406	73	46	91	7	7	8
17	NtQueryDirectoryFile	157	352	406	73	46	91	7	7	8
18	NtQueryFullAttributesFile	157	352	406	73	46	91	7	7	8
19	NtQueryInformationFile	157	352	406	73	46	91	7	7	8
20	NtReadFile	157	352	406	73	46	91	7	7	8
21	NtSetInformationFile	157	352	406	73	46	91	7	7	8
22	NtWriteFile	157	352	406	73	46	91	7	7	8
23	SHGetFileInfoW	127	352	406	73	46	91	7	7	0
24	URLDownloadToCacheFileW	98	352	406	73	46	91	7	7	0

Table 7: File Usage Number of Occurences

Figure V displays the unique features identified on each cluster. Cluster 1 and Cluster 0 have missing functions not identified in clusters 2-7; however, they have similarities with other clusters, establishing it to be the current baselines of the ransomware usage of file and directory manipulation functions. The outlier has more function file APIs not identified in the main clusters. The most common file manipulation characteristics are from cluster 1 to cluster 7.



{'NtDeleteFile', 'NtCreateNamedPipeFile', 'GetCurrentHwProfileW', 'DeleteFileW', 'InternetReadFile', 'CopyFileExW', 'FindFirstFileExW', 'DeleteFileA', 'SHGetFileInfoW', 'URLDownloadToCacheFileW', 'CopyFileW'}

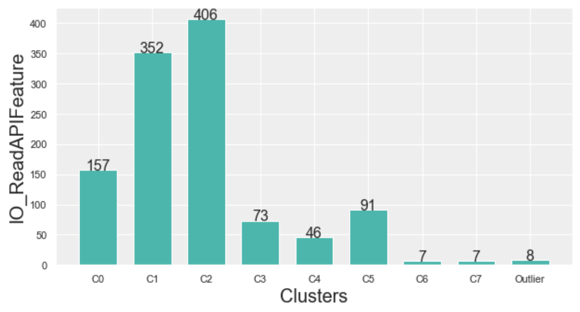
CopyFileExW

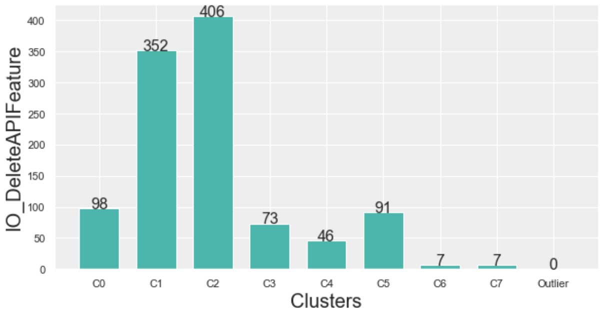
{'CopyFileExW', 'DeleteFileA', 'InternetReadFile', 'NtDeleteFile'}

Similarities Cluster 0-7: {'CopyFileW', 'NtCreateFile', 'FindFirstFileExW', 'NtQueryAttributesFile', 'NtQueryInformationFile', 'GetCurrentHwProfileW', 'NtOpenFile', 'SHGetFileInfoW', 'URLDownloadToCacheFileW', 'MoveFileWithProgressTransactedW', 'GetFileVersionInfoW', 'DeleteFileW', 'NtCreateNamedPipeFile', 'CopyFileA', 'NtQueryDirectoryFile', 'NtReadFile', 'NtQueryFullAttributesFile', 'NtDeviceIoControlFile', 'GetFileVersionInfoSizeW', 'GetSystemTimeAsFileTime', 'NtSetInformationFile', 'NtWriteFile'}

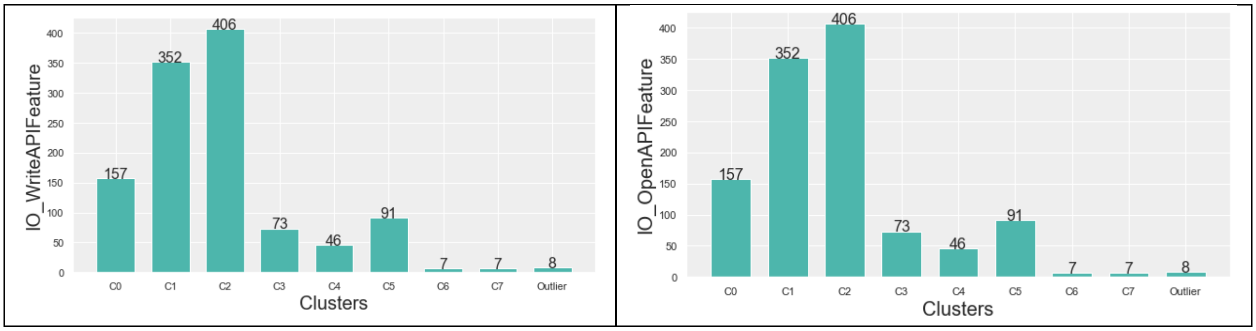
Figure V: File Similarities and Differences of each cluster and outlier

Figure V shows the uniqueness of the functions being used per cluster, which gives the maximum number of 26 unique functions ransomware uses. It coincides with Bar Chart 3 below, showing the total number of processes running per cluster. The activities show several closely related patterns, signifying that the ransomware is the collection of malware from the same source deployed in the wild.

<table>
<tr><th>Reading File</th><th>Deleting File</th></tr>
<tr><td>


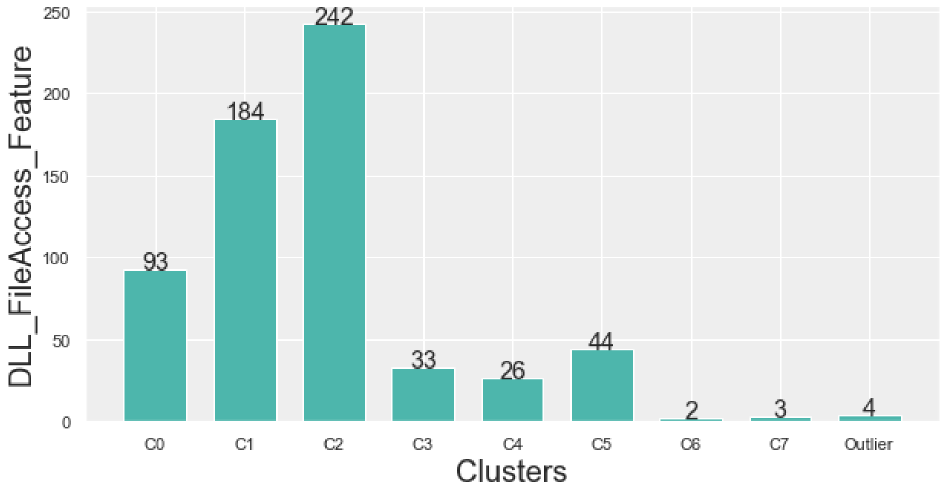
</td><td>


</td></tr>
<tr><td>Writing File</td><td>Opening File</td></tr>
</table>



Bar Chart 3: Total Number of Dynamic File Activities

For the static analysis of the portable executable (PE), the dynamic link library of the Shell.dll shows on Bar Chart 4 that the pattern shows the statically imported DLL into the malware. Shell32.dll, located in *C:\WINDOWS\system32\shell32.dll of Windows 10*, is generally part of the Windows DLL that could be used for opening web pages and files. However, it could also be used to replace the Windows shell file to infect the victim's computer.



Bar Chart 4: PE Import Shell32.dll

Drill Down Data: DLL Used [Static]

Portable Executable (PE) files use Dynamic Link Library (DLL) functions, the imported files that contain code and data. They are used by more than one program at the same time: it is a shared program library in the Windows environment. Figure W shows all the cumulative DLLs used by the ransomware. Seven clusters are identified. Notice that kernel32 and User32 are the most commonly used DLLs. Kernel32.dll is one of the significant findings for this research because it is responsible for other file functions than ransomware requires. The other DLLs pay attention to the "Crypt32" and "Cryptnet," which are used for cryptographic messaging and Microsoft's crypto network-related API. Figure W below describes the lined pattern's similarities, differences, and outliers identified in Figure X. Notice that clusters 2 to cluster 7 have very similar characteristics as compared to cluster 0 and cluster 1.

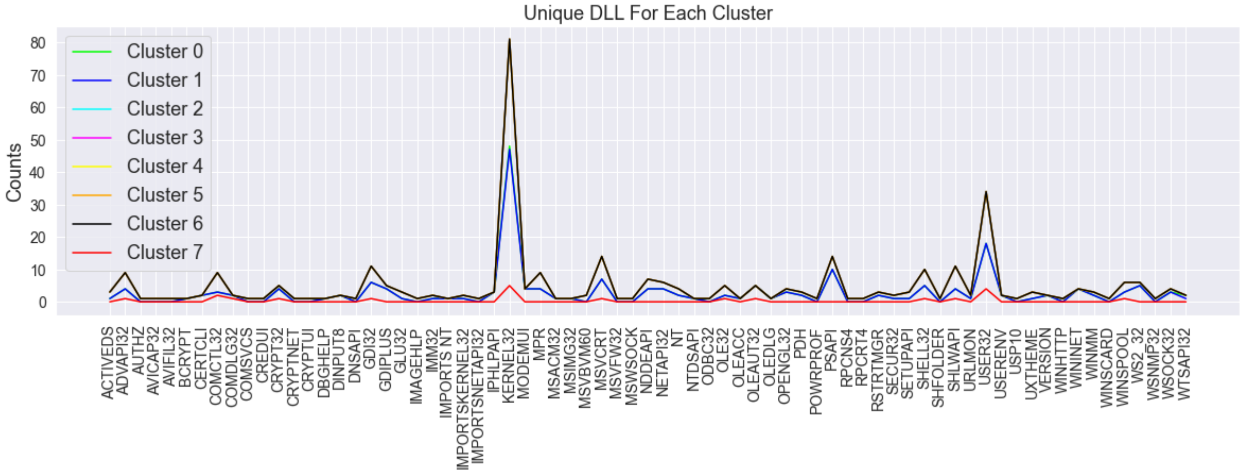
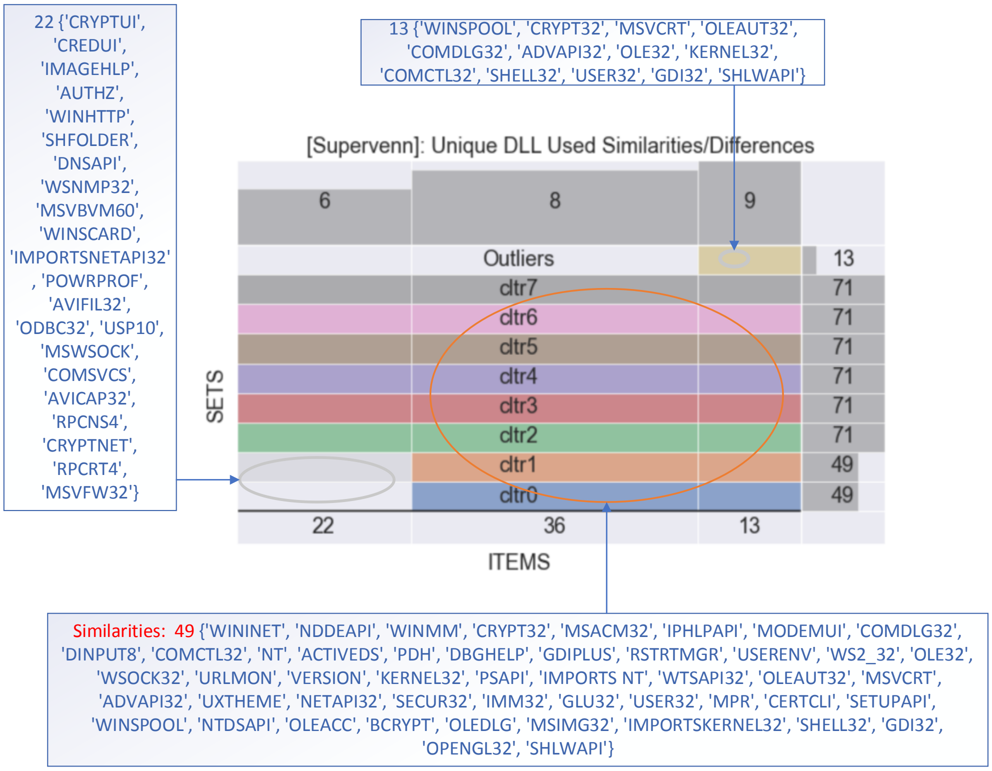


Figure W: Portable Executable DLL Names



22 {'CRYPTUI', 'CREDUI', 'IMAGEHLP', 'AUTHZ', 'WINHTTP', 'SHFOLDER', 'DNSAPI', 'WSNMP32', 'MSVBVM60', 'WINSCARD', 'IMPORTSNETAPI32', 'POWRPROF', 'AVIFIL32', 'ODBC32', 'USP10', 'MSWSOCK', 'COMSVCS', 'AVICAP32', 'RPCNS4', 'CRYPTNET', 'RPCRT4', 'MSVFW32'}

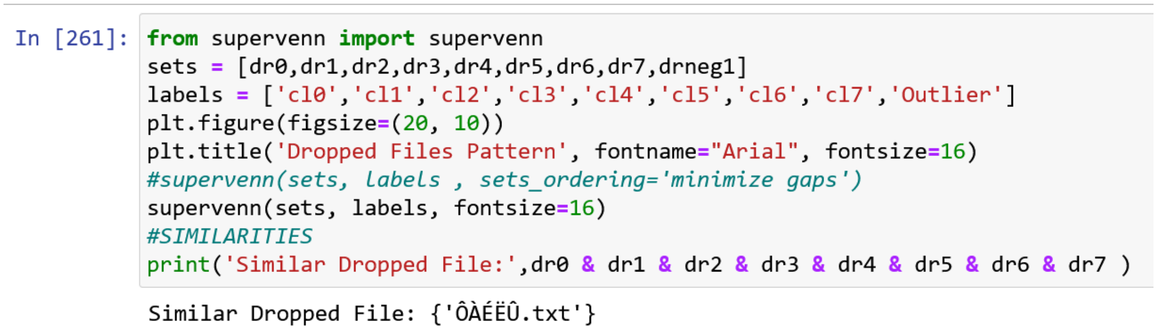
13 {'WINSPOOL', 'CRYPT32', 'MSVCRT', 'OLEAUT32', 'COMDLG32', 'ADVAPI32', 'OLE32', 'KERNEL32', 'COMCTL32', 'SHELL32', 'USER32', 'GDI32', 'SHLWAPI'}

Similarities: 49 {'WININET', 'NDDEAPI', 'WINMM', 'CRYPT32', 'MSACM32', 'IPHLPAPI', 'MODEMUI', 'COMDLG32', 'DINPUT8', 'COMCTL32', 'NT', 'ACTIVEDS', 'PDH', 'DBGHELP', 'GDIPLUS', 'RSTRTMGR', 'USERENV', 'WS2_32', 'OLE32', 'WSOCK32', 'URLMON', 'VERSION', 'KERNEL32', 'PSAPI', 'IMPORTS NT', 'WTSAPI32', 'OLEAUT32', 'MSVCRT', 'ADVAPI32', 'UXTHEME', 'NETAPI32', 'SECUR32', 'IMM32', 'GLU32', 'USER32', 'MPR', 'CERTCLI', 'SETUPAPI', 'WINSPOOL', 'NTDSAPI', 'OLEACC', 'BCRYPT', 'OLEDLG', 'MSIMG32', 'IMPORTSKERNEL32', 'SHELL32', 'GDI32', 'OPENGL32', 'SHLWAPI'}

Figure X: DLL Similarities and Differences

Drill Down Data: Dropped Files

Malware dropped files are files of different formats that are written to disk victims during the malware execution phase. They are dropped during the execution in the victims' system or in the malware sandbox anywhere in the Windows directories. Files could be dropped via the creation of the file or downloaded via the network of the malware. The files may contain codes to install a second file called Payload. They may also create readable log files; for example, in a Windows environment *.log, *.txt, *. readme, etc, are the most common files generated by ransomware. Figure W shows the similarities and differences between the files downloaded per cluster. In the line pattern of Figure Y, there are hardly any intersections, meaning file names do not seem to have exact similarities; they are randomly generated. However, using the Python intersection command of all the clusters, except the outlier, has shown one notable similarity, as shown below in the code snippet 1 “Similar Dropped File” label.



```
In [261]: from supervenn import supervenn
          sets = [dr0,dr1,dr2,dr3,dr4,dr5,dr6,dr7,drneg1]
          labels = ['cl0','cl1','cl2','cl3','cl4','cl5','cl6','cl7','Outlier']
          plt.figure(figsize=(20, 10))
          plt.title('Dropped Files Pattern', fontname="Arial", fontsize=16)
          #supervenn(sets, labels , sets_ordering='minimize gaps')
          supervenn(sets, labels, fontsize=16)
          #SIMILARITIES
          print('Similar Dropped File:',dr0 & dr1 & dr2 & dr3 & dr4 & dr5 & dr6 & dr7 )

          Similar Dropped File: {'ÔÀÉËÛ.txt'}
```

Code Snippet 1: Similar Dropped File of All 8 clusters.

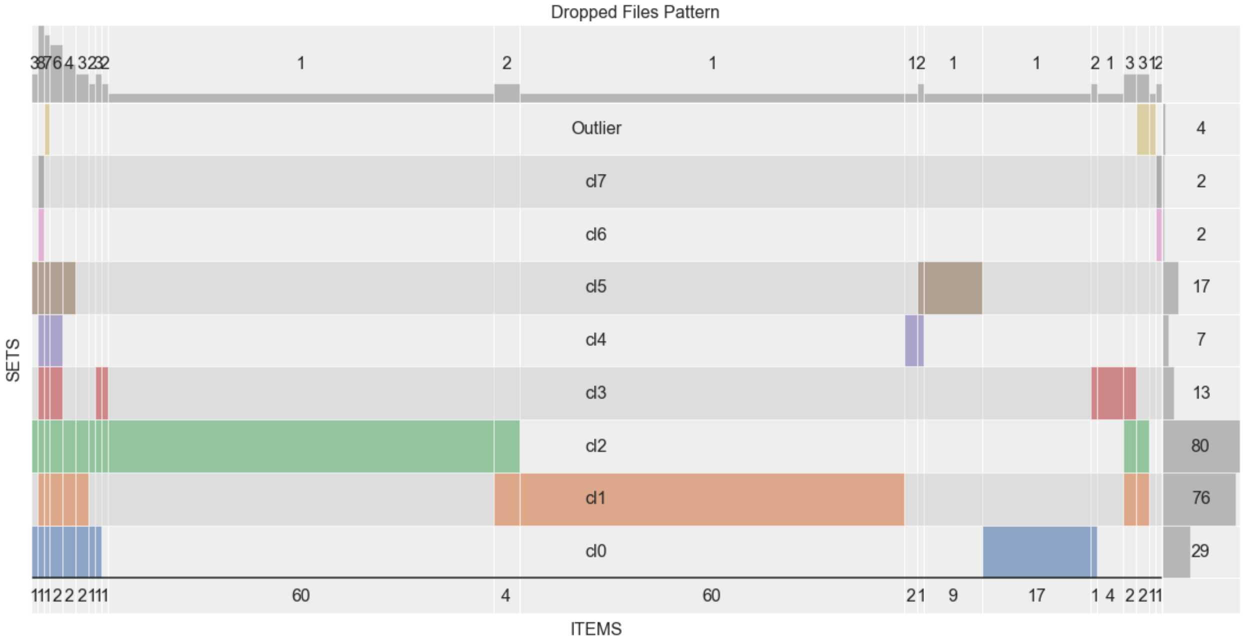


Figure Y: Files Dropped by Ransomware Per Cluster

In addition to observing the dropped files, one of the notable characteristics of ransomware is data corruption and file encryption with randomized and generated files. In sections A, B, and C of Figure Z, the executable is similar to randomly generated string filenames. Section B shows *.chk is windows corrupted file isolated by Windows. Section A shows the *.tmp and *.txt extensions with the most generated files. Section C shows some *.txt and *.logs extensions randomly generated files containing content messages.

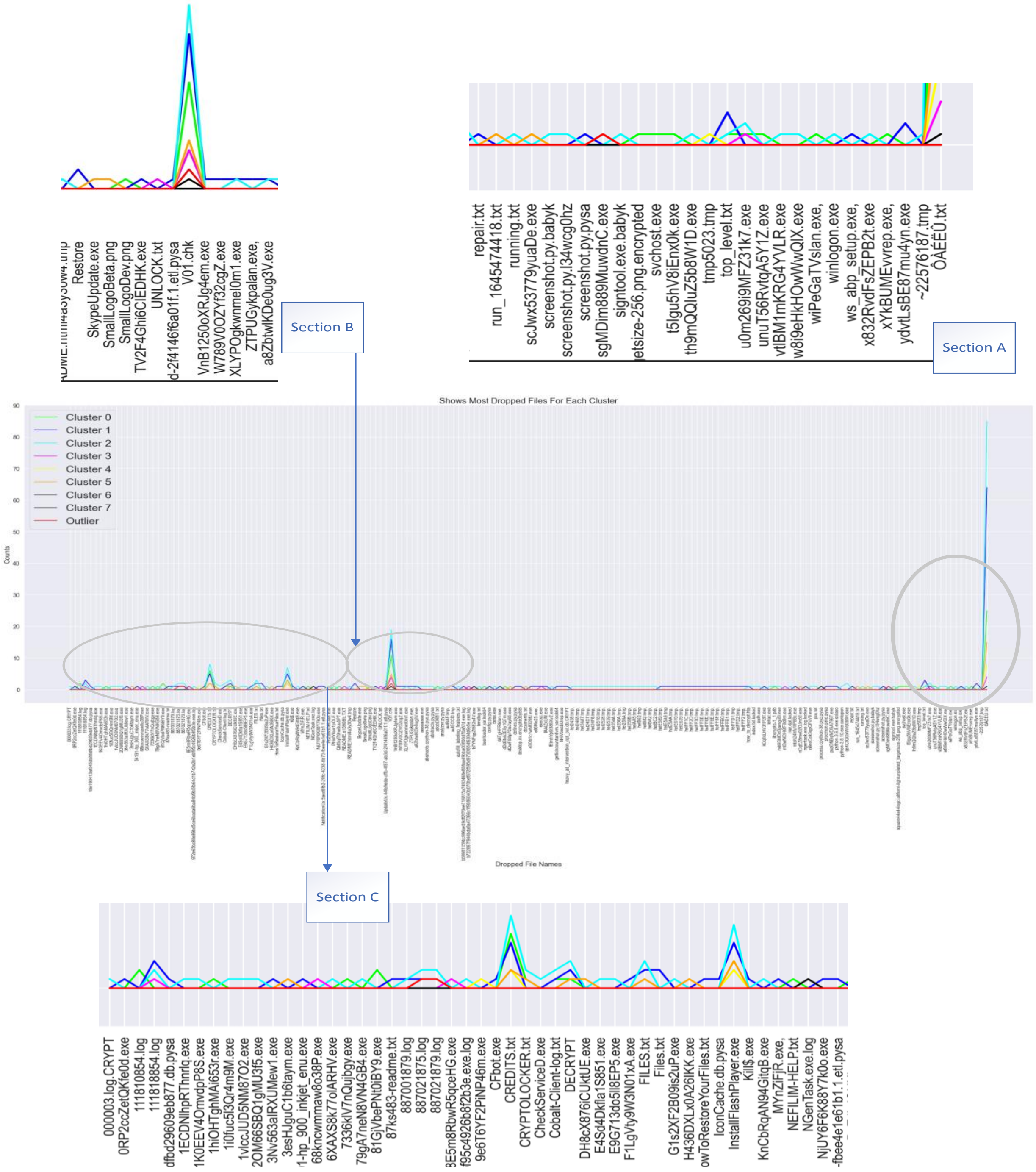


Figure Z: All Dropped Files Pattern Magnified by Section

Drill Down Data: Network Communication Process (DLL)

Network sockets are used as the bidirectional communication channel of the endpoints. They communicate within the Operating System (OS) processes, between processes on the same machine, or between processes on different machines located within the intranet or the internet on different continents. It could be implemented on different types of channels, such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). There is malware that needs to communicate with the internet to perform some functions.

The diagram below, Figure AA, shows two identified application programming interfaces (API), WSOCK32 and MSWSOCK. Microsoft Build description states, “Windows Sockets 2 (Winsock) enables programmers to create advanced Internet, intranet, and other network-capable applications to transmit application data across the wire, independent of the network protocol being used. With Winsock, programmers are provided access to advanced Microsoft® Windows® networking capabilities such as multicast and Quality of Service (QoS) “ [63]. Winsock is a form of API communication between the network software and the network services running in Windows operating systems (OS). Its protocol uses Transmission Control Protocol/Internet Protocol (TCP/IP) to convert the request from the software. Mswsock is a dynamic link library invoked dynamically from the Windows operating system (OS) software applications. Windows sockets (Winsock) is a traditional network programming that is implemented in a Windows environment. It is implemented and compiled using Visual C++. Figure Y shows the related clusters that mostly used WSOCK32 dynamic link library.

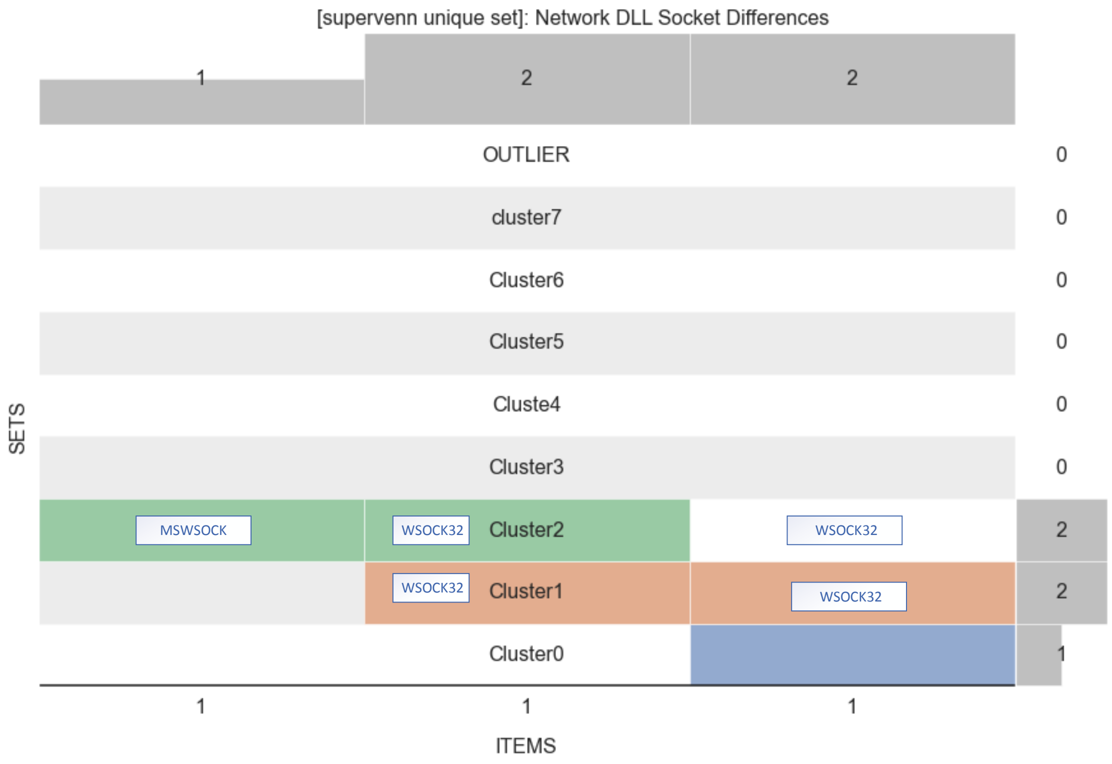
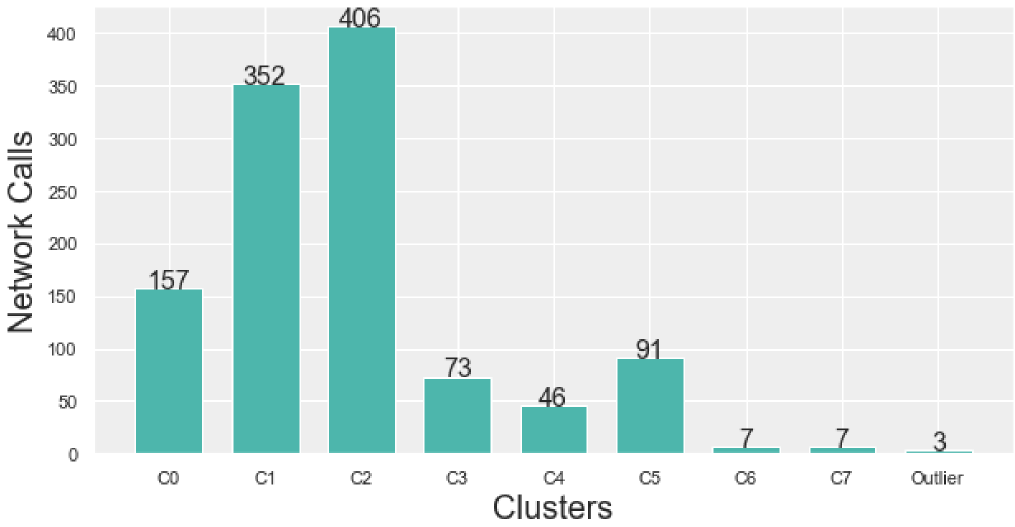


Figure AA: DLL Import Network

Scanning the dynamic API call processes, there are several methods called using HTTP, socket, connect, send, recv, GetAdaptersAddresses, and bind functions. They all exist in all the clusters rendering to be some of the socket connections not using Windows API sockets. They could be using independent API not related to Windows Sockets. Bar Chart 5 shows the socket process connections for each cluster.



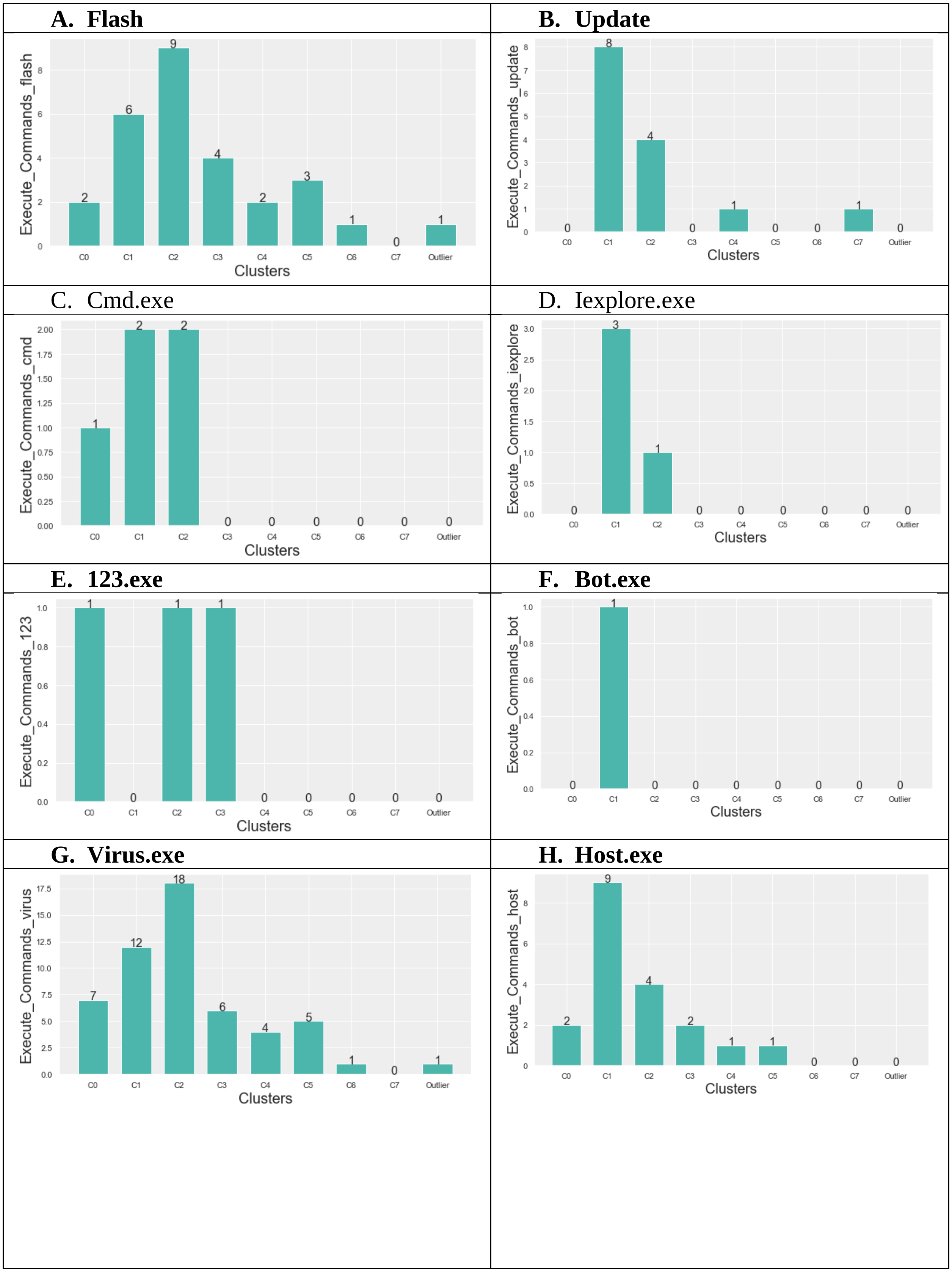
Bar Chart 5: Network Communication Process (Socket API) - Dynamic

In conclusion, Figure AA of the Windows socket of the first three clusters has been used more often than the rest because Bar Chart 5 clusters 0, 1, and 2 have the most function calls since the research observation lab runs on Windows 10 environment. Cluster 3 to 7, which may have used different independent network communication libraries, displays fewer activities than clusters 0 to 2.

Drill Down Background Execution

The default command-line interpreter is the Windows disk operating system (DOS) “Command Prompt”, such as cmd.exe or cmd command. It has existed since the inception of the Windows Operating System (OS). It is mainly used to execute command lines of code to get information or fulfill a user's task. During malware operations, users or systems experience these command prompt windows that appear and disappear automatically. The command executions get executed in the background and appear to execute without the victims' knowledge. They are typically executed to modify the computer configuration settings without the consent of the computer users.

Bar Chart 6 displays some of the common usages of execution on a command prompt after the deployment of the ransomware malware per cluster – flash.exe, update.exe, cmd.exe, iexplore.exe, 123.exe, bot, exe, virus.exe, and taskmanager.exe. The identified names would have been customized to mimic the operating system's commonly used files to masquerade itself to appear as a legitimate software execution.



A. Flash

B. Update

C. Cmd.exe

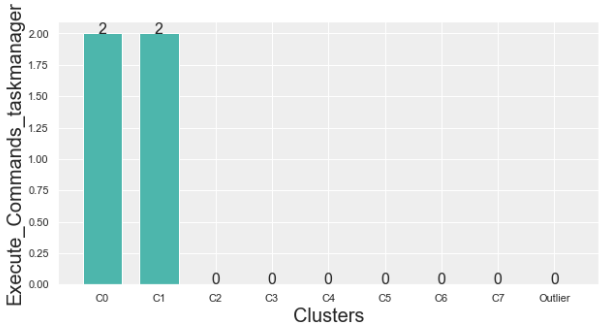
D. Iexplore.exe

E. 123.exe

F. Bot.exe

G. Virus.exe

H. Host.exe

I. Taskmanager.exe	
	

Bar Chart 6. Common Captured Execution Command

3.3.4 Phase IV: Anticipation (Research Questions)

Answering the research questions of the study requires descriptive data analysis and quantitative statistical data captured during the observation phase from section 3.3.3 of the ABI process, which has the collection of the static, dynamic, and network behavior of the ransomware. The research questions undergone to several phases to discover patterns created by the DBSCAN, which generated 8 clusters, including outliers. Each cluster signifies similarities, and the outliers are the data points that do not belong to any clusters. The closer the data points to the cluster group it belongs to, the more similar ransomware is. ABI states," *Intelligence anticipation* is the ability to warn and describe future states of the environment based on the manipulation and synthesis of past and present data. Anticipation includes near-term warning and longer-term forecasting to alert and prepare decision makers to events relevant to their responsibilities." [3]. The data ingested into the DBSCAN were ransomware collected over the years to study its behavior quantitatively. It identifies baselines of the current process and behavior of the malware under study. It also identifies events that are not naturally regular or the course of action when the ransomware was released. As a result, identifying the similarities, differences, and outliers or anomalies of the ransomware among the group answers the research questions as follows under study.

3.3.4.1 Research Question 1 (Similarities/Differences): What are the feature **similarities and differences (anomalies)** of the malware feature activities collected over the years?

There are 1300 uniquely identified ransomware collected from Virus Share and Malware Bazaar website repositories, which have been collected for years. The features of malware have significant similarities and slight differences, along with anomalies or outliers. Tables 3, 4, and 5 summarize the outcome of the descriptive analysis derived from Figures N (All API Calls), P (Encryption Calls), R (File APIs), T (DLLs), and U (Dropped Files). The assessment concludes that the sample profiles of the ransomware in each cluster show significant similarities. It proves that the drill-down investigation of the ransomware static, dynamic, and network analysis demonstrates the object behavior of the ransomware. The same clusters are more similar to each other than to objects in other clusters.

Table 8 describes the anomalies where fewer APIs are used regarding encryption, API events, and DLL files. It does not conform to the rest of the clusters. Each data point belonging to a cluster group shares significant similarities. Given the table data, all ransomware deployed in the wild share similarities where it could be concluded that they all came from the same source.

Features	**No**	**Similar Features**
Encryption (Figure P)	20	CryptGenKey, CryptDecodeObjectEx, CryptExportKey, CryptGenRandom, CryptImportKey, CryptCreateHash, CryptHashData, CryptDestroyKey, CryptDeriveKey, CryptAcquireContextW, SslEncryptPacket, SslDecryptPacket, CryptDestroyHash, BCryptImportKey, CryptImportPublicKeyInfo, BCryptEncrypt, CryptRetrieveObjectByUrlW, CryptAcquireContextA, CryptDecrypt, CryptEncrypt
API Calls (Figure N)	240	See Appendix C
File API (Figure R)	22	DeleteFileW, NtQueryInformationFile, NtQueryFullAttributesFile, NtCreateNamedPipeFile, NtQueryAttributesFile, NtSetInformationFile, GetSystemTimeAsFileTime, NtWriteFile, NtOpenFile, FindFirstFileExW, URLDownloadToCacheFileW, GetCurrentHwProfileW, SHGetFileInfoW, GetFileVersionInfoW, NtReadFile, NtDeviceIoControlFile, NtCreateFile, MoveFileWithProgressTransactedW, NtQueryDirectoryFile, GetFileVersionInfoSizeW, CopyFileW, CopyFileA
DDL Files (Figure T)	49	See Appendix D

Dropped Files (Figure U)	0	Unknown. The files that are dropped are randomly generated. Figure R displays no pattern, and they have a tiny percentage of similarities.
Network Calls (Figure X)	2	MSWSOCK, WSOCK32 (Cluster 0 to 2)

Table 8: Captured Similar Features

The descriptive analysis has shown similarities between all the clusters; however, they have distinct differences, as shown in Table 4. The table displays the features that are not in Table 3 by Clusters. The identified minor differences are the deviations or changes that do not conform to the current state of APIs mainly used by each cluster. As the ABI stated, it is a warning sign that something has changed [3]. It demonstrated that there are added or reduced functionalities of the ransomware, rendering this malicious software continually evolving or changing by the malware's author.

Features	**Cluster**	**No**	**Differences in Features**
Encryption (Figure P)	0	4	CryptDeriveKey, CryptDecrypt, BCryptEncrypt, CryptImportPublicKeyInfo
API Calls (Figure N)	0	44	See Appendix E
	1	18	recvfrom, InternetGetConnectedState, HTTPSFinalProv, PStoreCreateInstance, listen, CopyFileExW, NetUserGetInfo, RegDeleteValueA, accept, OutputDebugStringW, DnsQuery_A, FindWindowExA, RegDeleteKeyA, FindResourceExW, Module32NextW
	2	7	FindResourceExW,HTTPSFinalProv,OutputDebugStringW, PStoreCreateInstance,RegDeleteKeyA,RegDeleteValueA, recvfrom
	3	2	FindResourceExW,RegDeleteValueA
	4	1	RegDeleteValueA
	5	1	RegDeleteValueA
File API (Figure R)	1	1	CopyFileExW
	0	3	CopyFileExW, DeleteFileA, InternetReadFile, NtDeleteFile
DDL Files (Figure T)	0	22	POWRPROF, AUTHZ, MSVFW32, RPCRT4, USP10, IMPORTSNETAPI32, WINSCARD, MSWSOCK, AVICAP32, MSVBVM60, ODBC32, CRYPTUI, AVIFIL32, WSNMP32, SHFOLDER, CRYPTNET, CREDUI, WINHTTP, IMAGEHLP, RPCNS4, COMSVCS, DNSAPI
	1	22	

Dropped Files (Figure U)	0	0	Unknown. The files that are dropped are randomly generated. Figure R displays no pattern, and they have a tiny percentage of similarities.
Network Calls	3-7	0	There are no network calls found

Table 9: Captured Differences of File API Calls.

Finally, DBSCAN describes anomalies that are not part of the clustered data. It is interchangeable with the word outlier. Typically, a group of clusters bands together to show their similarities. However, some minority data points do not fit the rest of the clustered data well. The results use fewer API calls than the rest of the clusters. Table 10 shows the anomalies. Outliers or anomalies are often described as abnormal observations or erroneous data entry. However, in terms of ABI, they are significant enough to define unusual activity or suspicious data that do not conform to the rest of the data, which uses more API functionalities. The malware's author could have created or formed another group of ransomware with minimal functions to avoid being detected.

Features	**No**	**Outliers**
Encryption (Figure P)	2	CryptAcquireContextW, CryptGenRandom
API Calls (Figure N)	159	See Appendix E
File API (Figure R)	15	NtOpenFile, NtQueryDirectoryFile, NtQueryFullAttributesFile, MoveFileWithProgressTransactedW, NtSetInformationFile, NtCreateFile, GetSystemTimeAsFileTime, NtReadFile, NtWriteFile, NtDeviceIoControlFile, CopyFileA, NtQueryAttributesFile, GetFileVersionInfoW, GetFileVersionInfoSizeW, NtQueryInformationFile
DDL Files (Figure T)	13	MSVCRT, KERNEL32, COMCTL32, ADVAPI32, SHLWAPI, OLE32, WINSPOOL, SHELL32, GDI32, COMDLG32, OLEAUT32, USER32, CRYPT32
Dropped Files (Figure U)	0	Unknown. The files that are dropped are randomly generated. Figure R displays no pattern, and they have a very small percentage of similarities.
Network	0	There are no network calls.

Table 10: The Outliers/Anomalies

In conclusion, all the identified clusters indicate a degree of similarities from encryption calls, general API calls, API calls, DLL import, and some minor dropped files and network files. As part of the ABI process and profiling of the target malware, the differences among all these clusters are either added or excluded from the existing ransomware. Ultimately, all the 1300 ransomware came from the same source because they display likeness not only in their static behavior but also in their static characteristics.

3.3.4.2 Research Question 2 (Baselines): How do we identify the constraints to develop a common baseline, a hidden data pattern, for each malware classification or cluster type?

A baseline is a minimum or reference point of comparison in the timeline for future measurements of the malware feature behaviors. It provides a starting point if there has been progress since the initial baseline estimation that quantifies the difference between two points in time. During the study, ransomware behavior was quantified into three different behavior sets: static, dynamic, and network. The quantification of the measures is discussed in the descriptive analysis section of the paper. Based on the 1300 input, there are currently eight identified clusters, meaning that each cluster is similar or closely related, rendering them one form of a ransomware variant.

In addition, the ransomware from features generated from the descriptive analysis of 3.3.3 gives a conclusion from the line graphs in Figures Q, S, U, W, and Z that all ransomware deployed in the wild have a high degree of similarities. The line patterns, along with the description results, would be used as a basis for the current ransomware behavior trend. In addition to the line graphs' trends and patterns have identified unique characteristics for each cluster. The similarity at this point is not precisely the baseline of the ransomware behavior. However, they would be used as the starting point of measurement. The captured similar behavior gives the research the standard current behavior of all 1300 ransomware. Having the baselines at this point would give us the traits or characteristics that may differ from the future samples. It can provide helpful information about the changes or improvements over time. Figure BB, the combined line graphs from different static and behavioral displays the trends that each cluster has similar patterns, described in detail from the descriptive analysis. The patterns demonstrate the number of occurrences of clusters that the up and own lines have similar directions among the clusters.

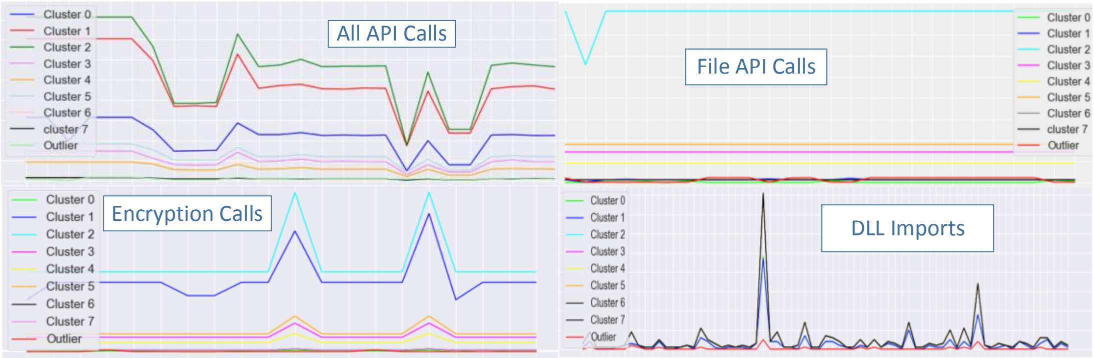


Figure BB: Combined Ransomware Features Pattern Summary

The unique behavioral characteristics define the current APIs used for each cluster, which identifies the baseline process API calls the ransomware. Figure CC describes the summary of the unique clusters with similar characteristics. The API calls from the file, encryption, and general API, as shown in the supervenn diagram, indicate an equal number of unique behaviors, indicating that all the clusters have a significantly high degree of homogeneousness.

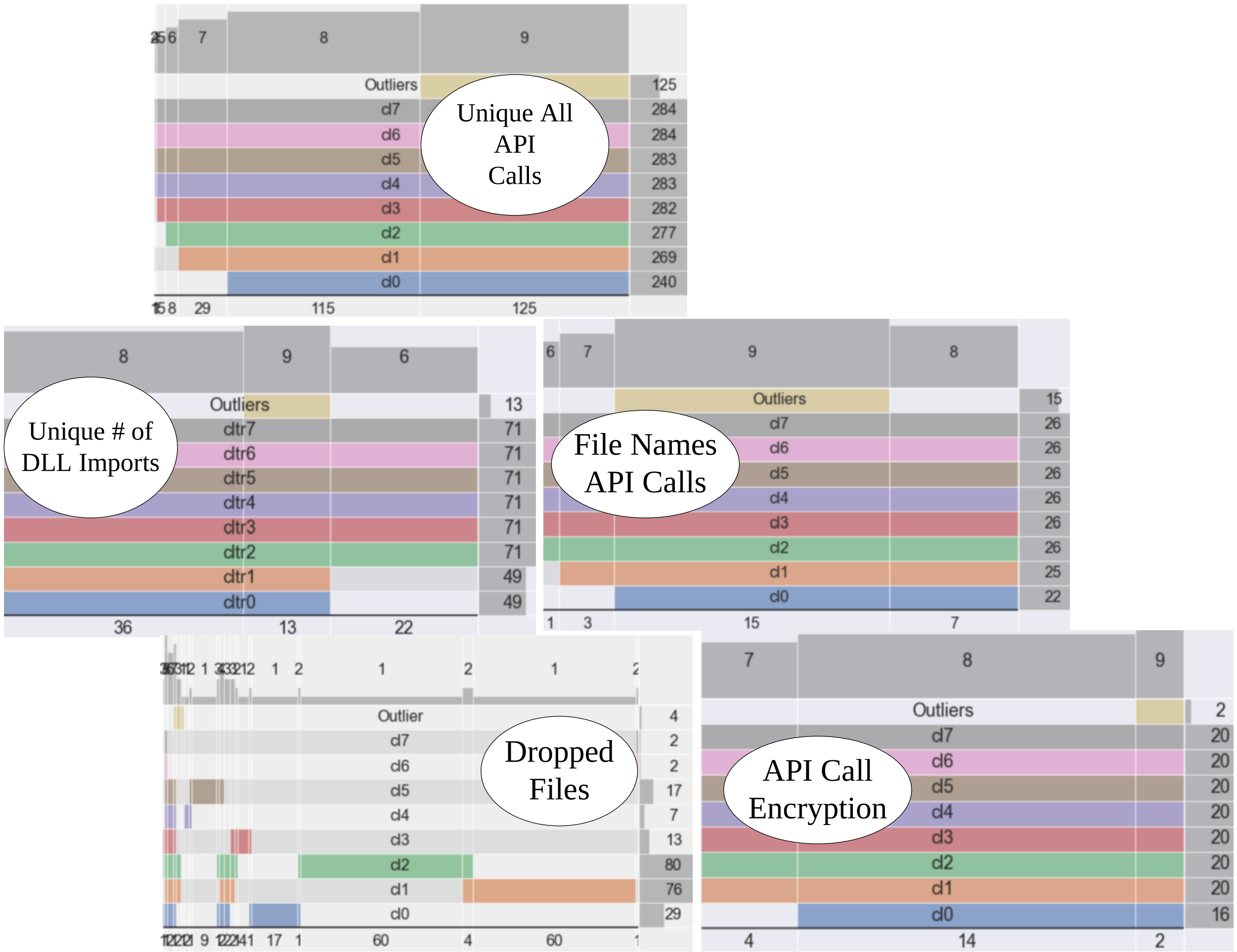


Figure CC. Summary Unique Features

In conclusion, comparing the baseline differences helps monitor changes and provides precious information for decision-makers. In conclusion, Table 3 is the starting point of a baseline of the research. Any future ransomware deployed in the wild could be compared against the similarities to track the differences.

3.3.4.3 Research Question 3 (Source): How can we determine if the variant types of malware in the wild were the original copies of other types of malware?

- Ransomware has generated several interesting features, namely the usage of encryption function, the file manipulation application programming interface, API Calls, the dynamic link library (DLL), and the sequence of API events. The answer to our research questions is that all ransomware. However, they belong to different clusters and portray similarities of calls of events to a high degree. The following five categories are described in detail about their significant similarities. What sets them apart are their behavioral and static differences making them slightly different.

1. Encryption Function

Figure R, the API encryption comparison, shows that cluster 1 (one) to cluster 7 (seven) have the same number of unique encryption API(s) being used. Cluster 0 has four encryption APIs not used in the clusters between 1 and 7. The Figure shows that they are similar enough to come from the same source, rending them to be originated from the exact copy.

2. File API Function

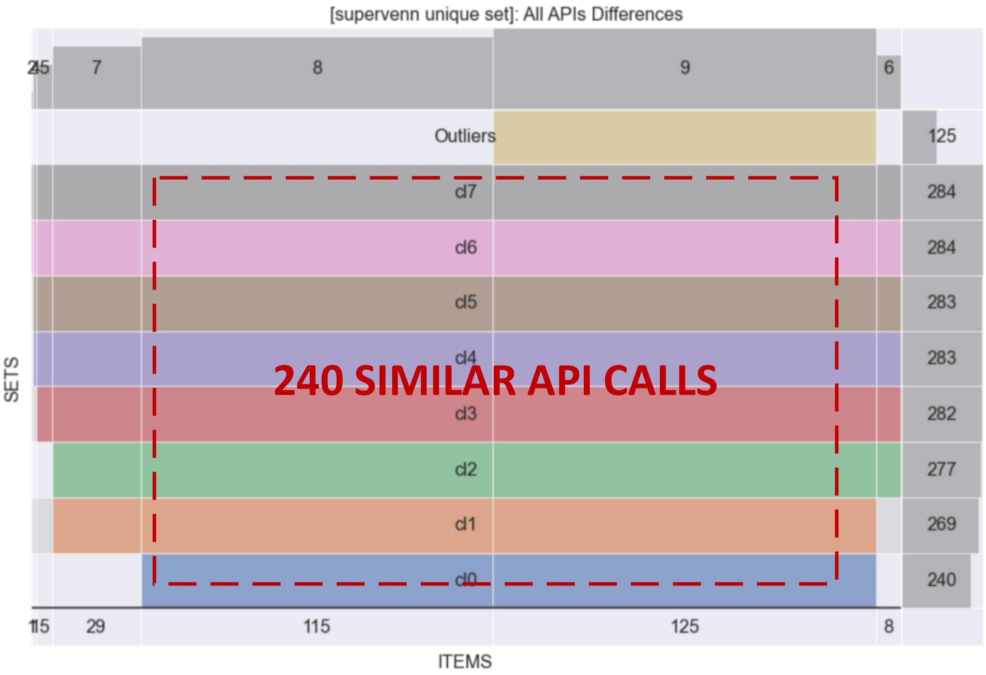
Figure V, the file API function calls, identifies that all clusters use 26 unique API calls with four differences, which are identified as CopyFileExW, DeleteFileA, InternetReadFile, and NtDeleteFile. The Figure concludes that all file access of the ransomware has been significantly used by all malware rending, making them derived from the same source with slight modification.

3. Dynamic Link Library (DLL(s))

Figure V, the DLL comparison, shows that there are 171 imported DLL(s), as it is shown from cluster 2 to cluster 7. Cluster 0 and Cluster 1 have 22 similar DLL(s) that are not identified between Cluster 2 and Cluster 7. Overall, cluster 0 to cluster 8 have 41 similar DLL imported values. It is consequential enough that all the sample ransomware have similar values in this category.

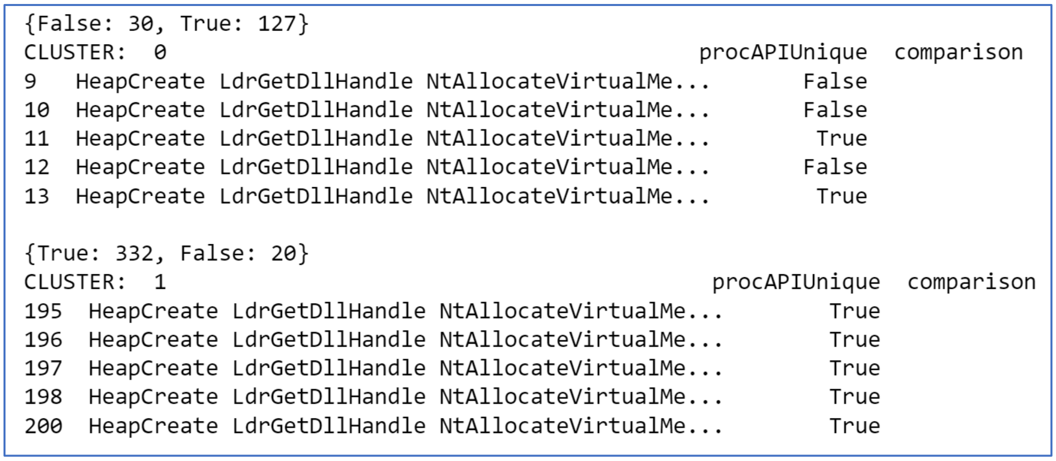
4. All API Calls

All ransomware API calls have been collected and compared against each cluster. Based on Figure P, there are 240 identified event calls from each cluster from cluster 0 to cluster 7. It signifies the similarities of all ransomware regardless of their location in the clusters; they are generally similar. The differences between the clusters are from 1 to 15 API calls in general.



5. API Sequence of Events

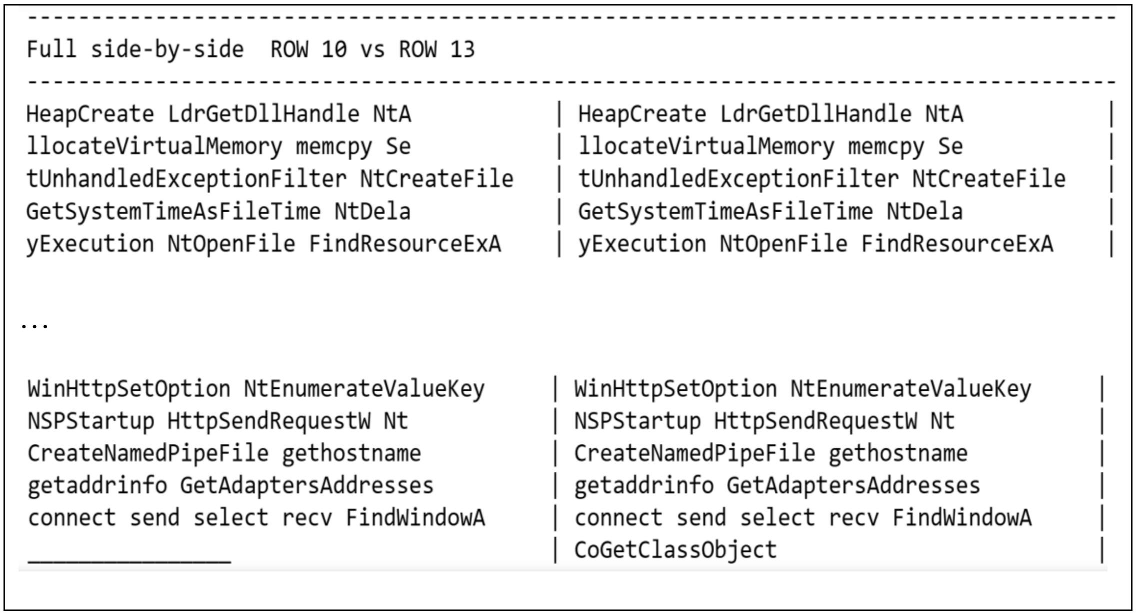
As part of the ABI methodology, one should compare each row's differences and why they differ. What essential points to capture here as part of ABI methodology is to understand the added features of each malware sample since all these samples are ransomware. The sample below, Sample A, is cluster 0, compares the difference between the False and True comparisons. Note that "True" means they have identical sequence calls.



```
{False: 30, True: 127}
CLUSTER:  0                                         procAPIUnique  comparison
9   HeapCreate LdrGetDllHandle NtAllocateVirtualMe...       False
10  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...       False
11  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
12  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...       False
13  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True

{True: 332, False: 20}
CLUSTER:  1                                          procAPIUnique  comparison
195  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
196  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
197  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
198  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
200  HeapCreate LdrGetDllHandle NtAllocateVirtualMe...        True
```

Sample A: Comparison of row-by-row similarities of API Calls

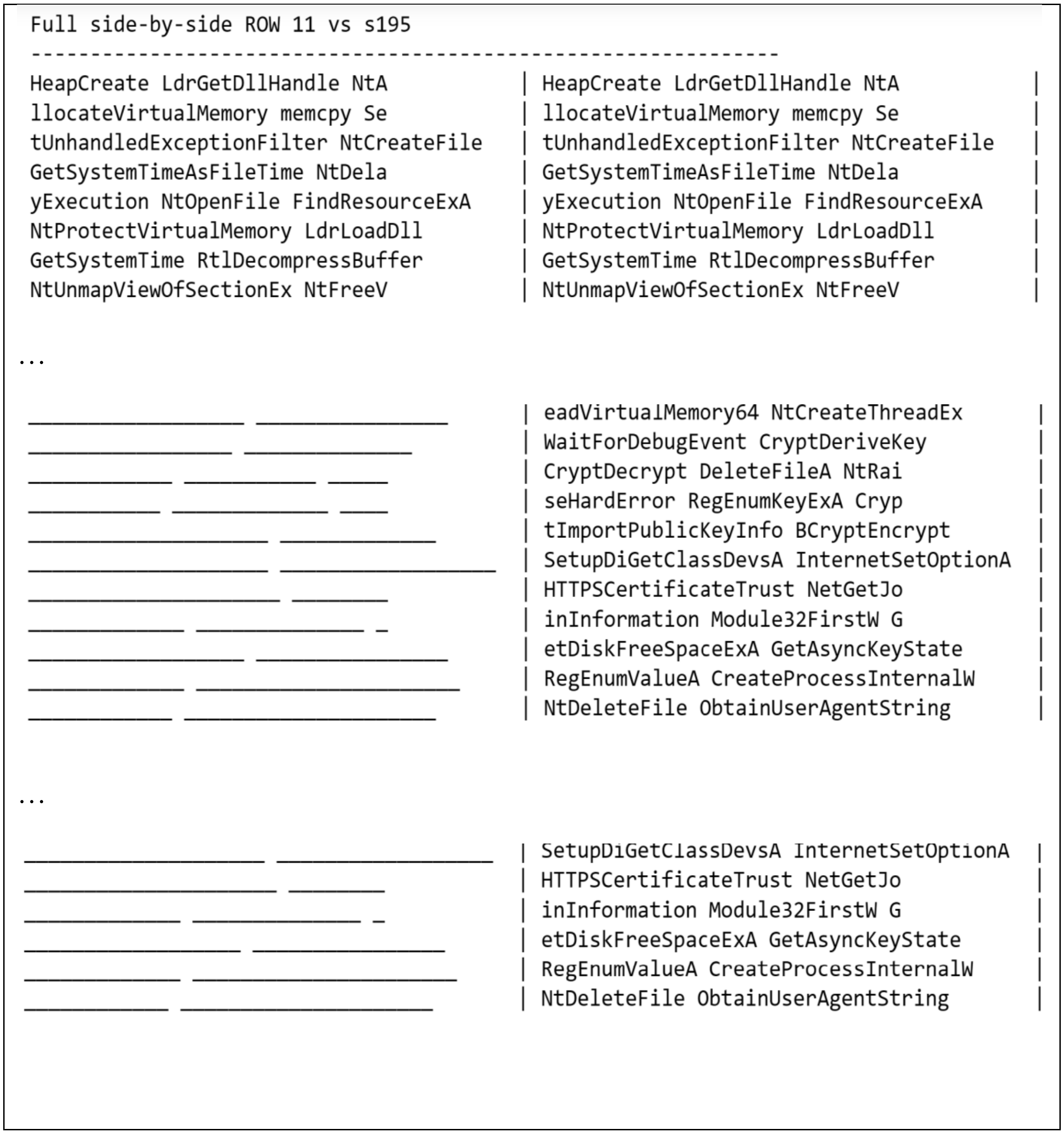
Taking two samples of rows from Figure T, the following comparison is from Cluster 0. It is concluded that this malware is almost identical because there is only one added feature at the bottom of the API calls. The left column of the sequence of the events has one missing object call called "CoGetClassObject." This can infer that this malware is either upgraded from the original version created by the malware author.



```
-------------------------------------------------------------------------------
Full side-by-side  ROW 10 vs ROW 13
-------------------------------------------------------------------------------
HeapCreate LdrGetDllHandle NtA          | HeapCreate LdrGetDllHandle NtA          |
llocateVirtualMemory memcpy Se          | llocateVirtualMemory memcpy Se          |
tUnhandledExceptionFilter NtCreateFile  | tUnhandledExceptionFilter NtCreateFile  |
GetSystemTimeAsFileTime NtDela          | GetSystemTimeAsFileTime NtDela          |
yExecution NtOpenFile FindResourceExA   | yExecution NtOpenFile FindResourceExA   |


…

WinHttpSetOption NtEnumerateValueKey    | WinHttpSetOption NtEnumerateValueKey    |
NSPStartup HttpSendRequestW Nt          | NSPStartup HttpSendRequestW Nt          |
CreateNamedPipeFile gethostname         | CreateNamedPipeFile gethostname         |
getaddrinfo GetAdaptersAddresses        | getaddrinfo GetAdaptersAddresses        |
connect send select recv FindWindowA    | connect send select recv FindWindowA    |
________________                        | CoGetClassObject                        |
```

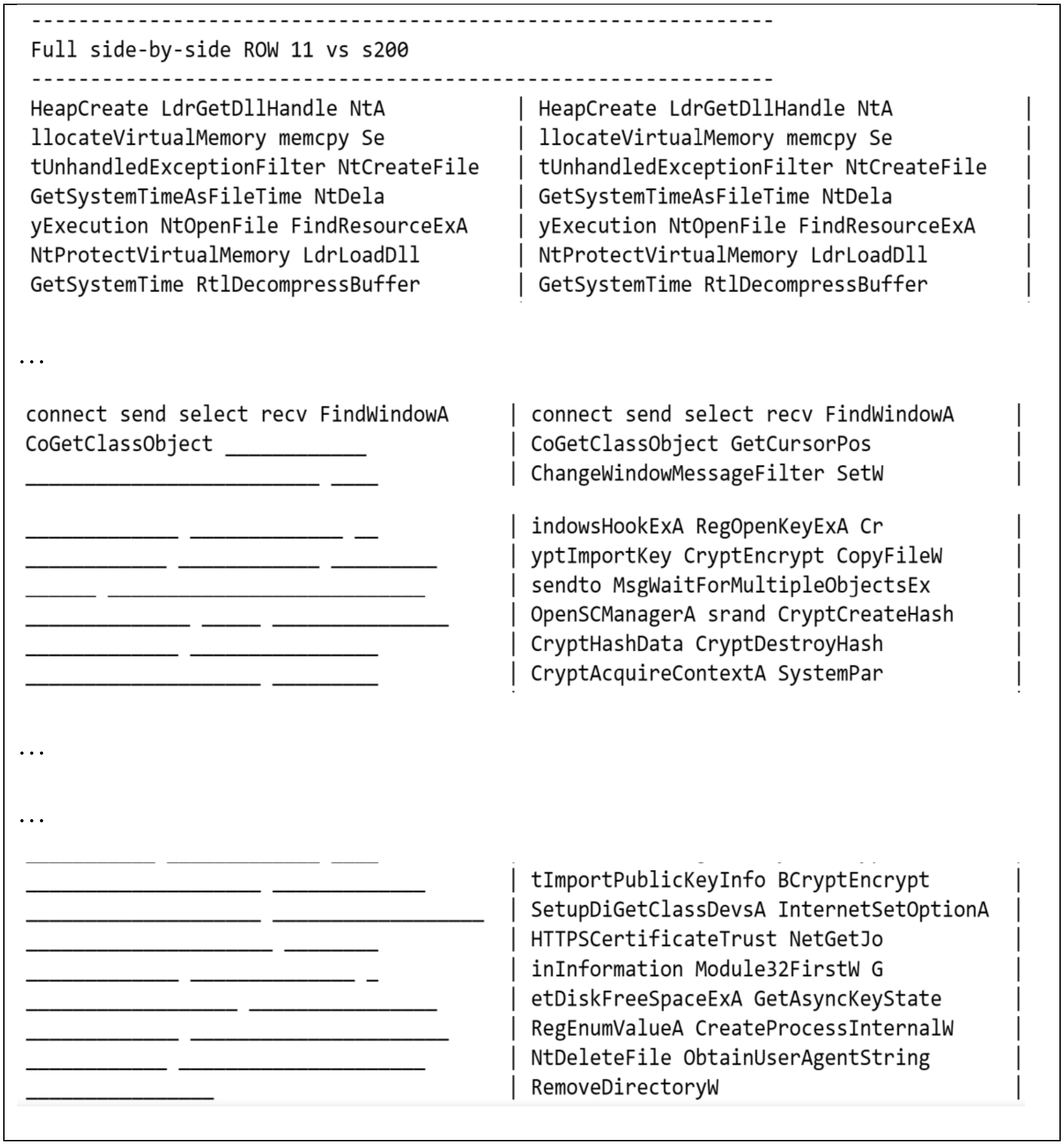
Sample B: Comparison of row 10 and row 11 from Sample A

Reviewing the Sample C and Sample D comparison of different clusters, the difference between the data points belonging to different clusters has significantly increased. Although Sample C and Sample D are identified as True, they are very similar because they belong to different clusters.



```
Full side-by-side ROW 11 vs s195
---------------------------------------------------------------
HeapCreate LdrGetDllHandle NtA           | HeapCreate LdrGetDllHandle NtA           |
llocateVirtualMemory memcpy Se           | llocateVirtualMemory memcpy Se           |
tUnhandledExceptionFilter NtCreateFile   | tUnhandledExceptionFilter NtCreateFile   |
GetSystemTimeAsFileTime NtDela           | GetSystemTimeAsFileTime NtDela           |
yExecution NtOpenFile FindResourceExA    | yExecution NtOpenFile FindResourceExA    |
NtProtectVirtualMemory LdrLoadDll        | NtProtectVirtualMemory LdrLoadDll        |
GetSystemTime RtlDecompressBuffer        | GetSystemTime RtlDecompressBuffer        |
NtUnmapViewOfSectionEx NtFreeV           | NtUnmapViewOfSectionEx NtFreeV           |

…

__________________ ________________      | eadVirtualMemory64 NtCreateThreadEx      |
_________________ ______________         | WaitForDebugEvent CryptDeriveKey         |
____________ ___________ _____           | CryptDecrypt DeleteFileA NtRai           |
___________ _____________ ____           | seHardError RegEnumKeyExA Cryp           |
____________________ _____________       | tImportPublicKeyInfo BCryptEncrypt       |
____________________ __________________  | SetupDiGetClassDevsA InternetSetOptionA  |
_____________________ ________           | HTTPSCertificateTrust NetGetJo           |
_____________ ______________ _           | inInformation Module32FirstW G           |
__________________ ________________      | etDiskFreeSpaceExA GetAsyncKeyState      |
_____________ ______________________     | RegEnumValueA CreateProcessInternalW     |
____________ _____________________       | NtDeleteFile ObtainUserAgentString       |


…

____________________ __________________  | SetupDiGetClassDevsA InternetSetOptionA  |
_____________________ ________           | HTTPSCertificateTrust NetGetJo           |
_____________ ______________ _           | inInformation Module32FirstW G           |
__________________ ________________      | etDiskFreeSpaceExA GetAsyncKeyState      |
_____________ ______________________     | RegEnumValueA CreateProcessInternalW     |
____________ _____________________       | NtDeleteFile ObtainUserAgentString       |
```

Sample C: Comparison of row 11 (cluster 0) and row 195 (cluster 1)



```
---------------------------------------------------------------
Full side-by-side ROW 11 vs s200
---------------------------------------------------------------
HeapCreate LdrGetDllHandle NtA            | HeapCreate LdrGetDllHandle NtA            |
llocateVirtualMemory memcpy Se            | llocateVirtualMemory memcpy Se            |
tUnhandledExceptionFilter NtCreateFile    | tUnhandledExceptionFilter NtCreateFile    |
GetSystemTimeAsFileTime NtDela            | GetSystemTimeAsFileTime NtDela            |
yExecution NtOpenFile FindResourceExA     | yExecution NtOpenFile FindResourceExA     |
NtProtectVirtualMemory LdrLoadDll         | NtProtectVirtualMemory LdrLoadDll         |
GetSystemTime RtlDecompressBuffer         | GetSystemTime RtlDecompressBuffer         |

…

connect send select recv FindWindowA      | connect send select recv FindWindowA      |
CoGetClassObject ____________             | CoGetClassObject GetCursorPos             |
_________________________ ____            | ChangeWindowMessageFilter SetW            |

_____________ _____________ __            | indowsHookExA RegOpenKeyExA Cr            |
____________ ____________ _________       | yptImportKey CryptEncrypt CopyFileW       |
______ ___________________________        | sendto MsgWaitForMultipleObjectsEx        |
______________ _____ _______________      | OpenSCManagerA srand CryptCreateHash      |
_____________ ________________            | CryptHashData CryptDestroyHash            |
____________________ _________            | CryptAcquireContextA SystemPar            |

…

…

____________________ _____________        | tImportPublicKeyInfo BCryptEncrypt        |
____________________ __________________   | SetupDiGetClassDevsA InternetSetOptionA   |
_____________________ ________            | HTTPSCertificateTrust NetGetJo            |
_____________ ______________ _            | inInformation Module32FirstW G            |
__________________ ________________       | etDiskFreeSpaceExA GetAsyncKeyState       |
_____________ ______________________      | RegEnumValueA CreateProcessInternalW      |
____________ _____________________        | NtDeleteFile ObtainUserAgentString        |
________________                          | RemoveDirectoryW                          |
```

Sample D: Comparison of row 11 (cluster 0) and row 200 (cluster 1)

3.4 Ethical Considerations

The undertaken research study does not involve people, medical records, anonymized tissues, or data collection, rendering to be not a violation of the Committee on Publication Ethics (COPE) guidelines, the Institutional Review Board (IRB), or the Ethics Committee of any organization. The malware-shared community repository sources are the only ones that have been measured, analyzed, and published. This study has no conflicts of interest, including personal, political, academic, or financial interests. As a result, it has no bearing on the study's methodology or results. The appendix and reference section contains all of the references and statistical summaries for the sources used in this paper.

3.5 Results and Discussion

Artificial-Based Intelligence (ABI) pattern of life techniques is not new because they have been theoretically discussed in military research organizations. It has not been fully developed or applied to its full potential. As a result, this research uses the ABI methodology process using quantitative analysis using DBSCAN machine learning to establish a pattern of life. This quantitative analysis demonstrated the full potential in identifying the similarities, differences, and anomalies of a single type of malware collected over the years from the research repository. The result of the study identified 8 clusters of ransomware. Each group of clusters is closely similar in terms of API function usages, encryption calls, and sequence API calls. Despite having 8 clusters and outliers, they all have a degree of similarity, leading to the conclusion that they derived from the same source. This research is crucial because it leads us to understand the primary behavior of ransomware. The methodology could be used on other malware types to find the patterns. Previous research has never applied the pattern of life using DBSCAN with unlabeled data, so performing this research is a novel approach to identifying specific malware behavior. In this case, the encryption function, file manipulation, and random files generated are created during the malware run. The literature collected in this research has only applied the identification of true or false malware signatures based on the behavior. However, they have not drilled down to the core of differences and similarities. The specific behavior of the network activities, DLL usages, and any behavioral analysis were never included in any of the studies.

In addition, network analyses are not discussed extensively due to the limited data collection running in the sandbox malware lab. In other words, ransomware rarely uses network communication with the malware creator. Ransomware primarily uses the encryption of files and file API(s) to manipulate files and directories. The research has undergone several iterative processes where other researchers could return to the raw data collected and identify new features to get a new set of clusters.

3.6 Limitations

A quantitative Artificial-Based Intelligence (ABI) methodology framework provides a technique for analyzing various data sources to integrate them into meaningful and coherent insights. Since the world of malware creators is complex and riddled with human and unpredictable behavior with different geographic locations, histories, and cultures, this paper will only partially or entirely predict the goals and intentions of the creators. It can only be deduced from the captured portable executable (PE) malware from the research community, such as MalwareBazaar.com and VirusShare.com. This research only focuses on several sources (static, behavioral, and network data) where future research could add more data sources for more accurate descriptions of events for intelligence gathering. In addition, since the data being used is not labeled, the target class does not require to have to match the ratio of observations in each cluster, homogeneity score, Rand Index (RI), Completeness, Precision & Accuracy, V-measure, and Adjusted Mutual Information (AMI). The only validation used in this research is the Silhouette metric, which combines ideas of cohesion and separation of the clusters.

3.7 The Researcher

The researcher was trained in conducting technical analysis, feature engineering, machine learning, and analysis. The proposed scheduled timeline gives ample time to build the researcher's skills, familiarity, and knowledge of the application of Machine Learning (ML), Feature Engineering, and Architectural Setup and conduct more research to make the work efficient and accurate during the lifetime of research. In addition, for the last 20 years, the researcher has worked as a programmer, software engineer, database administrator, system administrator, technical analyst, and network administrator in the private and public sectors. To this date, he works with the Department of Defense (DoD) as a Cybersecurity IT Specialist and Assessor. The researcher's task is to conduct Cybersecurity Assessments tasking on Defense Industrial Base (DIB) to assess their compliance with NIST SP 800-171A and Cybersecurity Maturity Model Certification (CMMC), a mandated DFARS contract with the DoD. The researcher holds a BS in Computer Science, BS in Criminal Justice, and a Minor in Mathematics. In addition to the undergraduate degree, the researcher has MS in Software Engineering and MS in Database Management and Administration, along with professional industry certifications in Oracle DB Processional, MS SQL Professional, Security+, SANS GSLC, Red Hat Linux Professional certificate, and Defense Acquisition Workforce Improvement Act (DAWIA III). The researcher acquired training to conduct technical research and documentation on different technologies in different platforms necessary before deployment and implementation into the company's Information Technology infrastructure.

CONFLICT OF INTERESTS

Because the study does not involve people, medical records, or anonymized tissues, the data collected for this study does not violate the Committee on Publication Ethics (COPE) guidelines, the Institutional Review Board (IRB), or the Ethics Committee of any organization. The malware-shared community repository sources are the only ones that have been measured, analyzed, and published. This study has no conflicts of interest, including personal, political, academic, or financial interests. As a result, it has no bearing on the study's methodology or results. The appendix and reference section contains all of the references and statistical summaries for the sources used in this paper.

CHAPTER 4

RESULT

The research has explored several phases of the ABI methodology process in depth, from data collection, discovery, assessment, explanation, anticipation, and report delivery, as described in Figure C. The sample data, a 1300 ransomware, has been placed in different clusters, where each group of clusters has significant similarities in terms of method of operations statically and dynamically. It was found that it there are identified eight clusters that answer the research questions being sought. The statistical similarities, differences, anomalies, and baselines of detailed outcomes are listed in the Anticipation Phase, where each research question is narrated in detail to address the descriptive analysis from the Explanation Phase, the statistical feature analysis. The explanation phase, the fourth phase, of the ABI methodology describes the malware behavior patterns, which derives from the Assessment Phase, where Principal Component Analysis (PCA) and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) are tools to parse the ransomware feature data from CAPEv2 malware sandbox. The following summary statement below describes the accomplishments of the research questions using the tools and methodology process.

- Phase 0: All the collected data over the years did not have a specific date when they were collected and integrated into MalwareBazaar.com or VirusShare.com. The 1700 ransomware ingested into the CAPEv2 malware sandbox output static, dynamic, and network behavioral data. During data analysis preparing for feature engineering, several hundred rows were deemed not helpful to go through the process because of missing or null values. Some data did not produce dynamic behavioral results, rendering it not running successfully. The output JSON format is extracted and converted into comma-delimited files for machine-learning processing.

- Phase 1: The Discovery Phase is the engineering phase of the features. The malware features are correlated using Heatmap, which is used to derive the Principal Component Analysis (PCA). There are 27 features extracted as part of feature engineering drawn into the heatmap and reduced to 3-column features.
- Phase 2: The Assessment Phase tool uses the three features to create patterns. There are eight discovered patterns, which are clusters. Clusters indicate that the ransomware belonging to the group is more similar to the other ransomware from the other group. The significance of the clusters is that having 1300 ransomware collected from the wild produces eight variations of ransomware, endangering the Information Technology (IT) industry from being locked out of the computer systems.
- Phase 3: The Explanation Phase gives a descriptive statistical ransomware analysis. The patterns of the line graphs show the similarities, differences, and outliers described in detail using a supervenn diagram indicating their unique behavior, table columns description of each result, and bar charts of statistical occurrences. Based on the patterns and trends, all eight clusters show similar significant patterns indicating that they came from the same source. The 1300 ransomware are variants modification with several updates.
- Phase 4. The Anticipation Phase answers each research question in detail. Summarizing all three questions concludes that ransomware has significantly similar characteristics compared to the difference and the outliers, rendering them to be copied from the same source. The encryption, API file manipulation function, sequence of API calls, and the DLL file used to indicate their similar behavior pattern using the line graphs and supervenn diagram.
- Phase 5. The Delivery Phase addresses the conclusion or outcome of the research. Out of 1300 ransomware samples from Malware Bazaar and Virus Share repository sites, eight variants are clustered into groups using the combinational of Principal Component Analysis (PCA) and DBSCAN. Drilling into the detail of each cluster shows the significant similarities of all the

clusters in the behavioral and static analysis. They share unique usage of application programming interfaces from encryption, file management, and dynamic link library. In addition, the research was able to isolate outliers that are remotely different from the rests of the groups.

In conclusion, out of 1300 ransomware samples, eight possible variants need to be recognized by research communities using the Artificial-Based Intelligence methodological process with DBSCAN machine learning analysis. Based on static, dynamic behavior, and network analysis, there are four overall captured common data occurences of ransomware characteristics: encryption, API usages, file manipulation, and dropped files. They represent the most significant similarities rendering them to have come from the same source of malware author, as shown in Table 4 below.

Profile		**Total API Call**	**Unique**
Encryption [Figure Q]			
	0	2203	16
	1	7341	20
	2	8932	20
	3	1606	20
	4	1012	20
	5	2002	20
	6	154	20
	7	154	20
	-1	6	2
Overall API Usages [Figure P]			
	0	32182	240
	1	90144	269
	2	111204	277
	3	20444	282
	4	13006	283
	5	25753	283
	6	1986	284

	7	1988	284
	-1	805	125
File Manipulation [Figure S]			
	0	3162	26
	1	8332	26
	2	10429	26
	3	1898	26
	4	1196	26
	5	2366	26
	6	182	26
	7	182	26
	-1	106	15
DLL [Figure U]			
	0	1124	49
	1	2334	49
	2	2908	71
	3	486	71
	4	324	71
	5	610	71
	6	32	71
	7	38	71
	-1	26	13
Dropped Files [Figure W]	Files are randomly dropped.		

Table 4: Concluded summary of ABI profile tracking

Having Table 4 showing the final output data findings, which can be synthesized to develop reports of possible changes in the behavior of the ransomware target malware. From the initial process to the report, fact-finding gives the ability to drill down into bits and pieces of the behavior of the target entity. With the current finding, Aritificial-Based Intelligence could be used as a practical fact-finding approach for intelligence gathering, as stated,” *Intelligence delivery* is the ability to develop, tailor, and present intelligence products and services according to customer requirements and preferences [3]“. The requirement for the research is to target specific malware. However, regardless of the type of malware, the process is the same.

Contribution

- Artificial-Based Intelligence (ABI) has conceptually and theoretically demonstrated the process in journals and articles about intelligence gathering to identify anomaly behavior. However, it has never been used extensively in applying malware classification to identify similarities and differences. As a result, the main contribution is that the research uses the process in practice and application of identifying the malware pattern of life of all the samples that can differentiate how closely related samples are. The usefulness of the process and methodology of ABI gives deep insight into the malware being studied.
- DBSCAN is a machine learning that has been taught in an academic paper, but it has never been used in practice. The paper uses the DBSCAN to apply malware behavior to identify the clusters of the ransomware. It demonstrates the effectiveness of using silhouette scoring, where it is able to isolate samples into clusters that are closely related.
- The study has undergone to identify the baselines, similarities, differences, and anomalies of the ransomware as the target malware. It could be used similarly to the same framework analysis of the paper for other researchers. In conclusion, the ransomware, using DBSCAN, shows three concrete methods of operations being used – encryptions described in Figure S, Figure T for file usages, Figure V for DLL most commonly used, and finally, the random usage of the dropped files.

CHAPTER 5

CONCLUSIONS

The research has undergone several phases using Artificial-Based Intelligence (ABI) methodology from data collection and analyzing events of each malware's static, network, and dynamic behavior. The research fulfilled the strategy model to flesh out research activities, giving a methodology of intelligence gathering of malware-type behavior. The research has recognized the usefulness of the practical application of the ABI. Based on the descriptive analysis derived from the clustered data shows the profile data differences and similarities. The data was parsed, interrogated, and discovered for their meaning, where the decisions on the ransomware display the differences and similarities and communicate the findings. There is good reason to believe that although ransomware belongs to different cluster groups, they are derived from the same copy source. Each event was analyzed to discover patterns' similarities and differences, outliers, and baseline behaviors. There are several static and behavioral baseline functions that the target malware (ransomware) identified – the encryption API function, the file library function, DDL, the sequence of call events of each malware, and random-generated files dropped during the malware run. However, the network analysis data did not give enough values to be included in the descriptive analysis, rendering it to be excluded. The research applies machine learning called DBSCAN using Silhouette scoring as the input variable for epsilon and a minimum number of points. The clustering technique is identified as sound segmentation rendering the clustered data more similar than the other cluster belonging to another group. It has been concluded that the research has identified the ransomware's general baselines using the malware's initial similarity features. In short, the significant similarities of all the clusters give the highest possible reason that they came from the same source. The code modifications, mostly in their behavioral application programming, set the malware apart because of the upgrades on the part of the malware author. Other researchers could collect future data related to ransomware and compare the results if the trends' API mode of operation continues to be used and may try to identify other functions besides the established baselines. There may be a myriad of malware behavioral threats that can interrupt the function of the systems with the organizational software systems that have not been discovered when new samples run through the ABI processes.

REFERENCES

REASON FOR RESEARCH REFERENCE

[1] Gross, Geoff A et al. "Application of Multi-Level Fusion for Pattern of Life Analysis." 2015 18th International Conference on Information Fusion (Fusion). ISIF, 2015. 2009–2016. Print.

[2] Brejova, Brona & Dimarco, Chrysanne & Vinar, Tomas & Romero-Hidalgo, Sandra & Holguin, Gina & Patten, Cheryl. (2001). Finding Patterns in Biological Sequences. In: Technical Report CS-2000-22, University of Waterloo.

[3] Atwood, Chandler P. "Activity-Based Intelligence Revolutionizing Military Intelligence Analysis": Forum JFQ 77, 2nd Quarter 2015.

[4] Platas, Linda M. "The Mathematics of Patterns and Algebra." *The Mathematics of Patterns and Algebra | DREME TE*, This Website Is a Project of the Development and Research in Early Math Education (DREME) Network, 2022, https://prek-math-te.stanford.edu/patterns-algebra/mathematics-patterns-and-algebra.

[5] Zhang, Mingrui. "Use Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithm to Identify Galaxy Cluster Members." IOP conference series. Earth and environmental science 252.4 (2019): 42033–. Web.

[6] Yang, Yuan et al. "Fault Diagnosis in Gas Insulated Switchgear Based on Genetic Algorithm and Density- Based Spatial Clustering of Applications With Noise." IEEE sensors journal 21.2 (2021): 965–973. Web.

[7] Creswell, John W.; J. David Creswell. Research Design Qualitative, Quantitative, and Mixed Methods Approaches. Independently. Kindle Edition.

[8] Watkins, Marley. (2018). Exploratory Factor Analysis: A Guide to Best Practice. Journal of Black Psy chology. 44. 009579841877180. 10.1177/0095798418771807.

[9] A. A. Bushra and G. Yi, "Comparative Analysis Review of Pioneering DBSCAN and Successive Den sity-Based Clustering Algorithms," in IEEE Access, vol. 9, pp. 87918-87935, 2021, doi: 10.1109/ACCES S.2021.3089036.

MALWARE ALGORITHM AND FEATURE ENGINEERING REFERENCES

[10] Gao, Yifeng et al. "Adaptive-HMD: Accurate and Cost-Efficient Machine Learning-Driven Malware Detection Using Microarchitectural Events." 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS). IEEE, 2021. 1–7. Web.

[11] Binglin Zhao, Jin Han, and Xi Meng. "A Malware Detection System Based on Intermediate Language." 2017 4th International Conference on Systems and Informatics (ICSAI). IEEE, 2017. 824–830. Web.

[12] Liu, Ya-shu et al. "A New Learning Approach to Malware Classification Using Discriminative Feature Extraction." IEEE access 7 (2019): 13015–13023. Web

[13] Aparicio-Navarro, Francisco J, Jonathon A Chambers, Konstantinos Kyriakopoulos, Yu Gong, and David Parish. "Using the Pattern-of-life in Networks to Improve the Effectiveness of Intrusion Detection Systems." 2017 IEEE International Conference on Communications (ICC) (2017): 1-7. Web.

[14] Happa, Jassim, Thomas Bashford-Rogers, Ioannis Agrafiotis, Michael Goldsmith, and Sadie Creese. "Anomaly Detection Using Pattern-of-Life Visual Metaphors." IEEE Access 7 (2019): 154018-54034. Web

[15] Islam, R et al. "Classification of Malware Based on String and Function Feature Selection." 2010 Second Cybercrime and Trustworthy Computing Workshop. IEEE, 2010. 9–17. Web.

[16] Khan, Muhammad Salman, Sana Siddiqui, and Ken Ferens. "Cognitive Modeling of Polymorphic Malware Using Fractal Based Semantic Characterization." 2017 IEEE International Symposium on Technologies for Homeland Security (HST). IEEE, 2017. 1–7. Web

[17] Daly, T, and L Burns. "Concurrent Architecture for Automated Malware Classification." 2010 43rd Hawaii International Conference on System Sciences. IEEE, 2010. 1–8. Web.

[18] Ye, Yanfang et al. "DeepAM: a Heterogeneous Deep Learning Framework for Intelligent Malware Detection." Knowledge and information systems 54.2 (2017): 265–285. Web.

[19] Tuncer, Turker, Fatih Ertam, and Sengul Dogan. "Automated Malware Recognition Method Based on Local Neighborhood Binary Pattern." Multimedia Tools and Applications 79.37-38 (2020): 27815-7832. Web.

[20] Hellal, Aya, and Lotfi Ben Romdhane. "Minimal Contrast Frequent Pattern Mining for Malware Detection." Computers & Security 62 (2016): 19-32. Web.

[21] Kohout, Jan, Tomáš Komárek, Přemysl Čech, Jan Bodnár, and Jakub Lokoč. "Learning Communication Patterns for Malware Discovery in HTTPs Data." Expert Systems with Applications 101 (2018): 129-42. Web.

[22] Banin, Sergii, and Geir Olav Dyrkolbotn. Detection of Previously Unseen Malware Using Memory Access Patterns Recorded Before the Entry Point. Institute of Electrical and Electronics Engineers (IEEE), 2021. Web.

[23] Fan, Yujie, Yanfang Ye, and Lifei Chen. "Malicious Sequential Pattern Mining for Automatic Malware Detection." Expert Systems with Applications 52 (2016): 16-25. Web.

[24] Li, Hongcheng et al. "Identifying Parasitic Malware as Outliers by Code Clustering." Journal of computer security 28.2 (2020): 157–189. Web.

[25] Yu, Ken F, and Richard E Harang. "Machine Learning in Malware Traffic Classifications." MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM). IEEE, 2017. 6–10. Web.

[26] Ficco, Massimo. "Malware Analysis By Combining Multiple Detectors and Observation Windows." IEEE Transactions on Computers (2021): 1. Web

[27] Ronghua Tian et al. "Differentiating Malware from Cleanware Using Behavioural Analysis." 2010 5th International Conference on Malicious and Unwanted Software. IEEE, 2010. 23–30. Web.

[28] Ghiasi, M, A Sami, and Z Salehi. "Dynamic Malware Detection Using Registers Values Set Analysis." 2012 9th International ISC Conference on Information Security and Cryptology. IEEE, 2012. 54–59. Web.

[29] Ren, Zhuojun, Guang Chen, and Wenke Lu. "Malware Visualization Methods Based on Deep Convolution Neural Networks." Multimedia tools and applications 79.15-16 (2019): 10975–10993. Web

[30] Cordeiro de Amorim, Renato, and Carlos David Lopez Ruiz. "Identifying Meaningful Clusters in Malware Data." Expert systems with applications 177 (2021): 114971–. Web.

[31] Tyng Ling, Yeong et al. "Structural Features with Nonnegative Matrix Factorization for Metamorphic Malware Detection." Computers & security 104 (2021): 102216–. Web.

[32] Pektaş, Abdurrahman, and Tankut Acarman. "Malware Classification Based on API Calls and Behaviour Analysis." IET information security 12.2 (2018): 107–117. Web.

[33] Black, Paul, Iqbal Gondal, Adil Bagirov, and Md Moniruzzaman. "Malware Variant Identification Using Incremental Clustering." Electronics (Basel) 10.14 (2021): 1628. Web.

[34] Yuxin, Ding, and Zhu Siyi. "Malware Detection Based on Deep Learning Algorithm." Neural computing & applications 31.2 (2017): 461–472. Web.

[35] Bai, Jinrong, Junfeng Wang, and Guozhong Zou. "A Malware Detection Scheme Based on Mining Format Information." TheScientificWorld 2014 (2014): 260905-11. Web

[36] Zhang, Hao, Danfeng Yao, Naren Ramakrishnan, and Zhibin Zhang. "Causality Reasoning about Network Events for Detecting Stealthy Malware Activities." Computers & Security 58 (2016): 180-98. Web.

[37] McLaren, Peter, Gordon Russell, and Bill Buchanan. "Mining Malware Command and Control Traces." 2017 Computing Conference. IEEE, 2017. 788–794. Web.

[38] Musgrave, John et al. "Semantic Feature Discovery of Trojan Malware Using Vector Space Kernels." 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 2020. 494–499. Web.

[39] Zhang, Yongchao, Zhe Liu, and Yu Jiang. "The Classification and Detection of Malware Using Soft Relevance Evaluation." IEEE transactions on reliability (2020): 1–12. Web.

[40] Naval, Smita et al. "Employing Program Semantics for Malware Detection." IEEE transactions on information forensics and security 10.12 (2015): 2591–2604. Web.

[41] Liu, Zhicheng et al. "Efficient Malware Originated Traffic Classification by Using Generative Adversarial Networks." 2020 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2020. 1–7. Web.

[42] Kornish, David et al. "Malware Classification Using Deep Convolutional Neural Networks." 2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR). IEEE, 2018. 1–6. Web.

[43] Ki, Youngjoon, Eunjin Kim, and Huy Kang Kim. "A Novel Approach to Detect Malware Based on API Call Sequence Analysis." International Journal of Distributed Sensor Networks 2015.6 (2015): 659101. Web.

DATA SOURCE AND YARA SCRIPTS USED

[44] Kevoreilly. (n.d.). Kevoreilly/CAPEV2: Malware configuration and Payload Extraction. GitHub. Retrieved March 4, 2023, from https://github.com/kevoreilly/CAPEv2

[45] Malware Sample Exchange. MalwareBazaar. (n.d.). Retrieved March 4, 2023, from https://bazaar.abuse.ch/

[46] VirusShare.com. (n.d.). Retrieved March 4, 2023, from https://virusshare.com/

[47] 3vangel1st. (n.d.). 3vangel1st/yara: Yara rules. GitHub. Retrieved March 4, 2023, from https://github.com/3vangel1st/Yara

[48] bartblaze - Overview. (n.d.). GitHub. Retrieved March 5, 2023, from https://github.com/bartblaze

[49] Yara-rules/rules/ransomware at master · bartblaze/Yara-rules. (n.d.). GitHub. Retrieved March 5, 2023, from https://github.com/bartblaze/Yara-rules/tree/master/rules/ransomware

[50] BinaryAlert: Serverless, Real-Time & Retroactive Malware Detection. (2023, March 1). GitHub. https://github.com/airbnb/binaryalert

[51] Genheimer, M. (2023, January 13). yara_rules. GitHub. https://github.com/f0wl/yara_rules

Boldewin, F. (2023, February 17). YARA-rules. GitHub. https://github.com/fboldewin/YARA-rules

[52] Worth, M. (2023, March 5). mikesxrs/Open-Source-YARA-rules. GitHub. https://github.com/mikesxrs/Open-Source-YARA-rules

[53] Roth, F. (2023, March 5). Signature-Base. GitHub. https://github.com/Neo23x0/signature-base

[54] Project. (2023, March 5). GitHub. https://github.com/Yara Rules/rules/blob/master/malware/RANSOM_Petya.yar

[55] ReversingLabs YARA Rules. (2023, March 4). GitHub. https://github.com/reversinglabs/reversinglabs-yara-rules

yara-rules/malware at master · tenable/yara-rules. (n.d.). GitHub. Retrieved March 5, 2023, from https://github.com/tenable/yara-rules/tree/master/malware

[56] TJN. (2022, August 19). yara_repo. GitHub. https://github.com/tjnel/yara_repo

[57] Roth, F. (2023, March 5). Signature-Base. GitHub. https://github.com/Neo23x0/signature-base/blob/master/yara/crime_wannacry.yar

[58] Yara Rules Project. (n.d.). GitHub. Retrieved March 5, 2023, from https://github.com/Yara-Rules

advanced-threat-research/Yara-Rules. (2023, March 3). GitHub. https://github.com/advanced-threat-research/Yara-Rules

[59] gecko984. (n.d.). *Gecko984/supervenn: Supervenn: Precise and easy-to-read multiple sets visualization in Python*. GitHub. Retrieved March 5, 2023, from https://github.com/gecko984/supervenn

[60] *Sklearn.cluster.DBSCAN*. scikit. (n.d.). Retrieved March 5, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

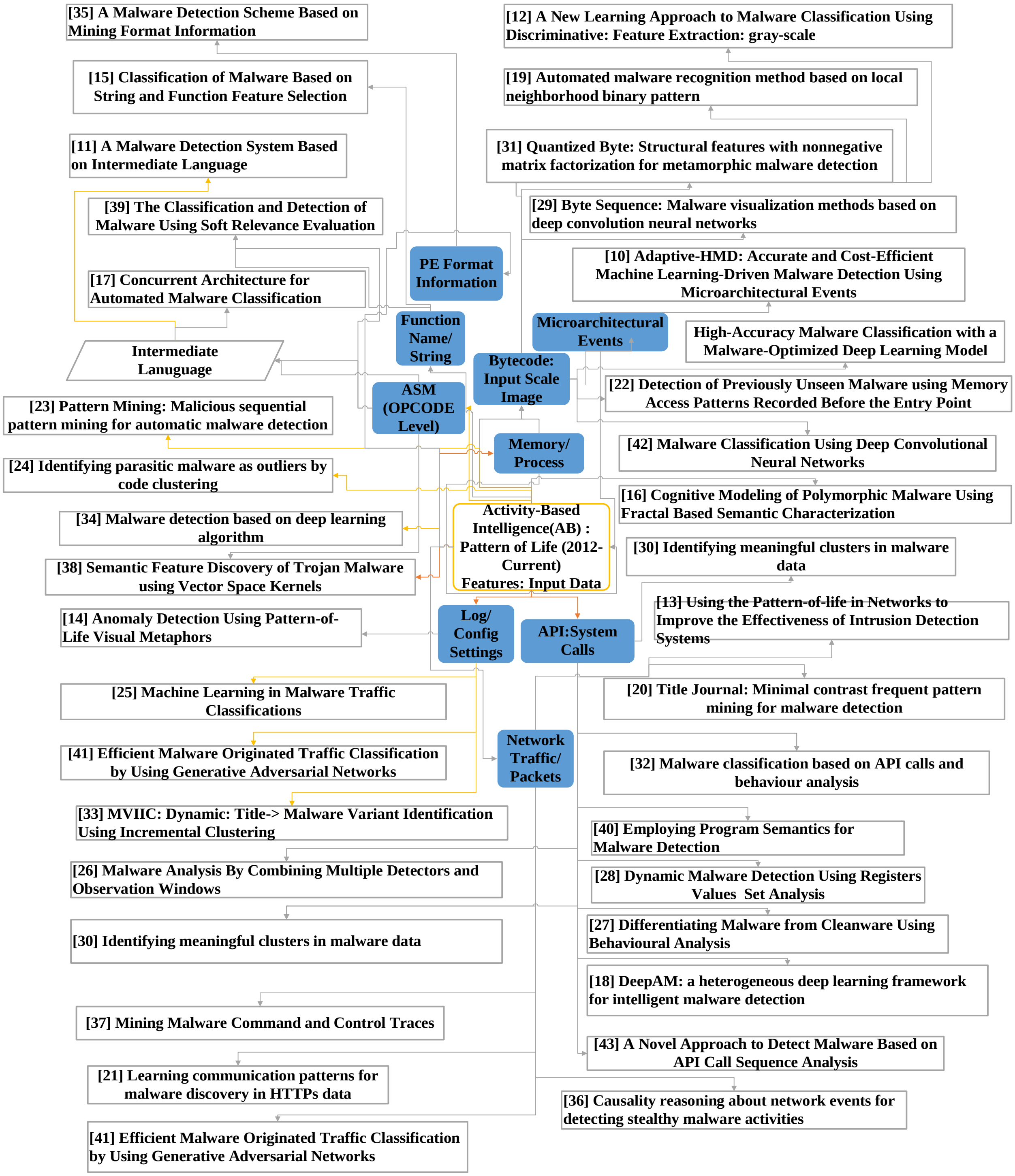
[61] Cangelosi, Richard & Goriely, Alain. (2007). Component retention in principal component analysis with application to cDNA microarray data. Biology direct. 2. 2. 10.1186/1745-6150-2-2.

[62] S. -S. Li, "An Improved DBSCAN Algorithm Based on the Neighbor Similarity and Fast Nearest Neighbor Query," in *IEEE Access*, vol. 8, pp. 47468-47476, 2020, doi: 10.1109/ACCESS.2020.2972034.

[63] Stevewhims, S.W. (no date) *Windows sockets 2 - win32 apps*, *Win32 apps | Microsoft Learn*. Microsoft Documentation. Available at: https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2 (Accessed: April 16, 2023).

APPENDICES

Appendix A. Identified input used for feature engineering for Machine Learning (ML)


[35] A Malware Detection Scheme Based on Mining Format Information
[15] Classification of Malware Based on String and Function Feature Selection
[11] A Malware Detection System Based on Intermediate Language
[39] The Classification and Detection of Malware Using Soft Relevance Evaluation
[17] Concurrent Architecture for Automated Malware Classification
Intermediate Lanuguage
[23] Pattern Mining: Malicious sequential pattern mining for automatic malware detection
[24] Identifying parasitic malware as outliers by code clustering
[34] Malware detection based on deep learning algorithm
[38] Semantic Feature Discovery of Trojan Malware using Vector Space Kernels
[14] Anomaly Detection Using Pattern-of-Life Visual Metaphors
[25] Machine Learning in Malware Traffic Classifications
[41] Efficient Malware Originated Traffic Classification by Using Generative Adversarial Networks
[33] MVIIC: Dynamic: Title-> Malware Variant Identification Using Incremental Clustering
[26] Malware Analysis By Combining Multiple Detectors and Observation Windows
[30] Identifying meaningful clusters in malware data
[37] Mining Malware Command and Control Traces
[21] Learning communication patterns for malware discovery in HTTPs data
[41] Efficient Malware Originated Traffic Classification by Using Generative Adversarial Networks
[12] A New Learning Approach to Malware Classification Using Discriminative: Feature Extraction: gray-scale
[19] Automated malware recognition method based on local neighborhood binary pattern
[31] Quantized Byte: Structural features with nonnegative matrix factorization for metamorphic malware detection
[29] Byte Sequence: Malware visualization methods based on deep convolution neural networks
[10] Adaptive-HMD: Accurate and Cost-Efficient Machine Learning-Driven Malware Detection Using Microarchitectural Events
High-Accuracy Malware Classification with a Malware-Optimized Deep Learning Model
[22] Detection of Previously Unseen Malware using Memory Access Patterns Recorded Before the Entry Point
[42] Malware Classification Using Deep Convolutional Neural Networks
[16] Cognitive Modeling of Polymorphic Malware Using Fractal Based Semantic Characterization
[30] Identifying meaningful clusters in malware data
[13] Using the Pattern-of-life in Networks to Improve the Effectiveness of Intrusion Detection Systems
[20] Title Journal: Minimal contrast frequent pattern mining for malware detection
[32] Malware classification based on API calls and behaviour analysis
[40] Employing Program Semantics for Malware Detection
[28] Dynamic Malware Detection Using Registers Values Set Analysis
[27] Differentiating Malware from Cleanware Using Behavioural Analysis
[18] DeepAM: a heterogeneous deep learning framework for intelligent malware detection
[43] A Novel Approach to Detect Malware Based on API Call Sequence Analysis
[36] Causality reasoning about network events for detecting stealthy malware activities
PE Format Information
Function Name/ String
ASM (OPCODE Level)
Bytecode: Input Scale Image
Microarchitectural Events
Memory/ Process
Activity-Based Intelligence(AB) : Pattern of Life (2012-Current) Features: Input Data
Log/ Config Settings
API:System Calls
Network Traffic/ Packets

Appendix B: Feature Engineering, Level of Scan, Algorithm Used

Reference column maps to the "References" section of MALWARE ALGORITHM AND FEATURE ENGINEERING REFERENCES. Feature engineering is a technique to select the right features for the model. The "Level of Scan" is where the malware is running, from which features of the malware are being extracted. The technique column describes whether machine learning (ML) or other algorithms are used to detect malware.

REF	Feature Engineering	Scan Level/Data Conversion	Technique (Algorithm/Machine Learning (ML))	Task [Dynamic/Static]
10	Monitor HPC (Hight Performance Counter) HPC registers	Monitor microarchitectural events by built-in Hardware Performance Counter (HPC)	Adaptive-HMD (Hardware Malware Detection)	Malware Detector/Dynamic
11	API CALLS/OPCODE	LSTM (Long Sort Term Memory)	MDIL (Malware Detection Using Intermediate Language)	Malware Detector/Dynamic
12	Image-based descriptors	Binary Images	Bag of Visual Words (BoVM) model	Malware Detector/Static
13	Network Traffic Packets	Port Scans, HTTP streaming	Robust PCA	Malware Detector/Static
14	Output/log patterns	Patterns of raw logs, or to visualize the output of detection systems	Visualization Techniques (color & position on the screen). Pattern-of-Life Visual Metaphors	Malware Detector/Dynamic
15	Full-Length Frequency (FLF) Printable String Information (PSI)	Functional Length/String Methods	Classification Algorithm (Pattern Recognition Algorithm) - k-fold cross validation	Malware Detector/Dynamic
16	Edge Graph	Process Monitoring: process tree-based temporal directed graph.	Fractals and Correlation/Dimension	Malware Detector/Dynamic
17	Code Structure LOOPS/Ifs	Loop constructs	Classification Algorithm (Uses Concordia)	Malware Detector/Dynamic

18	Programming interface (API) calls extracted from the portable executable (PE) files.	API CALLS	DeepAM: heterogeneous deep learning framework for intelligent malware detection	Malware Detector/Dynamic
19	3×3, 5 × 5, and 7 × 7 sized blocks - feature concatenation	Bytecode	Local Neighborhood Binary Pattern (LNBP), Neighborhood component Analysis (NCA), Principal Component Analysis (PCA)	Malware Detector/Static
20	API Calls	Symantec Signature	MCFSM (Minimal contrast frequent pattern mining for malware detection)	Malware Detector/Static
21	Timestamp, Username, URL Referer, User-Agent, Duration Bytes up, Bytes down MIME-type	HTTP Logs	Random Forests Neural Networks Gradient tree boosting	Malware Detector/Dynamic
22	1M of memory access operations. Each sequence is later split into the set of overlapping n-grams of the size n=96: memory access patterns	Memory Traces Before Entry Point (BEP)	Random Forest and k-nearest Neighbors	Malware Detector/Dynamic
23	micro-level features	Instruction Sequence	MSPMD algorithm to get the sequential pattern and classify it using ANN (All-Nearest-Neighbor classifier - failed.	Malware Detector/Dynamic
24	Firmware GUIDs	Unified Extensible Firmware Interface (UEFI)	Code Clustering	Malware Detector/Static
25	HTTP Traffic	Network Traffic	extremely lightweight intrusion detection (ELIDe) system	Malware Detector/Dynamic
26	Combination of generic and specialized Deep Neural Networks (DNNs).	Cuckoo Sandbox – API Sequences, Virtual Address of Monitored Process	Ensemble Approach – API Call Sequence, API Sequence Alignment Detector (ASD), Markov chain detector (MCD) – Multiple Detectors	Malware Detector/Dynamic
27	API Calls	API Calls (Dynamic Behavior)	HCL Technologies – Trace Tool Hooking API	Intercepts API Call
28	API Calls	Memory Contents	Value Set Analysis (VSA)	Malware Detector/Dynamic

29	Uses fractal curves to visualize the one-gram features of byte sequences, i.e. malware files themselves, and distinguishes the printable characters from non-printable ones by different colors	Byte Sequence	Byte Sequence Filling Curve Mapping (SFCM)/ Markov Dot Plot (MDP) Method	Malware Detector/Static
30	Cuckoo extracted number of features from PE	Drive by malware generated features from Cuckoo Sandbox (used Drive-by download malware)	Iterative cluster-dependent feature rescaling (k-means, Ward's method clustering, and DBSCAN clustering comparison.	Malware Detector/Dynamic
31	N-vector Structure Features with Nonnegative Matrix Factorization	Quantize byte data	Nonnegative Matrix Factorization (NMF) Algorithms	Malware Detector/Dynamic
32	API call sequencing mining	API System Calls/Behavioral Analysis (network, files, memories)	Voting Experts Algorithm/Online Learning Algorithm	Malware Detector/Dynamic
33	Network packet, CPU, Process, Ram LOGS – Extracted from Cuckoo Box	API Call Names, Sequences, DNS Record Lookup	Malware Variant Identification Using Incremental Clustering (MVIIC) Yara Pattern Matching Technique	Malware Detector/Dynamic
34	Opcode Sequences	Opcode	Deep Belief Network (DBN)	Malware Detector/Static
35	DLL Function Calls	Mining PE format formation	J48/Random Forest, Adaboost, Bagging	Malware Detector/Static
36	Pairwise features (Relation between nodes)	HTTP Traffic/User Inputs	Triggering Relation Graph (TRG)	Malware Detector/Dynamic
37	Network Traffic header, Payload, data flow	Network Level C2 Traffic	Rapid Miner	Malware Detector/Dynamic
38	Opcode	ASM	Vector Space Model and Kernel Methods	Malware Detector/Static
39	FileProps (Size of each ASM), ASM Contents, ASM Statistics, Block Size Distribution, OldNgram, FullineByte (frequency Distribution)	S-Value	Soft Reference Value (s-value)	Malware Detector/Static

40	API System Call Sequence	Semantically Relevant Path	Asymptotic Equipartition Property (AEP)	Malware Detector/Static
41	Normal vs Abnormal Traffic	HTTP Traffic	Traffic Generative Adversarial Network (GAN)/Deep convolutional Neural Network	Malware Detector/Dynamic
42	Visualization Image Similarities	Convert Binary Hex Files	Malware Classification using Deep Convolutional Neural Networks (CNN)	Malware Detector/Static
43	API Call Sequence	API Calls	APIMDS - API Malware Detection System	Malware Detector/Dynamic

Appendix C: All API Calls

There are 240 identified features that show similarities among the 8 clustered groups. The table below displays all API Calls.

WSARecv, NtOpenKeyEx, NtDeleteAtom, OpenServiceA, RegDeleteKeyW, NtQueryInformationThread, OpenServiceW, SHGetFileInfoW, NtDelayExecution, WinHttpSendRequest, LdrLoadDll, SslDecryptPacket, SetWindowsHookExW, gethostname, NtQueryDirectoryFile, IsDebuggerPresent, InternetOpenUrlA, RegQueryValueExA, GetDiskFreeSpaceExW, GetSystemTimeAsFileTime, NtProtectVirtualMemory, NtQuerySystemInformation, NtQueryFullAttributesFile, NtSetInformationFile, WSASendTo, CopyFileA, GetSystemTime, MoveFileWithProgressTransactedW, RegCreateKeyExW, GlobalMemoryStatusEx, InternetCloseHandle, NtDeleteValueKey, BCryptImportKey, RegDeleteValueW, CreateDirectoryW, NtSetInformationThread, NtTerminateThread, WSASend, HttpQueryInfoA, SetupDiGetClassDevsW, RegEnumValueW, GetKeyboardLayout, RtlAddVectoredExceptionHandler, FindResourceExA, RegQueryValueExW, WinHttpSetOption, GetComputerNameA, RegEnumKeyW, WinHttpReceiveResponse, CryptDecodeObjectEx, NtSetValueKey, ChangeWindowMessageFilter, InternetOpenW, OutputDebugStringA, InternetOpenUrlW, bind, NtReadVirtualMemory, NtCreateTransaction, HttpSendRequestA, SHGetKnownFolderPath, NtCreateFile, NtFreeVirtualMemory, shutdown, CryptHashData, NtOpenMutant, DbgUiWaitStateChange, SystemTimeToTzSpecificLocalTime, FindWindowA, GetSystemInfo, NtClose, HttpOpenRequestW, WinHttpOpenRequest, NtAllocateVirtualMemoryEx, NtTerminateProcess, CryptImportKey, ioctlsocket, NtOpenEvent, FindFirstFileExW, UrlCanonicalizeW, CreateTimerQueueTimer, SHGetFolderPathW, CLSIDFromProgID, WSARecvFrom, getaddrinfo, GetLocalTime, GetVolumeNameForVolumeMountPointW, GetSystemMetrics, ShellExecuteExW, WSASocketW, RegOpenKeyExA, GetUserNameA, LsaOpenPolicy, CryptRetrieveObjectByUrlW, RtlDosPathNameToNtPathName_U, GetAddrInfoW, WSAStartup, NtQueryLicenseValue, LockResource, memcpy, GetCurrentHwProfileW, WinHttpOpen, recv, CryptCreateHash, GetAdaptersInfo, CreateToolhelp32Snapshot, NtQueryMultipleValueKey, RtlDecompressBuffer, NtUnmapViewOfSection, WriteProcessMemory, CryptAcquireContextW, Process32FirstW, NSPStartup, connect, NtCreateUserProcess, NtSetContextThread, setsockopt, LdrGetDllHandle, NtQueryAttributesFile, NtOpenDirectoryObject, Process32NextW, RegEnumKeyExW, OpenSCManagerW, GetAdaptersAddresses, RegOpenKeyExW, srand, WinHttpGetProxyForUrl, CoCreateInstance, GetLastInputInfo, NtQueryKey, SetWindowsHookExA, SystemParametersInfoA, CoCreateInstanceEx, WinHttpGetIEProxyConfigForCurrentUser, LdrGetProcedureAddress, NtGetContextThread, NtWriteFile, NtWow64WriteVirtualMemory64, NtOpenKey, GetFileVersionInfo

SizeW, CreateRemoteThread, UnhookWindowsHookEx, CoGetClassObject, InternetOpenA, NtReadFile, HeapCreate, NtOpenFile, ReadProcessMemory, RtlSetCurrentTransaction, closesocket, VarBstrCat, RegQueryInfoKeyW, sendto, WinHttpSetTimeouts, CryptDestroyKey, NtCreateSection, RegSetValueExA, PostMessageW, WinHttpConnect, SaferIdentifyLevel, WriteConsoleW, WinHttpQueryHeaders, NtQuerySystemTime, MsgWaitForMultipleObjectsEx, FindWindowExW, SetUnhandledExceptionFilter, NtOpenSection, socket, GetDiskFreeSpaceW, NtResumeThread, HttpOpenRequestA, NtCreateMutant, select, NtCreateEvent, NtWaitForSingleObject, RasConnectionNotificationW, ControlService, RtlCreateUserThread, SendNotifyMessageW, ConnectEx, CryptDestroyHash, NtDeviceIoControlFile, InternetConnectA, NtEnumerateKey, PostMessageA, DeleteFileW, NtFindAtom, NtQueryInformationFile, FindWindowW, DeviceIoControl, gethostbyname, SizeofResource, NtSetInformationProcess, NtWriteVirtualMemory, NtQueueApcThread, URLDownloadToCacheFileW, NtWow64ReadVirtualMemory64, send, HttpAddRequestHeadersW, LookupPrivilegeValueW, OpenSCManagerA, CryptAcquireContextA, VirtualProtectEx, GetUserNameW, CreateThread, HttpSendRequestW, CopyFileW, RegSetValueExW, GlobalMemoryStatus, GetCursorPos, InternetConnectW, NtCreateKey, LoadResource, NtEnumerateValueKey, NtSetTimerEx, CryptEncrypt, SslEncryptPacket, RegCloseKey, NtAllocateVirtualMemory, GetComputerNameW, NtReleaseMutant, CryptGenRandom, NtQueryInformationAtom, SystemParametersInfoW, NtOpenThread, NtYieldExecution, NtCreateNamedPipeFile, RegNotifyChangeKeyValue, CryptGenKey, CryptExportKey, NtMapViewOfSection, NtUnmapViewOfSectionEx, NtDuplicateObject, __anomaly__, NtAddAtomEx, NtQueryValueKey, NtOpenProcess, NtDeleteKey, GetFileVersionInfoW, NtSuspendThread, RegCreateKeyExA

Appendix D: Imported DDL Similarities

There are 49 Dynamic Link Libraries (DDL) that share similarities of all 8 clusters.

OLEAUT32, DBGHELP, MSACM32, DINPUT8, WININET, UXTHEME, SHELL32, IMM32, NETAPI32, SECUR32, NTDSAPI, WSOCK32, CRYPT32, WINMM, NDDEAPI, ACTIVEDS, COMDLG32, USERENV, MSVCRT, IPHLPAPI, IMPORTS NT, URLMON, OLEACC, ADVAPI32, NT, WS2_32, MSIMG32, COMCTL32, PDH, SHLWAPI, OLE32, SETUPAPI, VERSION, MODEMUI, KERNEL32, USER32, MPR, OLEDLG, RSTRTMGR, WINSPOOL, WTSAPI32, GDI32, PSAPI, GDIPLUS, BCRYPT, OPENGL32, IMPORTSKERNEL32, GLU32, CERTCLI

Appendix E: 159 API files that are not found in the common features

WSARecv, NtDeleteAtom, OpenServiceA, RegDeleteKeyW, OpenServiceW, SHGetFileInfoW, PStoreCreateInstance, WinHttpSendRequest, DeleteFileA, SslDecryptPacket, SetWindowsHookExW, gethostname, InternetOpenUrlA, NtLoadKeyEx, RegQueryValueExA, GetDiskFreeSpaceExW, NetUserGetInfo, GlobalMemoryStatusEx, InternetCloseHandle, NtDeleteValueKey, BCryptImportKey, RegDeleteValueW, NtSetInformationThread, NetGetJoinInformation, WSASend, HttpQueryInfoA, SetupDiGetClassDevsW, RtlAddVectoredExceptionHandler, FindWindowExA, WinHttpSetOption, WinHttpReceiveResponse, GetComputerNameA, HttpQueryInfoW, CryptDecodeObjectEx, ChangeWindowMessageFilter, InternetOpenW, OutputDebugStringA, InternetSetOptionA, InternetOpenUrlW, NtCreateTransaction, HttpSendRequestA, accept, WSAConnect, shutdown, CryptHashData, CryptImportPublicKeyInfo, HTTPSFinalProv, SystemTimeToTzSpecificLocalTime, FindWindowA, HttpOpenRequestW, CryptDecrypt, WinHttpOpenRequest, NtAllocateVirtualMemoryEx, GetAsyncKeyState, CryptImportKey, ioctlsocket, FindFirstFileExW, DnsQuery_A, UrlCanonicalizeW, CopyFileExW, CLSIDFromProgID, geta ddrinfo, GetLocalTime, RegOpenKeyExA, GetUserNameA, LsaOpenPolicy, CryptRetrieveObjectByUrlW, GetAddrInfoW, InternetCrackUrlW, HttpAddRequestHeadersA, GetCurrentHwProfileW, WinHttpOpen, recv, CryptCreateHash, GetAdaptersInfo, NtQueryMultipleValueKey, NSPStartup, InternetGetConnectedState, Module32NextW, CryptDeriveKey, connect, NtDeleteFile, NtSetContextThread, OpenSCManagerW, GetAdaptersAddresses, RegDeleteKeyA, srand, WinHttpGetProxyForUrl, SetupDiGetClassDevsA, RemoveDirectoryW, COleScript_ParseScriptText, SetWindowsHookExA, SystemParametersInfoA, WinHttpGetIEProxyConfigForCurrentUser, NtRaiseHardError, NtGetContextThread, UnhookWindowsHookEx, InternetOpenA, CoGetClassObject, VarBstrCat, WinHttpSetTimeouts, sendto, CryptDestroyKey, OutputDebugStringW, WinHttpConnect, SaferIdentifyLevel, WriteConsoleW, WinHttpQueryHeaders, MsgWaitForMultipleObjectsEx, FindWindowExW, socket, GetDiskFreeSpaceW, HttpOpenRequestA, select, RasConnectionNotificationW, ControlService, SendNotifyMessageW, CryptDestroyHash, InternetConnectA, RegEnumKeyExA, PostMessageA, DeleteFileW, FindWindowW, gethostbyname, CreateProcessInternalW, NtQueueApcThread, URLDownloadToCacheFileW, HTTPSCertificateTrust, ObtainUserAgentString, recvfrom, NtWow64ReadVirtualMemory64, send, FindResourceExW, Module32FirstW, HttpAddRequestHeadersW, OpenSCManagerA, CryptAcquireContextA, InternetCrackUrlA, HttpSendRequestW, CopyFileW, GetDiskFreeSpaceExA, RegEnumValueA, GetCursorPos, InternetReadFile, InternetConnectW, NtEnumerateValueKey, CryptEncrypt, WaitForDebugEvent, SslEncryptPacket, listen, NtCreateThreadEx, NtCreateNamedPipeFile, CryptGenKey, CryptExportKey, RegNotifyChangeKeyValue, BCryptEncrypt, RegDeleteValueA, NtDeleteKey, NtSuspendThread