TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Martin Mihalovic 223593IVCM

GENETIC MALWARE ANALYSIS

Master's Thesis

Supervisor: Alejandro Guerra Manzanares
PhD

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Mihalovic 223593IVCM

PAHAVARA GENEETILINE ANALÜÜS

Magistritöö

Juhendaja: Alejandro Guerra Manzanares
PhD

Tallinn 2024

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Martin Mihalovic

12.05.2024

Abstract

Today's Internet landscape witnesses the emergence of thousands of new malware variants daily, necessitating continuous vigilance from security teams. These teams must promptly counteract the evolving Techniques, Tactics, and Procedures (TTPs) embedded within malicious code. This study introduces a proactive approach to collecting and processing malware from the Internet, termed Genetic Malware Analysis. This domain combines the principles of Data Science with Malware analysis to make malware analysis in an on-premise environment more efficient. Genetic Malware Analysis encompasses fundamental static analysis, sample preprocessing via static code analysis, and subsequent analytics employing machine learning techniques. This research introduces a novel interpretation of malware genes and their feature engineering, with experimentation on various configurations of supervised machine learning algorithms. Experimental results demonstrate that the classification accuracy of malware families reaches up to 80%, with testing time reduced by up to ten times. The experiments confirmed that the developed algorithm can process up to 1000 samples within one hour. Additionally, this study contributes research findings in the form of code to an advanced malware analysis platform named MalTraits. Through the MalTraits web application, malware analysts can conduct and customize experiments involving machine learning, visually correlating samples using a 3D graph and percentage-based similarity calculations. Ultimately, this work facilitates the practical application of static malware genes and machine learning across various current malware families.

The thesis is written in English and is 83 pages long, including 8 chapters, 19 figures and 6 tables.

List of Abbreviations and Terms

IR	Incident Response
TLP	Traffic Light Protocol
IOC	Indicators of Compromise
APT	Advance Persistence Threat
CISC	Complex Instruction Set Computer
ML	Machine Learning
CERT	Computer Emergency Response Team
EP	Entry Point
PE	Portable Executable
OS	Operating System
ELF	Executable and Linkable Format
IL	Intermediate Language
SIEM	Security Information and Event Management
CBA	Cost-Benefit Analysis
IAT	Import Address Table
TTP	Tactics, Techniques and Procedures
CFG	Control Flow Graph
DBI	Dynamic Binary Instrumentation
KNN	K-Nearest Neighbors
LSH	Local Sensitive Hashing
FCG	Function Call Graph
AV	Antivirus Program
MFA	Multi-Factor Authentication
RAM	Random-access memory
ORM	Object Relational Mapping
PCA	Principal Component Analysis

Table of Contents

List of Figures

List of Tables

1. Motivation

Malware, sometimes referred to as malicious software (further referred to as malware), constitutes a crucial element of the well-established cybersecurity framework known as the Cyber Kill Chain [1], developed by Lockheed Martin. One of the common objectives of malware analysis is to answer questions about the various capabilities of the malware. Frequent questions on malware analysis are connected with malware capabilities, whether it can communicate to the Internet, damage data, or stay undetected by antivirus software [1].However, uncovering malware capabilities is not the sole objective of malware analysis. In practice, malware analysis is often applied as part of the Incident Response (IR) process, aiming to investigate the incident and attribute the threat actor. Malware analysis should also answer whether the malware was targeted, attribution of the author, or whether malware has already been seen in the past. Answering these questions requires a lot of experienced personnel, time and data. The Independent IT-Security Institute has published that they register more than 450,000 malware specimens per day [2]. Therefore, manual correlation and malware sample attribution represent a major challenge for cybersecurity teams. Moreover because of "Traffic Light Protocol" (TLP), malware specimen from incident cannot be submitted to cloud analyzers such as Virustotal [2]. In such cases, malware analysis teams are often referred to internal capacities and resources in an offline/on-premise environment.

Malware analysis poses a significant time and technical challenge, particularly in Incident Response scenarios where swift answers to critical questions - Who, When, Where, What, Why, and How - are essential. However, malware analysis takes significantly longer, often several weeks. The factor that significantly complicates the analysis process is mainly the offline environment due to high TLP. Many security teams view malware analysis as a reactive service, focusing solely on incident-related samples. This reactive approach proves inefficient, lacking the necessary data to expedite analysis in the offline environment. Cyber defense teams frequently struggle to verify whether similar samples have been encountered or analyzed previously. Consequently, they cannot promptly classify samples as unique, potentially targeted, or just ordinary cyber crime malware. Moreover, these teams typically rely on correlations between samples based on common Indicators of Compromise (IOCs), such as IP addresses, domains, or Mutexes. However, machine code-level correlations offer more excellent utility for detection, correlation, and classification.

[1]https://www.lockheedmartin.com/en-us/capabilities/ cyber/cyber-kill-chain.html
[2]https://www.virustotal.com

This research represents malware similar to genes in the human body. Each malware has its characteristics based on which it is possible to cluster similar entities and identify anomalies quickly. Thus, the primary objective of this study is to bridge the gap between Data Science and Malware Analysis. The first prerequisite for effective "Malware Data Analysis" is access to up-to-date and sufficiently abundant data. The second assumption is to create an efficient format for representing malware samples, such as hash values, arrays of bytes, and images. An essential contribution of this research is introducing a format known as "malware genes," which entails sequences of disassembled basic blocks from binary files. On average, each malware specimen comprises thousands of such genes [3]. Lastly, successful "Malware Data Analysis" necessitates the selection of features conducive to supervised and unsupervised machine learning methods. Thesis contribution extends to enhancing an advanced malware analysis platform named MalTraits. Specifically, this research contributes to malware classification on the MalTraits platform using the k-nearest neighbors (KNN) algorithm and clustering techniques.

The primary objective of this research is to create a solution for malware correlation by applying machine learning algorithms. The research product empowers organizations to automatically gather malware samples from the Internet and assess them within their infrastructure, capable of handling thousands of samples daily. The main reason why such a volume of samples can be analyzed is the effectiveness of static analysis. Static analysis is generally faster and less demanding on resources than dynamic analysis. Applied advanced algorithmic methods such as classifying and clustering malware samples can significantly reduce the time required to identify and initially analyze an unknown sample. This solution addresses the research problem by enabling security teams to establish their own malware repository equipped with advanced analytics. This solution eliminates the need to rely on third-party services on the Internet or cloud.

This research facilitates the application of machine learning models to the dataset of malware genes within an on-premise environment. This approach enhances the efficiency of various processes in malware analysis. These processes include identifying correlations between samples, deciphering code-writing techniques employed by different malware developers, clustering samples into malware families, attributing samples to specific groups or authors, and detecting anomalies. The solution's validation entails applying the research to malicious samples from well-known malware families and Advanced Persistent Threat (APT) groups. By leveraging malware genes and machine learning, it is possible to uncover the techniques employed by malicious code authors. It improves the ability to attribute samples to specific attackers.

1.1 Hypothesis

Proposed solution is grounded in the belief that, even within an on-premise environment, it's feasible to establish a sophisticated malware repository with advanced analytical capabilities. The criterion for 'Malware Data Analysis' is a large number of latest malware samples, and their diversity in the categories of malware families. Many malware samples are uploaded daily to public places on the Internet. Even with the utilization of static analysis and optimized sample representation via malware genes, the analysis process remains complex. Consequently, formulated research hypothesis is as follows:

- **H0**: Using this research solution, it is possible to automatically analyze more than 1000 samples per day.

This research solution encompasses two primary types of analysis: Genetic and Static analysis. Malware genes (refer to Section 3.2) serve as the foundational units for Genetic analysis. Static analysis generates these genes through a process known as disassembling. After disassembling the code, it is split into malware gene blocks, which are subsequently enriched with necessary metadata. However, Static analysis presents limitations, as detailed in the Limitations section (see Section 1.5). Packers compress and safeguard binary files and pose challenges for static analysis. As a result, malware genes are generated solely from the packer program rather than from the program itself being packed. Nevertheless, excluding packed samples enables the establishment of advanced correlations based on malware genetic characteristics. Packed samples will not be disregarded; instead, dynamic analysis will be employed for unpacking. Unpacked samples can be later used in static and genetic analysis. Section 3.3 provides further details on how this procedure operates on the MalTraits platform.

- **H1**: Genetic malware analysis can reveal correlations with great accuracy in malware which is not packed.

The primary hypothesis of this research project is that the application of data science methods in malware analysis can substantially enhance efficiency. Data science involves automation elements such as automated malware fetching, preprocessing, and sample analysis. However, for organizations, these processed samples in the form of malware genes represent a significant advantage in TLP Red/Amber incident cases. Organizations are liberated from manual sample classification and attribution tasks by leveraging a genome database. With a sufficiently high-quality set of malware, organizations can reduce reliance on third-party services.

1.2 Scope

Malware analysis is a vast area that one thesis cannot cover. Computer science recognizes several computer architectures with completely different instruction sets [4]. Notably, there is a growing trend of malware targeting MacOS with ARM architecture or embedded systems with MIPS architecture [5, 6]. Malware genes in this research solution are created by disassembling the binary code and depend on the specific type of instruction set. Research scope is limited to Complex Instruction Set Computer (CISC) architectures, specifically malware designed for x86 and x86-64 architectures. Malware still mostly targets selected architectures, and for this reason, this research decided to focus only on them 1.

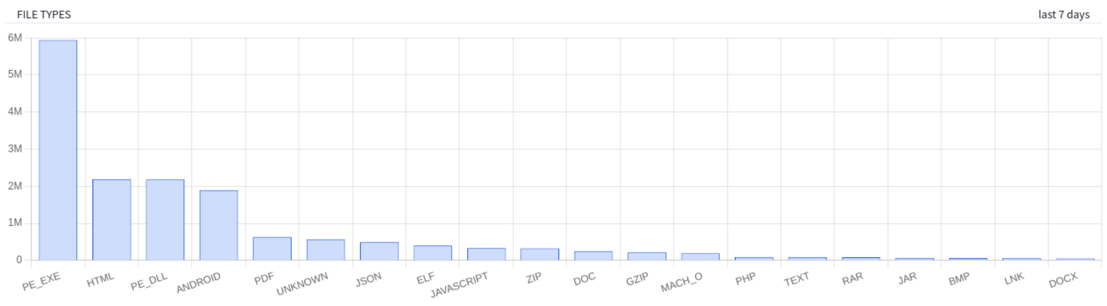


Figure 1. Virustotal submission statistics between 8.3.2024-15.3.2024 [7].

Malware genes rely on the functionality of the disassembler, which falls under the category of static analysis methods. The scope of this research was selected only for malware targeting Intel architectures. The disassembler used in this research works for all types of x86 and x86-64 binary files, regardless of their file type. A binary program is usually written in the header of the file, where the program's executable code is located, often also called Entry Point (EP). The structure of the header, as well as the structure of the binary program itself, varies slightly depending on the file type. Portable Executable (PE) files for Windows Operating System (OS) have a different structure than Executable and Linkable Format (ELF) on Linux OS [8]. Given that PE files represent the most prevalent category among all file types [7], this study exclusively focuses on analyzing PE files.

Considering the complexity of the issue, this research opted to exclude binary programs developed within the .NET framework. The main reason is the different structure of the binary code since it is an Intermediate Language (IL). When compiling .NET programs, 'Microsoft Intermediate Language' (MSIL) is created instead of machine code [9]. It is generally possible to create malware genes from these programs, but this research does not deal with Intermediate Languages such as .NET and Java.

1.3 Research Questions

This thesis continues the research from the bachelor's thesis by Mihalovic et al., who described the possibilities of using malware genes [3]. One of the main problems of the previous solution was the many-to-many comparison, which represented n^2 complexity. In practice, comparing thousands of samples using the many-to-many technique is unfeasible. This thesis aims to create an applied research that will come with a valuable tool for Genetic Malware Analysis. For this reason, developing another, more effective technique than analyzing malware genes was necessary. Machine learning (ML) is one of the possible methods to analyze malware genes more effectively, which makes it possible to analyze a specific volume of samples described in Hypothesis 1.1 every day. The main research question, which forms the essence of this research, is as follows.

How can machine learning be applied to genetic malware analysis?

However, the machine learning process consists of several steps, which are complex in the application to malware genes [10]. For this reason, the main research question is divided into three research subquestions. Pre-processing is one of the initial steps, sometimes called "Data Cleaning" [11]. Genetic malware analysis that applies machine learning is not practical in all cases. Packed malware samples are the main problem when Genetic malware analysis fails. Data cleaning is an initial research problem that is defined by the first sub-research question. The following research problem consists of architecture and feature selection for machine learning. There are several possible representations of machine learning features in malware genes. This issue was defined in the second research sub-question. Finally, one of the main parts of this research is verification and experimentation with machine learning in an application with malware genes. The third research sub-question seeks to identify the machine learning method or algorithm that is most effective for genetic malware analysis.

1. How can the data cleaning process be optimized to exclude packed malware samples?
2. What machine learning features can be extracted from the genetic characteristics of malware?
3. Which machine learning method or algorithm demonstrates the highest suitability for correlating malware?

This research involves analyzing raw data from malware binaries, requiring comprehensive preprocessing to enable machine learning application. This process encompasses normalizing data into malware genes, collecting data through integration with diverse malware repositories, and cleaning data by excluding irrelevant samples. The core focus of this research involves feature engineering of malware genes and testing the resultant machine learning model. When samples indicate poor correlation, conducting a reverse engineering analysis becomes crucial. The research contributes by analyzing the problem, proposing feature engineering for malware genes, and creating functional software to correlate malware samples.

1.4 Novelty

In most research studies, malware genes are typically generated through dynamic analysis [12]. However, this approach doesn't align with the practical need to analyze thousands of malware samples daily. Many studies employ n-gram analysis to interpret malware genes, which introduces significant overhead and lacks efficiency [13].

This research interprets malware genes by utilizing disassembled code segments enriched with analytical metadata. This approach interprets the logical structure of malware as its fundamental unit. To the best of author's knowledge, there hasn't been any application of machine learning to this interpretation of malware genes.

The novelty of this research lies in its examination of the challenges outlined in Section 4.10 regarding malware analysis research and the practical needs of the field. This research introduces the concept of integrating Data Analysis and Malware Analysis. More precisely, it merges a distinctive interpretation of malware with machine learning, termed "Genetic Malware Analysis". This research thesis is motivated by two main factors:

- Malware is not just one indivisible part
- The absence of machine learning integration in malware analysis remains a gap, especially in on-premise environments.

In practical terms, there exists a limited number of open-source tools that integrate both malware analysis and machine learning. The MWDB software [3], developed by the Polish Computer Emergency Response Team (CERT), stands out as one of the top tools for malware analysis, particularly for handling large volumes of malware. However, its primary focus lies in storing malware rather than correlating it. The objective of this applied research is to bridge the gap between practical requirements and academic research. The distinctiveness of this study resides in both the development of novel tools and the capability to analyze malware at scale.

[3]https://mwdb.readthedocs.io

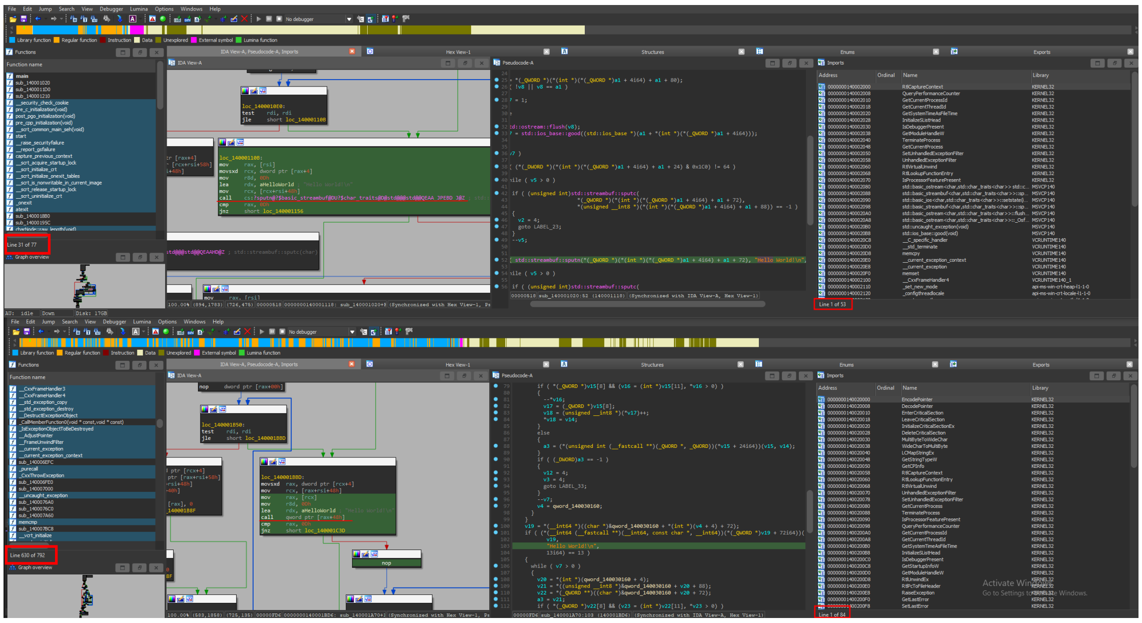


Figure 2. Side to side comparison between statically and dynamically linking of the binary.

1.5 Limitations

This research faces several limitations, primarily concerning the properties of static code analysis (see Section 3.1). These limitations were categorized into two main parts.

- Statically linked binaries
- Obfuscation Techniques, Packers and binary Protectors

During the compilation process, binary programs can be linked statically with libraries, which brings several advantages and disadvantages. From the programmer's point of view, it is an enormous advantage that the binary program does not depend on the libraries on the host device. Developers can compile a statically linked PE file that can be executed on different versions of Windows [14]. For malware authors, the advantage is that static linking significantly complicates the static analysis of the code since the code of the libraries is mixed with the code of the program. Statically linked programs are thus significantly more complex for static code analysis 2, and in such cases, library signatures such as FLIRT [4] can help us. A slight disadvantage of statically linked programs is the much larger size of the program since it also contains a library program. Static linking adds complexity to the research but does not make it impossible to analyze samples. Such solutions for proper distinguishing can be comparing malware gene libraries and analyzing binary programs, writing signatures, or using linear regression. Although this research only briefly touches on this topic, the MalTraits project actively investigates this problem.

[4] https://hex-rays.com/products/ida/tech/flirt/

Packer, protectors, and obfuscation techniques essentially take full-fledged static code analysis. Packer and protector are software designed to protect code, whether for a legitimate or malicious purpose. Packers can be compared to postal envelopes in real life. The envelope protects the letter or the contents that are sent in it. However, the recipient can open it and become familiar with its contents. It is similar to packers, which unpack the code in the runtime process. When the program is not running, the code is protected by various techniques such as encryption, virtualization of instructions, or obfuscation. The MalTraits project integrates static and dynamic approaches to unpack packed samples. However, this research does not delve into an exhaustive analysis of packers or the tools employed for unpacking.

1.6 Research Methods

This thesis adopts a hybrid approach that integrates both Quantitative and Qualitative research methodologies. The objective of this approach is to initially utilize Quantitative Research to gather fundamental information about a vast array of samples across various malware families. The primary sources of data in the form of malware for this research are platforms such as Malpedia [5], MalwareBazaar [6], VirusShare [7] and Vx Underground [8]. The size of the tested datasets ranges from 100 malware samples to 2000. In each experiment of this research, there are at least ten malware families. Through quantitative research, it can be created a malware data model and find correlations between groups of malware families, threat actors and the malware samples themselves. One of the outputs of quantitative research are the statistics of the solution with which accuracy it is possible to classify malware.

A significant role in this research is the qualitative method, which is carried out only after quantitative research. The main objectives of the qualitative method are divided into two parts:

- Analysis of selected samples that were not successfully classified or clustered.
- Verification of the significance of selected malware genes using reverse engineering.

The initial phase of the qualitative research entails conducting a thorough analysis of the malware families to which the sample belongs. Various factors may contribute to misclassification, including the possibility that the sample was packed and the developed

[5]https://malpedia.caad.fkie.fraunhofer.de/
[6]https://bazaar.abuse.ch/
[7]https://virusshare.com/
[8]https://vx-underground.org/

solution failed to detect the packer. Additionally, misclassification may occur due to the close relationship between certain malware families, with significant code overlap observed, as is often the case with sub-families of the "Zeus" family.

The subsequent stage of the qualitative research involves assessing the significance of the correlated malware genes present in the selected samples. This analysis aims to determine whether these genes hold significance and explore methods for automatic identification. In the mentioned qualitative methods, it has been used a professional malware analysis tool called IDA Pro [9].

1.7 Structure Description

Section 2 describes some frequently used terms in this research. Following that, Section 3 theoretically analyzes the most essential parts of the research. This section further elaborates on essential concepts previously examined in previous research, upon which the present study is built [3]. Section 4 is an essential part of the thesis, involving a literature review and selecting primary studies.

The contribution of this research, including code examples, is detailed in Section 5. Following that, Section 6 primarily presents the results of the research verification. The Discussion section (Section 7) summarizes these results and evaluates potential paths of further research.

[9]https://hex-rays.com/ida-pro/

2. Terms and Notations

Throughout this thesis, certain terms are frequently utilized, and it is imperative for the clarity and coherence of the research that their meanings remain consistent. In order to understand the essence of the research, it is crucial to understand the following terms.

Malware is a piece of code that can cause unwanted behavior to the endpoint, such as excessive use of system resources, remote access, destruction or exfiltration of personal data. Detailed features of the malware can be found in the MITRE Att&ck framework [1].

Benign sample is just an ordinary program that is not malicious. Such programs are used for several reasons, such as improving machine learning models for malware analysis. Adding benign samples to the data model can reduce the probability of a false positive rate in detection accuracy [15].

Malware Gene is a sequence of bytes that consists of disassembled parts of instructions. Each malware gene is enriched with metadata such as hash, cyclomatic complexity, and entropy. It recognizes two types of malware genes, the "block" type and the "function" type. It is the basic logical atom with which this thesis work.

Malware Genetics represents a novel approach to malware analysis, enabling automated analysis of binary files through data analytics [16].

Malware analysis is a broad area that is interested in researching the behavior/properties of malicious samples. Malware analysis often uses reverse engineering, which examines binary programs using disassembly or decompilation. From a procedural standpoint, malware analysis of binary files can be categorized into Basic static analysis, Behavioral analysis, and Code analysis [17].

Incident Response is a reactive event to a cyber incident, during which security teams try to mitigate the effects of the attack and analyze the course and consequences of the attack. Incident response usually consists of several sub-areas, such as data acquisition, forensic analysis, and malware analysis [18].

[1]https://attack.mitre.org/matrices/enterprise/

3. Background

In general, actions in security can be divided into three categories: reactive, proactive, and quality management services [19].In this thesis, the primary focus lies on the first two categories. Reactive services already react to a cyber incident and try to investigate it. On the contrary, proactive services take action before an incident occurs to prevent or prepare for an incident. Incident Response (IR) is often considered a reactive service or action. Malware analysis is part of the IR and therefore based on the water flow principle, this could also be considered as a reactive action. However, there are several problems that occur with the "reactive" approach in malware analysis. An example is a situation when an organization receives a TLP Red incident. The sample, as evidence from the incident, cannot be provided to third parties for analysis. Malware analysis should bring answers to questions such as:

- **Sample classification** - It is a well-known malware that is often spread on the Internet or it is a malware that has not been seen yet. The answers to these questions could indicate whether it is, for example, a targeted attack.
- **Malware attribution** - Attribution based solely on malware analysis is challenging. However, when combining information from other areas of IR (such as forensic analysis), it becomes possible to attribute a threat actor.
- **Detection of code reusing** - Code reuse is not a new phenomenon in malware, but it is becoming more frequent. Several malware such as Havoc [1], Covenant [2] or Metasploit [3] are open-source and many other malware developers adopt these parts of the code. Obtaining information about code usage can speed up malware analysis and can also provide important "Threat intelligence" information.
- **Threat actors monitoring** - It is very important for security teams to know who is attacking them. In this case, the consideration extends to simple phishing attacks, which contain malicious attachments, as part of the threat landscape. In this way, every organization can evaluate and create a Threat Model based on malware samples.

These questions are difficult to answer in offline enviornment and without preparation in the form of creating and maintaining a malware dataset. In general, proactive activities in cybersecurity are very important. "Preparation" 3 is the first and one of the four phases

[1]https://github.com/HavocFramework/Havoc
[2]https://github.com/cobbr/Covenant
[3]https://github.com/rapid7/metasploit-framework

of the IR lifecycle in the NIST 800-61 manual [20]. The entire following IR lifecycle depends on this phase. If high-quality monitoring in the form of Security Information and Event Management (SIEM) has been deployed, the attack chain can be detected in the early stages. If the organization is preparing in the area of malware analysis, the "Containment" and "Eradication" phases can be much faster and more efficient. Proactive steps are therefore very important in malware analysis, but the question is what should this "Preparation" phase look like.

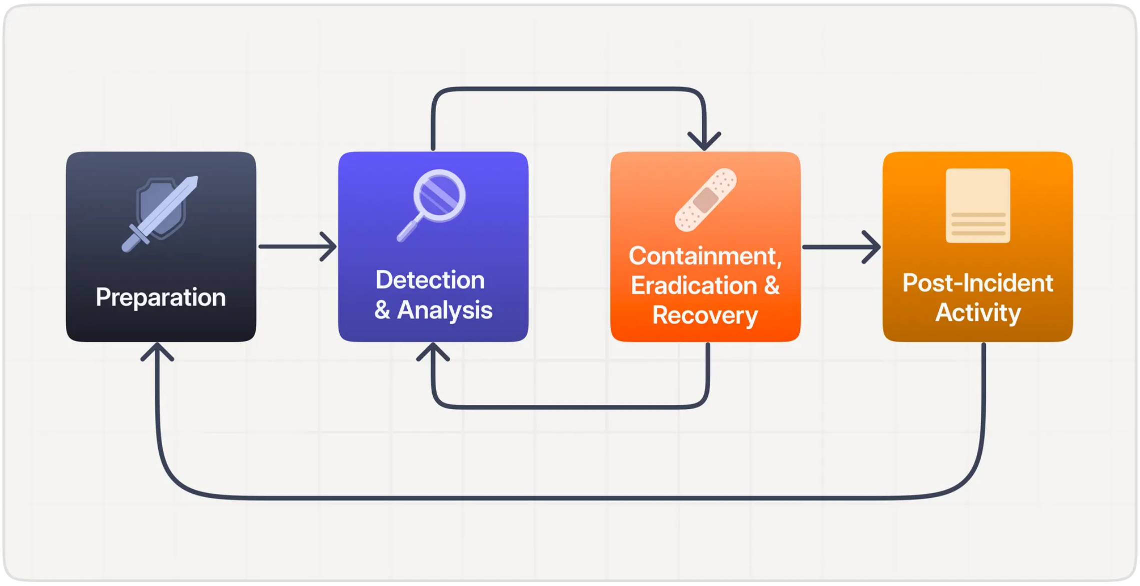


Figure 3. NIST Incident Response Lifecycle [21].

One of the frequently used methods in malware analysis and forensic analysis are Yara rules. The advantage is that these rules can be easily shared. In the case of an incident, it is enough to pull rules into an offline environment, where rules can be used to hunt and sometimes classify malware. However, the general problem with signatures is that they only look for a partial number of binary strings in the binary program. The author of the malware can change, for example, the library or the string, and his or her malware is become undetected. Moreover, based on the research data from Malpedia from March 2024, there is 37,38% malware families, which are not covered by YARA rules [22]. Finally, YARA signatures cannot react to unknown malware families since there are no rules for them.

In proactive malware analysis, it is necessary to collect samples in large volumes from the Internet, partner organizations, or the perimeter of the organization. However, scraping and simply downloading malware samples are not enough. From the point of view of Cost-Benefit Analysis (CBA), the biggest cost is considered to be the space for storing the sample. According to the mentioned statistics, 450,000 malware samples are registered every day [2]. Based on the research experiments described in section 6.2 and 6.3, the average sample size is 592 Kilobytes (kB). Testing dataset in these experiments contains 1916 malware samples. If it were possible to capture all these malware samples, one would need 266GB of storage space every day. Despite the factor that the given amount of malware is unlikely to be caught on a daily basis, this number represents a relatively large cost. The only benefit of this technique is obtaining metadata such as hash or file size.

A hash is a one-way mathematical operation that is often used to verify the integrity of content [23]. One of its key properties of the hash function is the so-called 'Avalanche effect'. This effect represents a phenomenon when the input changes minimally, for example only a single bit changes, so the output of the function is completely different [24]. This operation is very valuable in forensic analysis, but it has no great value in malware analysis. The model that describes what is difficult for the malware author to change is called the "Pyramid of Pain" 4. The hash value in the 'Pyramid of Pain' model is at the bottom in terms of usefulness for detecting and classifying malware. It is trivial task for an attacker to change an arbitrary parameter at the level of the entire sample, which is resulted in completely different hash.

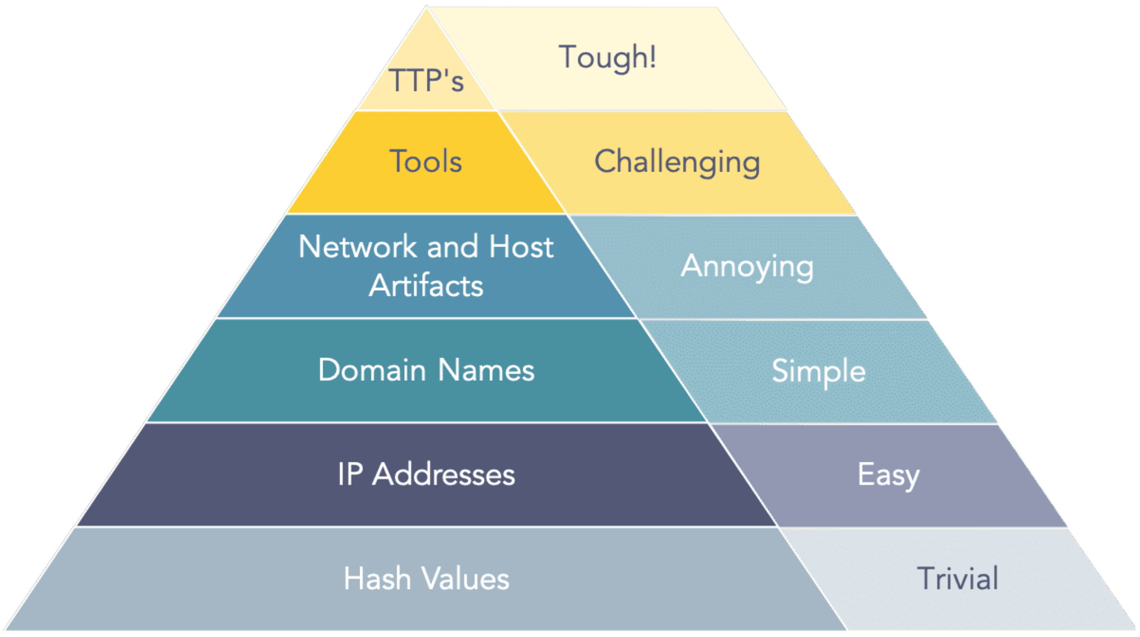


Figure 4. Pyramid of Pain created by David Bianco [25].

Research objective of the thesis is to find correlations between malware specimen at the code level, which represent the highest layer of Pyramid of Pain model. The code of various penetration tools can often be embedded in malware. Several programmers, and not necessarily only malware, have their habits, style, templates, and techniques that they learned during their careers. These features of programmers are very difficult to change, unlike, for example, the hash of the program or the IP address on which the malware communicates. For this reason, detection at the TTP's of the malware developer level is one of the most accurate technique for malware detection or categorization.

Malware or binary programs are not considered as one large chunk of bytes. On the contrary, malware is viewed as a commodity that can be effectively utilized for analytical purposes. A binary malware sample typically comprises hundreds or thousands of small logical particles. These particles, referred to as malware genes, encapsulate the compiler code, but primarily represent the code of the malware developer. Malware genes contain reused code, templates, techniques, tactics, and programming styles of the malware developer. This information stands at the top of the "Pyramid of Pain" 4, and it can effectively correlate, attribute, and analyze malware samples based on it.

3.1 Malware Analysis

Malware analysis is an area that examines samples of various file formats and evaluates whether it is a malicious or benign sample. In the case of malicious samples, the task of malware analysis is to detect "Indicators of the Compromise (IOC)", TTP and also to answer the questions mentioned in the previous section 3. At the highest level, malware analysis is divided into two methods [17]:

- **Static analysis** - is part of the analysis, where the code is analyzed without executing it. Static analysis is excellent for obtaining the overview of the analyzed sample. With this technique, an overview of the places where the encryption routine, network communication, or persistence are located can be obtained.
- **Dynamic analysis** - the code was executed directly in a "safe" environment, and the analyst observed how the sample behaved. In dynamic analysis, it is possible to monitor network communication, activity on the file system or at a lower level such as stack, memory or registers.

Each of the given methods can be divided into two submethods, which are basic and advanced [17] code. Basic static and dynamic analysis focus on sample expressions without analyzing the binary code. Among the outputs of the basic static analysis are the detection of imported functions from the Import Address Table (IAT), entry point (EP), hash values. Basic dynamic analysis is sometimes also called behavioral analysis [26]. The analysis is performed in an isolated environment, which is called a sandbox. The aim of this analysis is to monitor the behavioral properties of the sample, such as communication to the Internet, changing registers or creating new files. These basic analyzes provide important information for code analysis such as if sample is packed or not.

The second submethod of static and dynamic analysis is called ‘advanced’. ‘Advanced Static Analysis’, examines binary code statically without running the sample. In this thesis, the term ’Static Code Analysis’ is utilized, which is more descriptive than ’Advanced Static Analysis’, but denotes the same concept. In practice, this method is frequently employed for expedited comprehension of the program as a whole. Thanks to advanced static analysis, places where interesting occurrences such as encryption routines arise can be identified. Advanced dynamic analysis is often used by analysts when they unpack code. However, each of these two main methods has its strengths and weaknesses. Therefore, the ideal strategy for malware analysis is to combine these methods.

Basic Dynamic analysis can be done manually using various tools such as Sysinternals [4], Wireshark [5], Regshot [6], FakeNET [7]. The second option is to use a Sandbox solution such as Cuckoo [8], Cape [9] or Drakvuf [10]. Manual basic dynamic analysis is time-consuming and technically more demanding. In this research, a semi-automated solution is sought; therefore, manual dynamic analysis is not examined in more detail. Sandbox solutions are complex tools that can automatically analyze a sample and thus simplify the analyst’s work. However, malware sandboxes have several disadvantages from the point of view of this research.

- **Evasion Techniques** - There are several techniques that malware can use to determine whether a sample is running in a sandbox. An example can be an outdated API hooking technique ("NtLoadKeyEx") from the ntdll library. This evasion technique applies to Cape and Cuckoo sandboxes and causes the virtual environment to crash [27]. Cape uses a DLL called "capemon" for monitoring, which is injected into the process using APC injection and then hooks functions [28]. Malware can thus detect whether its process has been injected using APC and thus reveal the CAPE sandbox. There are many possible techniques for evading the sandbox - from simple sleep functions to targeted techniques per sandbox.
- **Hard to scale** - Analysis of a sample in a sandbox is very time and resource-intensive. One sample can be analyzed in several minutes/tens of minutes. This presents an issue for the objective, which aims to analyze malware in large volumes, in an on-premise environment, and semi-automated. To process at least one-hundredth of the malware published daily, thousands of samples need to be analyzed per day. Dynamic analysis is hard to scale to such an amount of samples each day.

[4]https://learn.microsoft.com/en-us/sysinternals/
[5]https://www.wireshark.org/
[6]https://github.com/Seabreg/Regshot
[7]https://github.com/mandiant/flare-fakenet-ng
[8]https://cuckoosandbox.org/
[9]https://capesandbox.com/
[10]https://github.com/CERT-Polska/drakvuf-sandbox

Basic static analysis is not sufficient in many cases. An example can be a technique where malware loads imports dynamically based on the hash value of library names and functions. Basic static analysis cannot detect this technique because it does not examine the code. In malware analysis, code analysis is often necessary to reveal and understand Tactics, Techniques and Procedures (TTP).

Static code analysis necessitates an understanding of reverse engineering. However, this thesis is not delve into the manual execution a static code analysis using tools such as IDA [11], Ghidra [12] or Binary ninja [13]. On the contrary, research objective is to make the job of the analyst more effective and to alert him to possible traps in the binary code. Examples of such alerts include identifying whether any analysts have previously examined a similar sample. Another aim is to classify the sample into a malware family cluster or potentially attribute malware to the threat actor. Static analysis has the advantage that it is fast. By statically analyzing and normalizing binary code, sophisticated analyses such as correlation and clustering can be conducted. The technique for normalizing statically analyzed binary code is referred to as malware genes (see Figure 5).

However, static analysis also has some disadvantages when analyzing malware. One of the most common problems for static analysis is packed and protected samples. These samples are unpacked during the program, which makes static analysis impossible. There are several statistics about percentage of packed malware samples. Some research says that 80% of malware samples are packed [29], another that it's 34,79% of all samples [30]. Because of such inconsistent results, a small experiment was conducted based on samples from MalwareBazaar [14], which were submitted between 25.-31. October 2023. This dataset contain 2023 samples, from which 60,11% represent PE files. A more detailed description of this experiment and its results is attached to the Appendix**??**.

Another problem for static analysis method are anti-analytical techniques such as anti-disassembling. These techniques attack the vulnerability of disassembly algorithms and the subsequent creation of a Control Flow Graph (CFG). Examples of this technique are "Opaque predicates", "Control Flow Flattering" or "Dead code". This thesis is focused on creating malware genes from static analysis. The Maltraits project, of which this thesis is a part, combines methods of static and dynamic analysis 3.3.

[11] https://hex-rays.com/ida-pro/
[12] https://ghidra-sre.org/
[13] https://binary.ninja/
[14] https://bazaar.abuse.ch/

3.2 Genetic Malware Analysis

Genetic malware analysis is a new term which was created by combining data analysis and malware genes. In this research malware genes are created by static analysis method. In general, it is also possible to have malware genes that are created using dynamic analysis. One of the option for such solution is utilize Dynamic Binary Instrumentation (DBI). DBI is software which is similar to debugger which can create malware genes during the runtime of the program [31]. However, this research focuses on malware genes that are created by static analysis. Genetic analysis works on outputs from static code analysis. It is important that if the static analysis fails, the genetic analysis of the malware will also fail by implication.

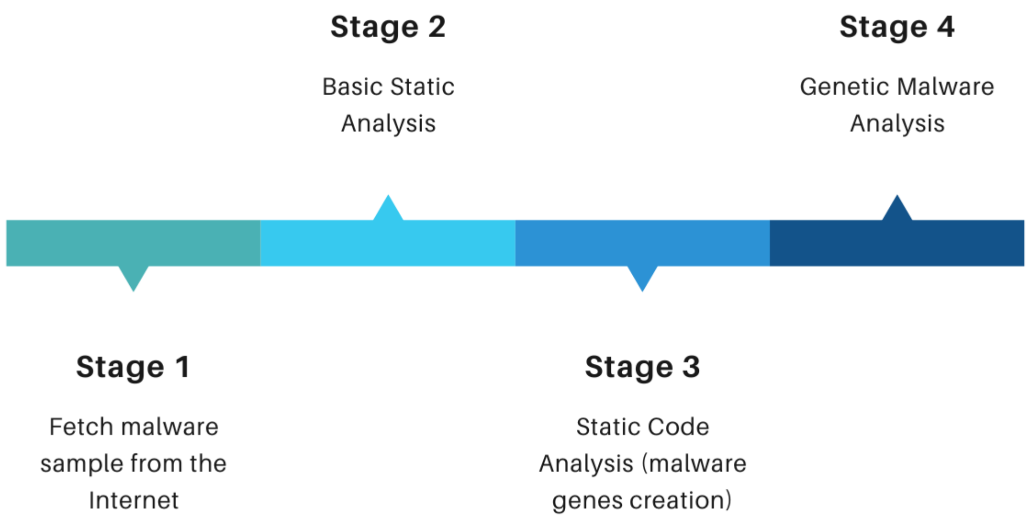

Stage 2
Basic Static Analysis
Stage 4
Genetic Malware Analysis
Stage 1
Fetch malware sample from the Internet
Stage 3
Static Code Analysis (malware genes creation)

Figure 5. Malware genes visualized by MalTraits.

Malware genes are disassembled parts of the code, which are enriched with advanced analytical metadata. Each sample has on average hundreds or thousands of such genes. A Canadian researcher with the nickname ‘c3rb3r3u5d3d53c’ came up with the idea of binary traits in a project called Binlex [32]. Malware genes are a continuation and improvement of this format, which is modified for use in Machine Learning algorithms. At the level of malware genes, advanced analyses can be performed, such as classification, clustering, and anomaly detection. This brings enormous added value to malware analysts, as samples can then be easily categorized and clustered based on parameters. Malware genes can be evaluated based on entropy, number of instructions, and size. A malware gene with two instructions certainly has no analytical value like, for example, another malware gene with twenty instructions. In the MalTraits project 3.3, three analytical instances are utilized, namely, (malware) sample, family, and threat actor. Each of these instances consists of a corpus of malware genes whose correlations can be graphically visualized as shown at 5.

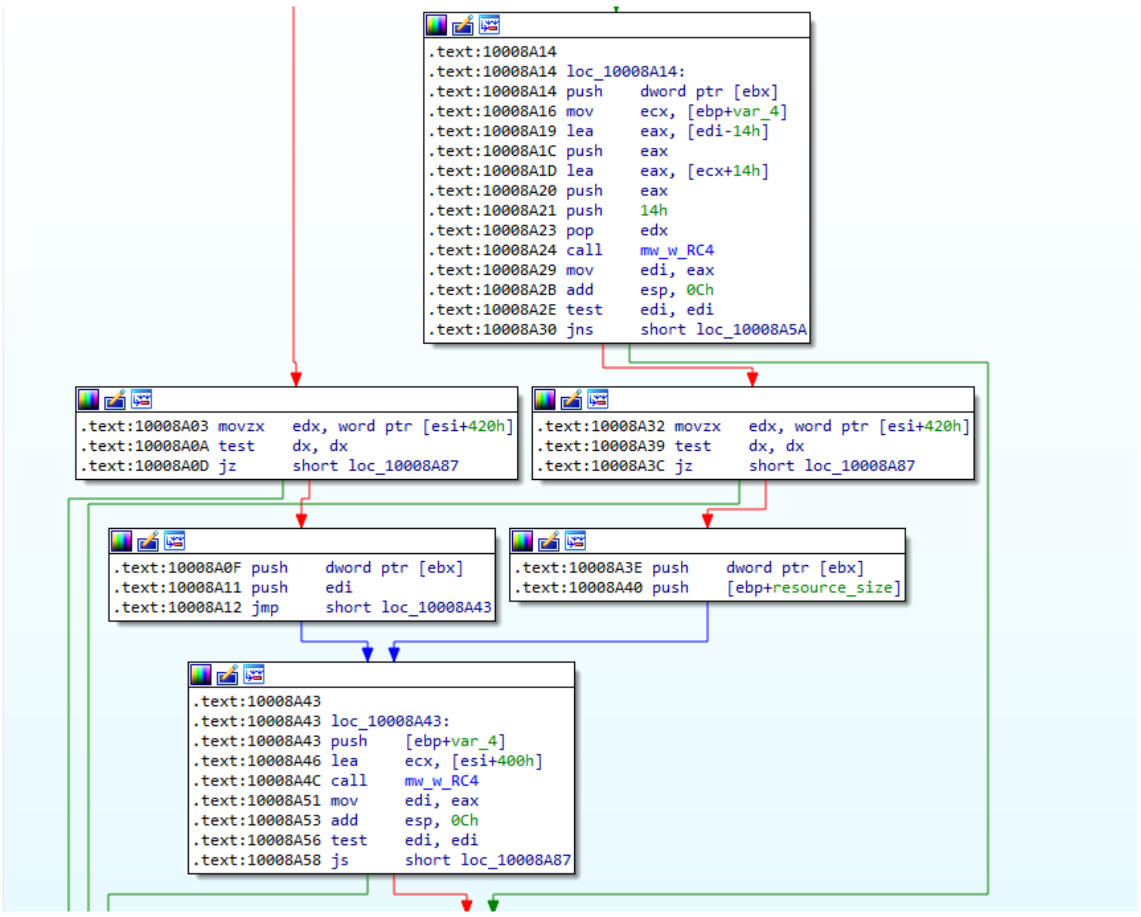
Correlations in malware analysis are very important, because they can reveal similarities between samples, malware families and threat actors. In the case of malware, it is common that the authors copy some part of the code from another program. This code may contain anti-analytic techniques such as obfuscation, anti-debugging, anti-virtual machine or other evasion techniques. If a given technique has already been seen in another malware from the dataset, the analyst can be automatically alerted instead of manual analysis. Use case for such alert can be example, where address 12345678 and the function "sub_87654321" contains a technique that has already been observed in the past. This is one of the possible examples of using correlation to make manual code analysis more efficient. The result of the correlation can be an automated list of the sample techniques that have already been seen in other samples.

However, with the help of malware genes, this can also be achieved. Certain parts of the code may be identified as suspicious and not found in other samples. On the image 5 you can see the correlations between the five malware samples. The analyzed sample is in the middle of the graph, which is represented by only one (dark blue) node. From the graph, which is created from the correlation of the malware genes of the samples, it is evident that the pale blue sample (at the bottom of the graph) has many genes in common with the analyzed sample. Malware gene correlation is powerful technique for determining sample similarity.

```
{
   "id" : 43,
   "architecture" : "x86",
   "average_instructions_per_block" : 7,
   "block" : 1,
   "bytes" : "00 00 04 17 6f 45 00 00 0a 03 7b e6",
   "bytes_entropy" : 2.9182956218719482,
   "bytes_sha256" :
      "525ed8951a4726a0f6d40c230db6f441233cc0370f0d8dd6b07356fa793df86a",
   "corpus" : "default",
   "cyclomatic_complexity" : 3,
   "edges" : 2,
   "instructions" : 7,
   "invalid_instructions" : 0,
   "offset" : 1337,
   "size" : 12,
   "trait" : "00 00 04 17 6f 45 00 00 0a 03 7b e6",
   "trait_entropy" : 2.9182956218719482,
   "trait_sha256" :
      "525ed8951a4726a0f6d40c230db6f441233cc0370f0d8dd6b07356fa793df86a",
   "type" : "block"
}
```

Procedure 3.1. Example of malware gene

Malware genes are thus the basic atom of the MalTraits project. In genetic malware analysis, two basic types of malware genes are distinguished: functions or blocks. A block is a sequence of bytes up to the nearest conditional or unconditional jump 6. Malware genes of type ‘function’ contain several malware genes of type ‘block’. Metadata of malware genes is very important for machine learning, which helps to add value to them. In practice, a malware gene that contains, for example, ten instructions has a greater value than a malware gene that contains only two instructions.



```
.text:10008A14
.text:10008A14 loc_10008A14:
.text:10008A14 push    dword ptr [ebx]
.text:10008A16 mov     ecx, [ebp+var_4]
.text:10008A19 lea     eax, [edi-14h]
.text:10008A1C push    eax
.text:10008A1D lea     eax, [ecx+14h]
.text:10008A20 push    eax
.text:10008A21 push    14h
.text:10008A23 pop     edx
.text:10008A24 call    mw_w_RC4
.text:10008A29 mov     edi, eax
.text:10008A2B add     esp, 0Ch
.text:10008A2E test    edi, edi
.text:10008A30 jns     short loc_10008A5A
```

```
.text:10008A03 movzx   edx, word ptr [esi+420h]
.text:10008A0A test    dx, dx
.text:10008A0D jz      short loc_10008A87
```

```
.text:10008A32 movzx   edx, word ptr [esi+420h]
.text:10008A39 test    dx, dx
.text:10008A3C jz      short loc_10008A87
```

```
.text:10008A0F push    dword ptr [ebx]
.text:10008A11 push    edi
.text:10008A12 jmp     short loc_10008A43
```

```
.text:10008A3E push    dword ptr [ebx]
.text:10008A40 push    [ebp+resource_size]
```

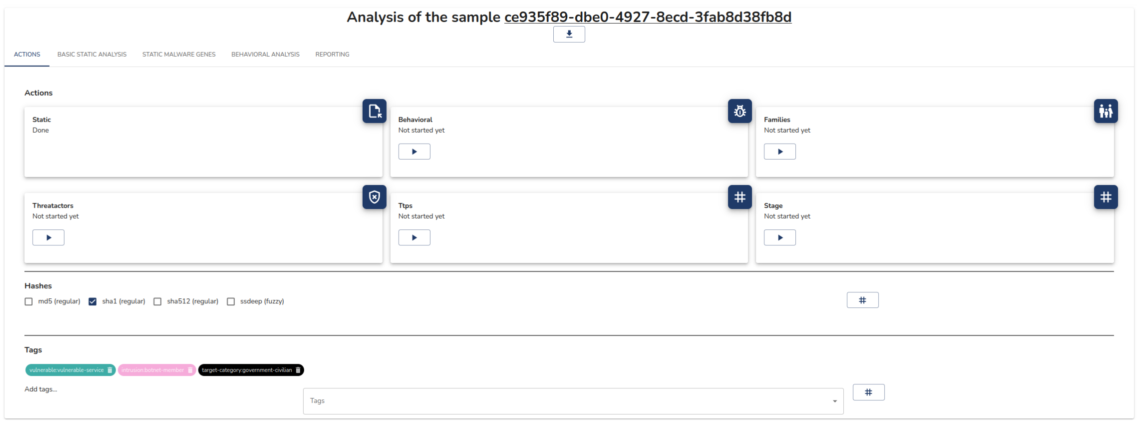
```
.text:10008A43
.text:10008A43 loc_10008A43:
.text:10008A43 push    [ebp+var_4]
.text:10008A46 lea     ecx, [esi+400h]
.text:10008A4C call    mw_w_RC4
.text:10008A51 mov     edi, eax
.text:10008A53 add     esp, 0Ch
.text:10008A56 test    edi, edi
.text:10008A58 js      short loc_10008A87
```

Figure 6. Type “block” Malware genes represented in IDA as basic blocks.

3.3 MalTraits

The MalTraits project was created as product of the author's previous bachelor thesis [3]. The aim of this platform is to connect research questions in the field of malware analysis with the needs of practice. The goal of this work is to do applied research and apply the acquired knowledge to the platform in the form of code. The goal of this research is to support organizations for proactive collection and analysis of malware samples. This industry was named "Malware Data Analysis." However, since there are not many tools in practice that would enable this approach in an offline environment, the decision was made to create MalTraits. The objective of MalTraits is to be a full-fledged platform for "Malware Data Analysis" in an on-premise environment.

The scope of the MalTraits project is much wider than the scope of this research 1.2. The ultimate goal of this platform is to create a platform for advanced malware analysis in an on-premise environment. This platform supports the creation of an on-premise malware repository and the modification of the binary program to malware genes. As described in the Limitation 1.5 section, static analysis is not effective when the sample is packed. For this reason, dynamic analysis was employed in the MalTraits research project. The objective of this method is to unpack the sample and return it for static analysis, where it can be analyzed using static and later genetic analysis.


Analysis of the sample ce935f89-dbe0-4927-8ecd-3fab8d38fb8d

Static
Done

Behavioral
Not started yet

Families
Not started yet

Threatactors
Not started yet

Ttps
Not started yet

Stage
Not started yet

Hashes
☐ md5 (regular) [✓ sha1 (regular)] ☐ sha512 (regular) ☐ ssdeep (fuzzy)

Tags
vulnerable:vulnerable-service
intrusion:botnet-member
target-category:government-civilian

Figure 7. Malware analysis page of MalTraits platform.

It is important to note that this thesis nor project MalTraits are not trying to create an Antivirus solution that will tell the analyst whether the sample is malicious or benign. This research is created for malware analysts and malware researchers, who decide for themselves whether the sample is malicious or not. Main objective of the thesis is to help simplify their work by research method called 'Malware Data Analysis'.

3.3.1 Platform Architecture

Malware analysis is a complex process that consists of several parts as described in section 3.1. This research connects malware analysis and data science, or increases the complexity of the solution architecture itself. The architecture of the MalTraits research platform consists of several main components.

- Sample Preprocessing - C/C++
- Backend - Python Django, Celery workers [15], Channels websockets [16]
- Frontend - React, Material UI [17]
- Databases - PostgreSQL, MinIO, MongoDB, Redis
- Machine Learning - Python scikit-learn [18]

The sample preprocessing code is a forked code of the open-source tool Binlex, which is written in the C/C++ language [32]. The purpose of this code is to disassemble the code and enrich malware genes with important metadata. The C language is very well suited for this, since disassembly requires work. In addition, a complicated operation such as disassembly requires an effective language such as C/C++. The Binlex program communicates with the Backend using Python bindings and the 'pybind11' library.

An important component of the entire research is the Backend written in the Django Python framework. Since automated analysis is a difficult process that takes several seconds to minutes, Celery workers are added to the Backend. Workers together with the Redis message broker enable the execution of complex processes distributed on several server nodes in the future. Instead of waiting for HTTP responses, workers work in the background and communicate with Frotend using web sockets. This research can effectively and distributedly process malware using tasks, queues and workers.

Frotend for this research is significant in terms of data representation and results. Since the aim of the research is to process a large amount of malware, one of the critical areas of the research are databases. Several types of databases are used in the research. The PostgreSQL relational database is utilized to store relational data such as malware genes, samples, and malware families. Each of the mentioned classes has some relationship with another class. An example is a malware family that contains one or more malware samples. However, the malware platform also needs to store binary programs as such, should they be

[15]https://docs.celeryq.dev
[16]https://channels.readthedocs.io
[17]https://mui.com/
[18]https://scikit-learn.org/stable/

needed in the future. Therefore, the MinIO database, which works with buckets, is used to store malware samples. Redis in this architecture is used as a message broker that enables communication between multiple workers and other application components [33]. Finally, the NoSQL database MongoDB is added to the architecture. In research, this database is intended for data related to machine learning.

In this research, the Python library Scikit-learn was used for problems related to machine learning. This library has available a large number of algorithms from the category of supervised or unsupervised machine learning [34]. The technological stack of this research is relatively demanding, regardless of the fact that the main technical requirement is experience in malware analysis. From this point of view, this research does not go into depth in several technologies, but only uses their basic parts.

3.4 K-Nearest Neighbors

One of the first supervised machine learning methods used in this research is called K-Nearest Neighbors (KNN). This method is useful when there is malware available that has a label, that is, it is known which malware family it belongs to. The objective of the KNN algorithm is to classify malware that does not contain a label. This classification is based on the calculation of the distance of the sample from the elements in the data model. This method is relatively simple and consists of only a few steps. The main step is the creation of a data model, which can represent, for example, an array of vectors. When a malware sample is received without a label, it is detected using the distance algorithm. A frequently used algorithm for determining the distance in KNN is actually the Euclidian distance algorithm [35].

$$||\mathbf{p} - \mathbf{q}||_2 = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3.1}$$

The Euclidean distance equation in equation 3.1 calculates the distance between two points represented by vectors p and q in n-dimensional space. In this case variable p represent labeled malware, from the model and another variable q represent unlabeled malware. Euclidian algorithm measures the length of the shortest path between analyzed malware samples and samples from the model. There are also other methods for obtaining the distance between vectors, such as the Manhattan or Minkowski distance [36, 37]. The algorithm for calculating the distance between vectors is very important for the results of the KNN algorithm itself [35].

The last step in the KNN algorithm is to determine the k coefficient itself. This coefficient determines how many nearest ‘neighbors’ to the analyzed sample are selected. Finally, the malware family that has the largest representation among the selected ‘nearest neighbors’ is selected. M. Rao et al. [38] used the KNN algorithm for malware classification and achieved an accuracy of 98.02%. The accuracy of KNN in the given research was better than that of the Random Forest or Decision Tree algorithm.

Among the main benefits of KNN is the simplicity of the algorithm combined with generally good classification results. The biggest disadvantages and concerns of the KNN algorithm in this research lie mainly in two points:

- High dimensionality
- Memory demanding

Malware sample represented as a sequence of malware genes may be described as multidimensional array. In high-dimensional spaces there is often problem that concept of proximity becomes less meaningful. Result of such phenomenon is that distances tend to be more uniform [39].

3.5 K-means

Malware samples often do not have a label that represents the name of the malware family to which it is assigned. In addition, malware families are constantly growing and creating new ones. The clustering of malware families and threat actors is significant from several points of view, such as:

- Discovery of new malware families. This area is interesting from a research point of view, being the "first" researcher who labeled a new malware family. However, it also has a practical dimension in revealing TTP’s new family.
- Code correlations between threat actors at the TTP level in binary code.
- Finding anomalous malware samples that contain, for example, exploits for new vulnerabilities. These vulnerabilities are often called zero-day vulnerabilities.

Supervised Machine Learning algorithms classify malware based on pre-existing data. When dealing with a new malware family, supervised machine learning struggles to detect its emergence. One potential approach involves establishing a static threshold for minimal distances to neighboring instances. Nonetheless, this method proves inefficient and prone to inaccuracies.

An effective solution for responding to newly emerging malware families is the use of unsupervised machine learning algorithms. One of the methods of unsupervised ML are clustering algorithms such as k-means. Clustering algorithms can create clusters of similar samples and also point to samples that do not contain similar malware. The K-means algorithm is one of the simplest clustering algorithms [40]. K-means algorithm can create the mentioned clusters and thus detect newly emerging malware families. The K-means algorithm utilizes centroids as pivotal points for clustering. Nevertheless, the selection of centroids represents a critical step in the algorithm, often susceptible to significant inaccuracies [41].

Similar to KNN, k-means employs the Euclidean algorithm to compute distances between centroids. However, the Euclidean algorithm can suffer from inflation in high-dimensional spaces, a phenomenon known as the "curse of dimensionality" [42]. One solution to mitigate the effects of high-dimensional space is Principal Component Analysis (PCA). The objective of the PCA algorithm is to condense the maximum amount of information into a reduced number of dimensions, typically two columns [43]. Employing PCA makes it feasible to enhance algorithms that are reliant on the Euclidean distance algorithm.

4. Related work

Malware analysis is a very broad area for research, because it can include many topics such as unpackers, decryptors, custom file formats, binary instrumentation, correlation of samples or involvement of machine learning. On the other hand, static analysis of malware genes is a new area that is not yet very well researched. In the study of related research, two areas closely connected with the topic of the thesis and the MalTraits research project were defined.

4.1 Review Protocol

The strategy for the literature review was the Systematic Literature Review (SLR) method, which consists of several steps. At the beginning of the research of related works, scientific databases were specified in which subsequent searches for related research were conducted. These libraries were selected based on the general qualities of the given databases and at the same time the possibility of access to them. The scientific databases that were selected for this thesis are IEEExplore [1], SpringerLink [2], ACM Digital library [3].

Since malware analysis is too broad a topic, the analysis of related works was defined in three parts, which were examined separately.

- Static malware analysis techniques
- Machine learning in malware analysis
- Attribution of malware to APT groups

Inclusion criteria for searching resources were based on key words that were defined and simple queries created. Another criterion was the year of viewing the work, where it was required that the work be newer than 2016. Due to time boundaries, the criteria were defined that the research from which the current state of the art will be analyzed should already have some citations or evaluation. The language criterion was that the work must be in English. The scope of work was included in the advanced search, focusing only on PE files. This criterion excluded a lot of research that was done for Android or mobile malware samples, which are different. For searching in scientific libraries, advanced search with the query 4.1 was used.

[1]https://ieeexplore.ieee.org
[2]https://link.springer.com
[3]https://dl.acm.org

```
("All Metadata": static  malware analysis )
NOT ("All Metadata":mobile)
NOT ("All Metadata":Android)
OR ("All Metadata":malware genes)
AND ("All Metadata":machine learning )
```

Procedure 4.1. Review protocol queries

4.2 Static Malware Analysis Techniques

Malware genes are not the only static method that can be used for the classification and correlation of malware. Well-known static methods are based on data from Control Flow Graph (CFG) [44], Function Call Graph (FCG) [45, 12]. Other static techniques include visualization of a binary [46, 47, 48], n-gram analysis of a binary program [49], machine learning based on imports [50], and also various types of hashes, from classic fuzzy hashes [51] to Local Sensitive Hashing (LSH) [52]. Each of these methods has its strengths and weaknesses. The problem with classic hashing for malware detection was previously mentioned in Section 3. Although common hashing is simple, it proves to be very ineffective in practice.

4.3 Machine Learning in Malware Analysis

Most of the research work on the involvement of machine learning in malware analysis was done based on data from dynamic analysis. I. Muhammad et al. [53] used machine learning in their research on data from static and dynamic analysis. Interestingly, the static analysis had a better accuracy of 99.36 %, while the dynamic analysis had only 94.64 %. D. Nikolopoulos and I. Polenakis [54] proposed in their research the interception of system calls using dynamic analysis. They called this technique System-call Dependency Graphs, from which they then transformed the data into a Gr-Graph structure.

Dynamic analysis was also used by J. Vrancken [55] when they captured the program’s API calls and its arguments to detect malware capabilities. The sequence of API calls together with the values of the arguments that are sent, is a very valuable source of information for the analyst and also for a potential data model. This information is difficult to obtain from static analysis because it can often be prevented by obfuscation of the program, such as API hashing [56].

4.4 Attribution of Malware to APT Groups

The research topic of C. Boot [57] proposed the attribution of the malware to the threat actor. The author of this research gets data from dynamic malware analysis, more concretely sandbox software. This research analyzes in great detail the possible approaches and uses of machine learning in malware analysis, focusing primarily on the comparison of deep neural networks and random forest classifiers. The objective of the research is to find out to what extent it is possible to attribute a sample to Advanced Persistence Threats (APT) groups using a machine learning model. The accuracy of the results depended, among other things, on the sandbox solution itself. There are slight differences between several sandbox solutions. In general, deep neural networks emerged in this research as a slightly more suitable algorithm for clustering and attributing samples to threat actors.

One of the essential benefits of malware gene format is the search for "code reusing" in malware programs. Research of Upchurch et al. [52] was focused on problem of code reusing in malware. They tried to find code reuse in malware using the Locality Sensitive Hashing (LSH) scheme. The advantage of this approach is the relatively small complexity of the solution. On the other hand, there is a possible problem if the LSH scheme is combined with n-gram code blocks. In such a case, there is a large overhead, and a large volume of blocks is needlessly hashed. In this case, hashes at the level of malware genes are significantly more effective.

4.5 Primary Studies

After a broader analysis of the current state of the art in the field of static analysis techniques and the use of machine learning in malware analysis, the primary sources of this thesis were specified in the following section. The factors on the basis of which these works were selected included the topic of the research, the potential benefit for the thesis, and the quality of the research.

Detecting Code Reuse in Malware via Decompilation and Machine Learning

O. Mirzaei et al. [58] created unique research, where they linked dynamic analysis with static. The motivation for this solution was the fact that both methods have their strengths and weaknesses. The applied methods in their research consisted in dynamic executing of the malware, and when the malware reached a "critical" location, a memory snapshot was taken. The code from memory was subsequently unmapped and decompiled in the open-source tool Ghidra [4]. In the decompiled pseudocode, the authors also used static methods such as n-grams and LSH to search for functions and finally vectorized the pseudocode using the Clang AST [5]. This vectorized output sends as input to an unsupervised machine learning algorithm. This research used a combination of dynamic and static analysis, which made it qualitatively different from the others. Disadvantages of this research are the enormous demand for resources required for this kind of sample analysis. Applying this procedure to data malware analysis would be very expensive in terms of resources. On the other hand, by using a combination of analytical methods, it can even analyze obfuscated samples.

Semi-Supervised Malware Clustering Based on the Weight of Bytecode and API

Y. Fang et al. [59] created research which combined dynamic and static analysis. In research, they come up with a solution for measuring the similarity of malware samples and the use of clustering with the help of semi-supervised machine learning. Precisely in the technique of clustering and involving machine learning, this research is very similar to this thesis topic. The combination of supervised and unsupervised machine learning is essential for recognizing even previously unknown malware or malware families. The algorithm used for machine learning is called Density-Based Spatial Clustering of Applications with Noise (DBSCAN). Accuracy achieved during research testing reached 98.7 %.

Exploring Function Call Graph Vectorization and File Statistical Features in Malicious PE File Classification

Y. Zhang et al. [45] divided static code analysis into two categories, which are graph-based and non-graph features. This research focused only on PE files. The primary technique used by the authors was the Function Call Graph (FCG), which they then vectorized (FCGV). They enriched this vectorized graph with non-graph statics. These features include cross-references in the disassembled PE file to Windows artifacts, such as the Registry. The authors used n-gram and LSH techniques for static analysis. They vectorized the vector composed of six distributed data sources and created a model using the Random Forest machine learning algorithm. The results of this research reached an accuracy level of 99.57 % on the Microsoft dataset (Kaggle BIG 2015).

A Gene-Inspired Malware Detection Approach

Y. Chen et al. [60] created gene-inspired malware detection research. This research agrees with belief that the often popular n-gram analysis for static analysis is fragmented and has a large overhead. In their solution, the authors use the IDA Pro disassembler by which they

[4]https://ghidra-sre.org
[5]https://clang.llvm.org/docs/IntroductionToTheClangAST.html

create malware genes. The authors use the Smith-Waterman algorithm to determine the difference between samples with "n" number of genes. These samples are clustered using the Neighbor-Joining (NJ) algorithm. During testing, this technique achieved an accuracy of 96.14 %. The advantage of the malware genes is that it generates much less data than n-grams and is therefore more easily scalable for malware data analysis use.

MGeT: Malware Gene-Based Malware Dynamic Analyses

J. Ding et al. [13] researched the field of dynamic malware genes, and their research is called "MGeT". This research described very precisely what malware genes are, why they are termed as such, and their connection with genes from biology. It has many similarities to this research topic, although the authors based the research on dynamic analysis. The authors represent the sample as a sequence of behavioral traits that they try to correlate with the behavioral set. The accuracy of the research was compared with three other methods - Hidden Markov model (HMM), n-gram algorithm, and Dynamic Time Wrap (DTW-SVM). The MGeT method of the authors was significantly better in the evaluation of accuracy, which was at the level of 90

End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware

Rosenberg et al. [61] focused their research on the attribution of APT malware samples. Deep neural networks (DNN) were used as classifiers for the attribution and clustering of threat actors into individual groups. As input data for machine learning, they used dynamic analysis in the form of a raw report from the Sandbox tool Cuckoo [6]. This research showed an accuracy of more than 98.6% in sample attribution. These results were tested on a dataset of only a thousand samples, which may be questionable what the results would be in a large dataset consisting of benign programs, common malware, and APT malware. The second problem of this research may be that the samples came primarily from Russian and Chinese APT groups. In another research [62], the same authors with the same method focused on the identification of new malware families. In this research, they achieved a result at a level of 97.7 % accuracy.

4.6 Assessment of Study Quality

The static analysis of malware genes is a topic that has not yet been extensively researched, and so far, based on available knowledge, only a few studies have been found on the given topic. On the other hand, several high-quality studies were found that were based either on another static method or on data from dynamic analysis. This research brought the field

[6]https://cuckoosandbox.org

closer to many problems that researchers aim to avoid. Examples of such problems include an uneven dataset during sample attribution or excessive reliance on a single analytical method in malware analysis. With the help of a literature review, the connection between malware analysis and machine learning's significant advantages was confirmed. However, to conduct machine learning, high-quality and up-to-date data are required. Overall, through the analysis of the current state of the art, a better overview of the algorithms used for machine learning was obtained. The result of this is the specification of machine learning algorithms for the research based on the results of previous studies.

4.7 Data Extraction

During the analysis of primary sources, several pieces of information were found that can assist in analyzing the research questions. Data vectorization is the most frequently applied method of editing input data for machine learning in selected primary sources. The Graph Edit Distance (GED) algorithm was often used to determine the difference between samples. Most of the research from primary sources used supervised machine learning. However, the research from [58], which applied a combination of supervised and unsupervised machine learning, was the most interesting for us. As a result of this combination, the machine learning algorithm should be able to recognize even previously unknown malware. The concrete machine learning algorithm is highly dependent on the input data. This input data is in the unique format of malware genes, making it difficult to find an answer from primary sources as to which machine learning algorithm is right for the data.

Another group of primary studies consists of the detection of APT threat actors and undetected malware [61]. This research demonstrated that samples from a given APT group can be attributed with great accuracy. However, leveraging malware genomes and machine learning, advancements in this research topic are sought to discern the common tactics, techniques, and procedures (TTPs) of APT groups at the binary code level. To the best of author knowledge, this area has not been explored yet, and it is believed that insights can be gleaned with the assistance of malware genes.

4.8 Data Synthesis

During the literature review, several high-quality studies on the topic of machine learning in malware analysis were found. The result of query in 4.1 was 361 research articles, from which the primary studies for the thesis are selected in Section 4.5. 298 research articles were excluded from the SLR as they did not align with the focus area of this research.

Examples included areas beyond the scope of this study, such as Linux binary programs, excessive emphasis on dynamic analysis, or binary programs in IL languages. 54 research articles were analyzed, of which 6 were selected as primary sources for this research. The research topic of the analyzed literature is the main criterion for the selection of the primary source. This topic has to be related to this thesis questions. Research topic of the most thesis is machine learning based on data from static or dynamic analysis.However, most of the research did not apply to an environment where the analysis of thousands of samples per day could be conducted semi-automatically. This poses a problem because when investigating the automatic correlation and detection of samples at the level of a machine learning model, it is necessary to constantly enrich the data model. However, several kinds of research, despite a high percentage of accuracy, are not applicable in practice due to high demands on resources.

4.9 Review Report

In this work, the methodology called "Systematic review" was followed. The main criterion for the literature review was the identification of research related to the research questions set. In the initial analysis of sources, a wider range of sources was selected, from which primary sources were subsequently chosen. Primary sources were selected on the basis of the similarity of the researched topic, those that dealed with similar research questions and at the same time met the qualitative standards set by us.

4.10 Challenges in Malware Analysis

There are several challenges in malware analysis that are not solved by current solutions. These problems were divided into several points. These challenges are addressed in the present work with the help of research questions defined in Section 1.3.

Lack of work with malware samples in the field of data analysis

Apart from antivirus companies, only a few security teams work in data analysis with malware samples. Data malware analysis includes operations such as sample collection, normalization and data analysis. One of the main reasons why security teams do not work with malware in the field of data analysis is the lack of methods to normalize malware samples. So far, one of the most frequently studied n-gram methods 4.2 has a large overhead and in practice often turns out to be inefficient. With malware genes, the aim is to address this issue and normalize malware at the code level, enriching it with a wealth of metadata and information that can be effectively utilized during analysis. In the first research question 1.3, the focus lies on identifying the ideal inputs from malware genes for machine learning. Subsequently, efforts are directed towards identifying enhancements to malware genes that would bolster machine learning clustering, correlation, and attribution of malware.

The use of machine learning in malware analysis

In recent years, several researchers have tried to use machine learning in the field of malware analysis. However, many of these solutions were not put into practice primarily due to high resource costs for analysis [58]. Thesis solution is based on malware genes that can analyze correlations in the malware gene dataset within seconds. Challenges in the current MalTraits solution and also in the field of genetic malware analysis is the effective use of machine learning. It is not defined what the input from malware genes should be for machine learning, how to calculate “distances” between malware genes, and also how to cluster samples. Finally, none of the analyzes of the use of machine learning in malware analysis solved the problem of sharing these data models. From security principles, it is known that sharing information with selected parties is critically important. That is why in the second and third research questions 1.3, these gaps of the current state of the art are analyzed.

Lack of information on the similarity of APT groups at the TTP and code level

From the "Pyramid of Pain" model 4, it is understood that the attacker’s TTP is the most valuable artifact for the cyber defense team. Thanks to this, better attribution of samples and attacks can be achieved, along with enhanced recognition of attackers. Insights into where attackers obtained the code, whether open-source code was used, whether traces were left during code download, and the tools or templates utilized, can all be gleaned from code analysis. Rosenberg et al. [61] adeptly attributed samples to APT groups in their research. However, this problem is approached differently here. The objective is to identify common malware genes and perform attribution at the code level, with all the aforementioned information being derived from code analysis in the form of malware genes. This technique aids in improved recognition of APT groups as threat actors. The fourth research question 1.3 addresses this issue.

5. Contribution

This research falls under the category of applied research, with one of its main objectives being the development of software aimed at enhancing the state-of-the-art in Malware Analysis. However, rather than developing an entirely new software, this research opted to enhance the MalTraits project further, acknowledging the complexity of the task. This platform aims to conduct proactive analysis of large malware volumes within an on-premise environment. Section 3.3 provides a detailed description of this platform. This research's primary contribution lies in utilizing machine learning instead of the comparative scale method. Given the complexity of the research, its contributions can be summarized in several key points:

1. **Data Model and Architecture Design** - one of the main results of the research is the description and data model engineering of malware genes. This type of malware genes has not been studied by any of the researchers so far, so it represents a crucial part of the research.
2. **Software Design & Programming** - during this research, the platform's experiments, algorithms, and analytical environment needed to be applied. Several screens were designed in wireframes from September 2023 to April 2024, and 174 Github commits were contributed. Together, thousands of lines of code in Python, React, and C++ were contributed to the research. The code for this research, as well as the entire MalTraits project, is currently accessible only through a link [63]. Following the conclusion of the research, there are plans to publicly release the project.
3. **Experimentation** - proved to be the most time-consuming aspect, involving extensive work with large datasets, editing, and testing various configurations and parameters. The most challenging aspect of experimentation lies in interpreting the results, understanding why an experiment succeeded or failed, and addressing instances of low accuracy.
4. **Reverse Enigeering & Malware Analysis** - an in-depth analysis was conducted on four malware samples, specifically from the QakBot and Emotet families, as a qualitative research component. Additionally, a program for extracting malware configurations was developed as a part of this study.

All four described areas were iterated linearly in consecutive cycles. Initially, a simple representation of the extracted features of malware genes described in the section 5.1 was prototyped. This representation was implemented as generic code that analysts can run

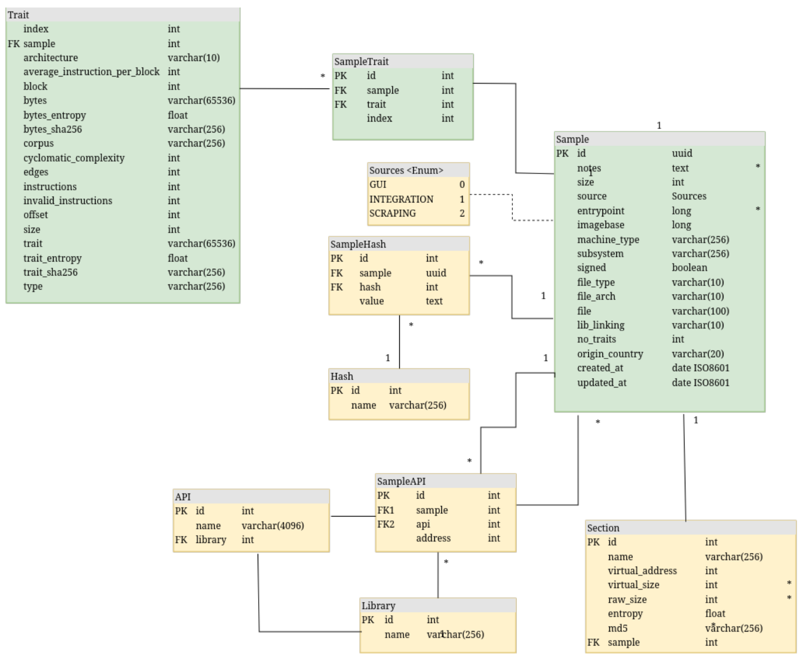


Figure 8. Database logical model of the Sample and the Malware gene.

and use in the application environment of a web browser. Subsequently, experiments and their evaluation were carried out on this designed and implemented architecture. In this document, the experiments are described in the Validation of the part in section 6. From this perspective, the Validation section stands out as a significant contribution of this research. It offers valuable insights into the application of machine learning in analyzing malware genes.

In certain instances, when deeper comprehension of the experimental results was required, malware samples underwent analysis via reverse engineering. After each iteration, the architecture was modified and improved. Main improvements define optimizations described in the section 5.1.3 or a new dataset representation described in the section 5.2.

5.1 Feature Engineering

Feature Engineering is critical in Genetic Analysis as it facilitates the transition from large-scale malware gene comparison to machine learning. In Figure 8, part of the logical database design illustrates the many-to-many relationship between the Trait (the term for the Malware gene) and the Sample of the malware. However, large-scale comparison poses challenges in terms of performance and fails to consider the significance of malware genes. This research does not aim to enhance the performance of large-scale malware gene comparison methods. Instead, this and subsequent sections outline a potential application and representation of machine learning in Genetic Malware Analysis.

The primary objective of this research is to define and extract inputs for machine learning, known as features. This process, termed feature engineering, forms a crucial aspect of the study [64]. In this research, features are represented as unique malware genes in the Genom Database. This database of unique genomes is shown in the Figure 8 under the table name 'Trait'. In this research, feature selection involves representing each unique malware gene in the 'Trait' table as an individual feature. This approach offers a highly accurate sample description through features. However, its potential drawback lies in the high dimensionality. A schematic representation of this method is depicted in matrix 5.1.

$$\begin{array}{cccc} \text{Malware Gene 1} & \text{Malware Gene 2} & \dots & \text{Malware Gene N} \\ \hline value_{1,1} & value_{1,2} & \dots & value_{1,N} \\ value_{2,1} & value_{2,2} & \dots & value_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ value_{M,1} & value_{M,2} & \dots & value_{M,N} \end{array} \tag{5.1}$$

When the MalTraits database contains thousands of samples, a single sample may possess a vector comprising millions of distinct malware genes. While solving the issue of storing and processing such a data model may not be necessary for research purposes, it presents a practical challenge. In practice, recalculating the machine learning model after each update becomes demanding. Consequently, this research addresses this challenge through two distinct approaches:

- Continuous storage of the data model in the database.
- Select only representative samples for the model.

A representative model can be made up of samples from databases such as Malpedia. The Malpedia dataset contains samples belonging to more than 2000 malware families and is very well maintained. Additionally, MalTraits analysts have the capability to supplement the core data model with samples they deem belong to specific families. As a result, the core data model comprises solely representative samples, thereby reducing the data requirements and algorithmic complexity.

This research categorizes two primary types of feature extraction of malware genes. The initial category of features is denoted by a binary style, commonly referred to as categorical data representation in the literature [65]. This method is detailed in Section 5.1.1. Conversely, the alternate approach assigns a feature value determined by the size of the malware gene. Consequently, a feature can assume values within the range $[0, \infty)$, as explained in Section 5.1.2.

5.1.1 Categorical Data Representation

The original idea of this research is the use of a categorical representation of the selected features. Machine learning algorithms tend to work on numeric values instead of strings. For this reason, Categorical Attributes is a method that statically assigns numeric values [65] to objects. In practice, these objects can be colors or animals. In this research, two categories were created:

- 0 - Malware gene $value_{s_1,i}$ from the genome database is not found in the sample
- 1 - Malware gene $value_{s_1,i}$ from the genome database is found in the sample

The algorithm for generating vector features of malware genes within a given sample iterates cyclically through the Genome Database, depicted as the initial gene chain in Figure 9. Subsequently, it conducts lookups to determine whether the sample contains any malware genes from the genome database, disregarding their position within the binary code. Upon finding a match between a gene from the genome database and a malware gene in the sample, the resulting vector stores the value 1; otherwise, it stores 0. Notably, the size of the resulting vector matches the number of unique malware genes in the Genome Database. The algorithm implementation is depicted in Procedure 5.1, utilizing the Numpy library [1].

```
def vectorize_genes(self, sample: Sample):
    sample_traits =
        np.array(sample.trait.values_list("trait_sha256",
        flat=True))
    return np.isin(self.traits_experiment_all,
        sample_traits).astype(int)
```

Procedure 5.1. Function which vectorize malware genes

The choice of machine learning method or distance algorithm heavily depends on the data representation utilized. While algorithms like Cosine similarity or Jaccard index are commonly employed for categorical data representation [66, 67], the Euclidean distance algorithm may not be the most suitable for multi-dimensional datasets containing categorical data. Thus, it is crucial to comprehend both the data and the algorithms being employed thoroughly. This study evaluates and experiments with various methods and data representations.

[1]https://numpy.org/doc/

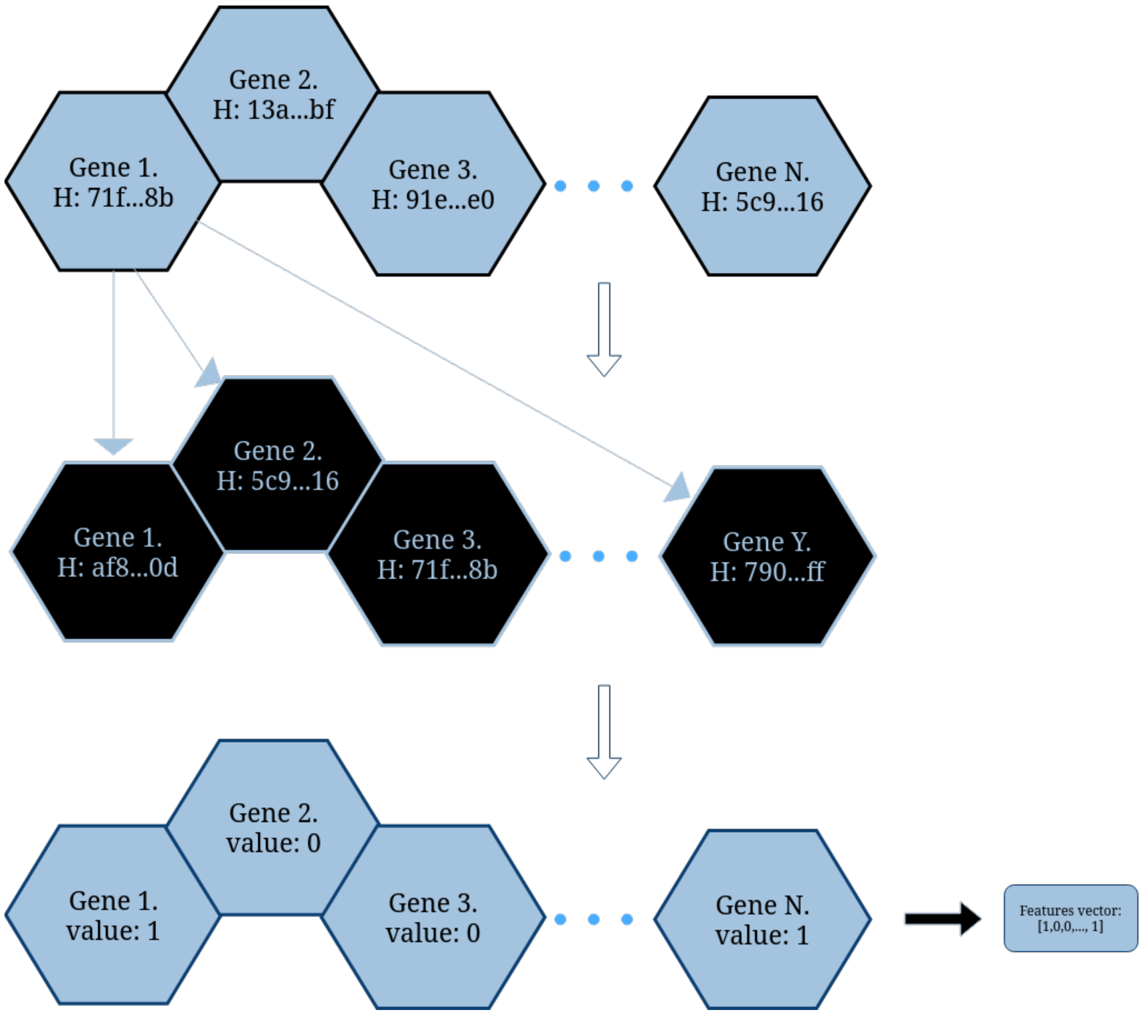


Figure 9. The process of generating a vector of categorical data.

Categorical representation is the most straightforward approach in machine learning for malware genes. However, this representation overlooks other crucial properties of malware genes, such as their size, entropy, or frequency of occurrence within samples. A malware gene's structure typically includes the number of bytes comprising it and the count of disassembled instructions. Consequently, it becomes feasible to store the size of a given malware gene instead of a singular value, thereby transforming the representation of the vector from categorical to numerical. In section 5.1.3, a more in-depth analysis is provided, accounting for the frequency of occurrence of malware genes from the Genome Database within malware samples.

5.1.2 Numerical Data Representation

Categorical data representation lacks the inclusion of malware gene size within the feature vector. Consequently, categorical representation fails to differentiate between a malware genome comprising more than a hundred disassembled instructions and a vector containing only one or two instructions. In reverse engineering terms, larger genes generally hold higher correlation values than smaller ones. This observation stems from the fact that larger genes encompass more program logic. Figure 6 illustrates the IDA CFG, where smaller IDA basic blocks often denote simple conditions and are thus less significant. Moreover, such genes are prevalent in numerous benign programs. Conversely, a malware gene containing up to 13 instructions possesses a considerably higher value. Therefore, this research also represents malware genes numerically based on their instruction count.

$$\begin{array}{cccc}
\text{Malware Gene 1} & \text{Malware Gene 2} & \dots & \text{Malware Gene N} \\
\hline
no\_instructions_{1,1} & no\_instructions_{1,2} & \dots & no\_instructions_{1,N} \\
9 & 13 & \dots & 93 \\
\vdots & \vdots & \ddots & \vdots \\
51 & 3 & \dots & 17
\end{array} \tag{5.2}$$

The schematic design of the feature matrix 5.2 utilizes the numbers of disassembled instructions solely for illustrative purposes. Numerical data representation is expected to provide more precise distance values for malware samples. Both numerical and categorical representations share a common feature: the interpretation of a malware gene's absence in a sample, denoted by a value of zero. However, if the malware gene is present in the malware at position i, its value is not merely one, but rather the value of the malware gene itself.

5.1.3 Optimalizations

During experimentation with machine learning of malware genes, several possible optimization algorithms were invented. These optimizations are designed to speed up the time needed to analyze malware genes or improve the accuracy of algorithms.

Selection of Relevant Genes

The initial optimization seeks to enhance the program's performance and reduce its time requirements. The optimization strategy selects only the relevant malware genes from the Genom Database rather than considering all available genes. This technique enables the selection of genes categorized as "function" type or those containing more than 15 instructions. In practice, databases may contain hundreds of thousands of malware samples, representing millions of malware genes. Managing such a vast volume of genes poses significant challenges, especially considering that genes are represented as features in machine learning.

It's common to define the set from which the model is generated to classify an unknown sample. Malpedia hosts a vast array of malware families. In practice, leveraging the Malpedia model to classify unknown samples, totaling 8118 samples, relieves the need to incorporate all database samples. When the sample is identified as belonging to one of the n malware families, the ML model can be restricted solely to these n families, significantly reducing computational complexity.

Frequency of Malware Genes

The basic algorithm's logic, exemplified in Listing 5.1, overlooks the frequency of malware genes within a sample. For instance, specific malware genes may be repeatedly present in various locations within the sample. As the Genome Database dictates the final structure of the feature vector, these recurrent genomes are tallied only once.

$$\text{no\_instructions} \times \text{no\_occurences} = \text{feature\_value} \tag{5.3}$$

The formula for calculating the feature value is presented in equation 5.3. This equation involves multiplying the sizes of malware genes by the number of occurrences of each gene in the sample. Utilizing these optimized values can enhance the structural representation of malware through vector features.

Neighboring Malware Genes

The final optimization technique aims to identify clusters or groups of neighboring malware genes. However, this approach is not applicable when creating the feature vector itself; instead, it is suitable for comparing or calculating distances between two vectors. The concept behind this optimization is that consecutive identical malware genes between samples exhibit a higher correlation value. Employing this technique makes it possible to enhance the contextual aspect of machine learning.

5.2 Data Pre-processing

Data selection and preprocessing represent crucial stages in machine learning. In this study, they hold particular significance as not all dataset samples are suitable for machine learning purposes. Genetic malware analysis employs static code analysis and machine learning techniques. This method is described in Section 3.2. Statically analyzing packed samples proves inefficient due to code obfuscation or storing code in non-executable segments and resources. Consequently, these samples must first undergo unpacking before static analysis can proceed. Addressing packer detection was thus an initial limitation to enable Genetic Malware analysis via machine learning.

5.2.1 Exclusion of Packed Samples

The purpose of this research is not to detect whether the sample is packed or not. However, in sections 1.5 and 3.1 it is mentioned that excluding packed samples helps the accuracy and efficiency of malware correlation. The solution to detect packers can be implemented using a malware gene, but it is beyond the scope of this research 1.2. Therefore, in this research, an already existing solution for detecting packed binaries named PeID [2] was integrated and improved. This tool contains more than 5500 static rules to detect languages, packers, compilers and also linkers.

```
[Simple UPX Cryptor V30.4.2005 -> MANtiCORE]
signature = 60 B8 ?? ?? ?? ?? B9 ?? ?? ?? ?? ?? ?? ?? ?? E2 FA
ep_only = true
```

Procedure 5.2. Example of PeID rule

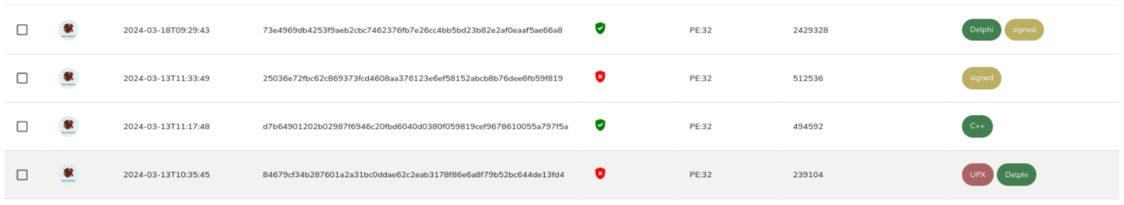
The characters "??" in the example PeID rule in Listing 5.2 represent wildcard bytes. These wildcard bytes most often replace address bytes, since they depend on the Image Base Address in the PE header [68]. Despite the large community that actively creates signatures, the PeID software has three main disadvantages.

- Cannot load binaries from the memory
- Signatures are not divided into categories, whether it is the detection of a programming language, packer, compiler or something else.
- PeID cannot detect multiple custom packers

The initial drawback of this tool has been addressed by rewriting it to enable the analysis of in-memory samples, in addition to those stored on disk. Beyond merely integrating PeID, this study also enhanced its functionality by categorizing signatures. PeID holds significant relevance in this research as it aids in excluding samples falling outside the defined scope 1.2. For example, the samples are written in the ".NET" programming framework, which were excluded from the research scope. The PeID tool allows a better understanding of sample categories in terms of qualitative research 1.6. This research can

[2]https://github.com/packing-box/peid

answer questions such as what have in common malware written in the "Delphi" or what correlation accuracy do have samples compiled in "MSVC" in comparison to "gcc". On the figure 10 it is possible to see based on Tags that the rewritten version of PeID managed to detect samples packed by UPX [3], written in C++ or Delphi.



☐		2024-03-18T09:29:43	73e4969db4253f9aeb2cbc7462376fb7e26cc4bb5bd23b82e2af0eaaf5ae66a8	✓	PE:32	2429328	Delphi signed
☐		2024-03-13T11:33:49	25036e72fbc62c869373fcd4608aa376123e6ef58152abcb8b76dee6fb59f819	✗	PE:32	512536	signed
☐		2024-03-13T11:17:48	d7b64901202b02987f6946c20fbd6040d0380f059819cef9678610055a797f5a	✓	PE:32	494592	C++
☐		2024-03-13T10:35:45	84679cf34b287601a2a31bc0ddae62c2eab3178f86e6a8f79b52bc644de13fd4	✗	PE:32	239104	UPX Delphi

Figure 10. PeID identification of packer and programming languages.

However, the detection of custom packers in this research still represents a challenge. The primary reason of it is that there are no rules in PeID that can detect these packers. On the figure 10 it may be seen the top sample, which was detected to be written in the Delphi programming language. From an analytical point of view, malware written in Delphi is not common, but it does occur occasionally. However, several packers and protectors are written in Delphi languages, such as BobSoft [69]. Therefore, this sample was more deeply examined from the point of view of qualitative research in the 6.1 section. The objective of this examination is to validate if PeID correctly matched sample to be written in Delphi.

5.2.2 Malware Family Name Aliases

One of the biggest challenges of this research as well as other research focused on malware analysis is the inconsistency in the naming of malware families [70]. With supervised machine learning, it is important to have a label to which malware family the samples belong. This assignment to a malware family is often referred to as the 'label' of the malware sample. Hurier et al. [71] in research describes this problem on Android malware. This problem is most obvious primarily with Antivirus programs (AV). The problem with malware naming is demonstrated in Table 1, where several AV products were selected. The sample analyzed by these products has the SHA256 value '3d574af4a43dd7ae7244cdbb6381af34fbad237a93627c47b6ca07ff7a8f04c6' [4]. This sample is labeled as MarsStealer on the MalwareBazaar malware repository. The goal of this malware is often stealing cryptocurrencies and attacking Multi-Factor authentication (MFA) [22].

[3]https://upx.github.io/
[4]https://bazaar.abuse.ch/sample/3d574af4a43dd7ae7244cdbb6381af34fbad237a93627c47b6ca07ff7a8f04c6/

Some AV companies classify malware with a generic name. Trellix [5] assigned the name to the sample in the name table ‘Generic.mg.b9c5f3129fee3344’. Other AV products have labeled this malware based on the type of malware, such as trojan or ransomware. Several AV products did not even match the type of malware. Finally, some AV products, instead of naming the malware based on the type or malware family, only evaluated it as ‘Malicious’.

Antivirus Product Name	**Malware name**
AVAST	Win32:RansomX-gen [Ransom]
Elastic	Malicious (high Confidence)
Eset	A Variant Of Win32/Kryptik.HVIU
Malwarebytes	Trojan.MalPack.GS
Microsoft	Trojan:Win32/Amadey.GAA!MTB
McAfee	Artemis!B9C5F3129FEE
Symantec	ML.Attribute.HighConfidence

Table 1. Malware names assigned by Antivirus products

The problem of inconsistency does not belong to the scope of this research. However it is necessary to solve it to be possible correlate samples from several sources. Two strategies were chosen in this research:

- Malware family aliases
- Name mangling

The solution to this problem was partially simplified for us by the excellent Database of malware families on the Malpedia [22] platform. Malpedia contains more than 2200 malware families, which also contain aliases for tax families. The malware called QakBot 6.1 contains four other aliases under which it is identified by other AV products or on other malware repositories. The second strategy was “Name mangling”, which was created during the experiment described in section 6.3. This strategy tries to respond to a slight inconsistency in the names of malware families.

5.3 Software Design

Software design is a section that serves chronologically before implementation as a specification of requirements. A major aspect of the design phase focused on creating a new module within the MalTraits project named "Experiments." This part aims to specify a generic environment for the preparation, execution, and evaluation of experiments. It is important to defend and specify the programmed components for the applied part of this research, which represent a software platform. Therefore, the following sections describe

[5]https://www.trellix.com

the implemented components and also simple wireframes.

5.3.1 Experiment Environment

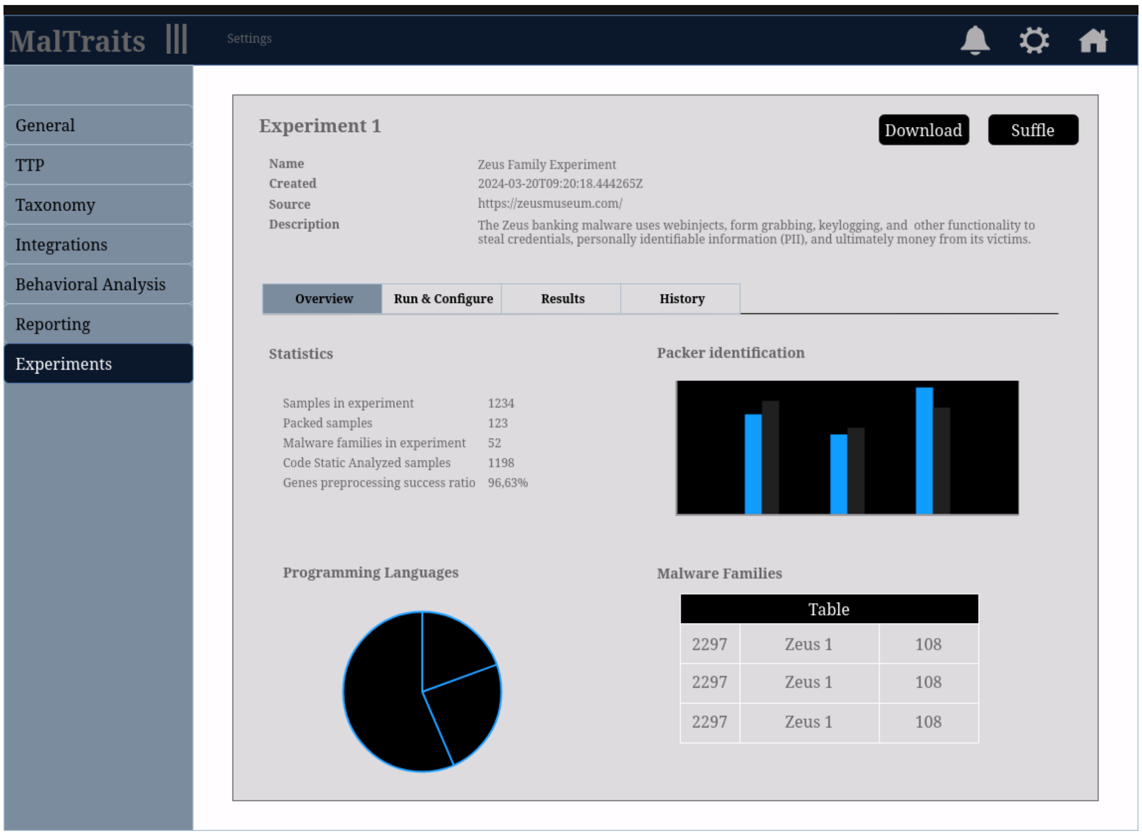
One of the most important parts of this research is experimentation in the 6 section, which verifies which implemented algorithms are the best for malware correlation. However, in order to be able to perform experiments effectively, it is necessary to have a pre-prepared analytical environment. The purpose of this Experiment Environment is to simplify the evaluation of results in machine learning of malware genes. This research explores a completely new approach to malware analysis, which is built on data made up of millions of malware genes. In addition, the research tests several kinds of datasets and experiments. For this reason, the goal is to create a uniform experimental environment that will enable the evaluation of the results. In the results, it is possible to obtain the accuracy rate of machine learning, evaluate the quality of the dataset and also configure various parameters. This environment should meet the following requirements:

- Download and static analysis of selected samples. These samples are finally assigned to the experiment.
- Dataset overview and statistics from the point of view of packers, programming languages.
- Configuration of the correlation algorithm and exclusion criteria.
- Summary of results from the last experiment
- History of all attempts in a given experiment.

5.3.2 Wireframes

The experiment in this research represents specific dataset or research approach that is evaluated. The experiment includes datasets from Zeusmuseum, Malpedia and other malware sources, which are listed in the section 1.6. The aim of these experiments is to evaluate to what extent malware gene correlation is effective.

The main part of the screen contains two buttons as can be seen in the figure 11. The first 'Download' button downloads samples from the given source. However, the sample download format is not uniform depending on the platform from which the data is drawn. Somewhere it is necessary to hack the HTML code of the website and get URL links to the malware. This approach is applied in the Zeusmuseum experiment 6.2. In other cases, it is enough to simply call the API, which returns the sample itself in the response. However, the sample in the background is not just downloaded. After downloading, a basic



Experiment 1

Name	Zeus Family Experiment
Created	2024-03-20T09:20:18.444265Z
Source	https://zeusmuseum.com/
Description	The Zeus banking malware uses webinjects, form grabbing, keylogging, and other functionality to steal credentials, personally identifiable information (PII), and ultimately money from its victims.

Statistics

Samples in experiment	1234
Packed samples	123
Malware families in experiment	52
Code Static Analyzed samples	1198
Genes preprocessing success ratio	96,63%

Malware Families

Table		
2297	Zeus 1	108
2297	Zeus 1	108
2297	Zeus 1	108

Figure 11. Design Wireframe of the Experiment page.

static analysis is automatically started, which evaluates whether the sample is suitable for Genetic malware analysis. If the sample is supported and not packed, the static code analysis is automatically started. This process is described in more detail in section 3.2.

The main part ‘Overview’ shows statistics about the analyzed dataset as a whole. These statistics are important, because in them it is possible to find many answers for very positive or negative results. For example, negative results can occur when the dataset contains a large number of packed samples. Detection of the programming language is a parameter that plays an important role in the interpretation of the results.

The second screen presents the configuration of the experiment and the machine learning algorithm. The aim is to compare various parameters of the experiments, such as the ‘k’ count in the KNN algorithm. Exclusion parameters were also included in the configuration, such as the minimum number of labeled samples per malware family. This configuration option enables a better understanding of the nature of the dataset.

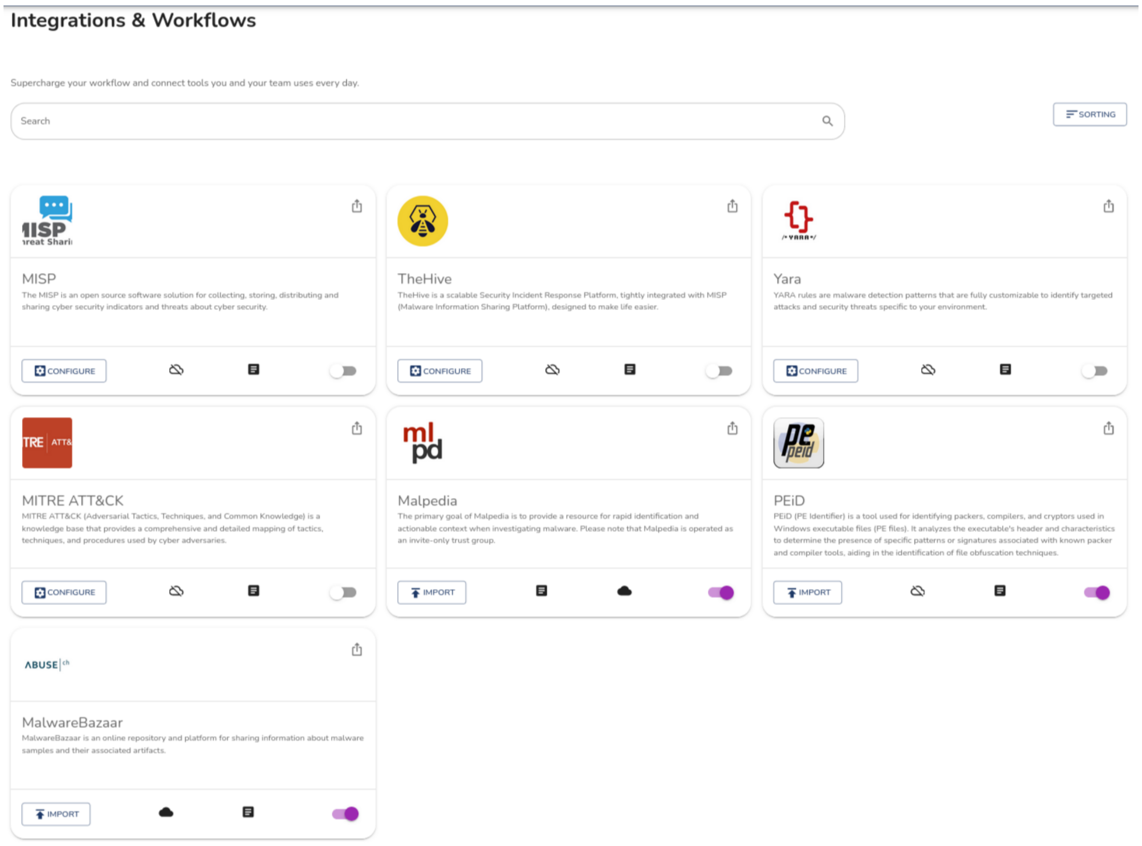
Finally, an important part of experiments is their evaluation. The experiment can be evaluated at the level of a sample, malware family or the whole. Functional requirements also include recording the history of experiment results and their parameters for a better evaluation of the results. The experiment environment gives security analysts better options for interpreting the results of Genetic Malware Analysis.

5.4 Implementation

The research objective is not only to provide insights into feature engineering through experimental results but also to deliver software capable of analyzing malware samples in an on-premise environment. This software, known as MalTraits, was developed in the author's previous research and facilitates the proactive collection and automated analysis of malware samples from the internet. As emphasized in previous sections, the main focus of this research is to establish an environment for the application of machine learning. Hence, the following modules were developed as part of this endeavor:

- **Experiment environment** - The implementation of a versatile environment for conducting experiments involves several components. This includes developing functionality for fetching malware samples, automating static analysis, vectorizing genes, and integrating machine learning algorithms.
- **Integrations** - Integrating MalTraits with various platforms enables automatic retrieval of malware samples. The implementation of this solution incorporates automated sample processing, depicted in Figure 5.
- **Computation of the malware correlations** - The visual representation of malware correlations has been previously implemented in prior research. In this study, additional correlations were computed to determine the extent of similarity between samples. Furthermore, enhancements have been made to the visual representation of sample correlations.
- **Other tasks** - The other tasks in this research primarily served a supportive role and were not directly related to the main objective, which is the application of machine learning. Nonetheless, these tasks encompassed essential enhancements for a practical platform, such as developing preliminary analysis, ensuring clarity in the malware tables, and establishing connections with malware families.

The main objective of the implementation was to establish an experimental environment that was facilitative in testing machine learning algorithms. However, the scope of the experiments extended beyond merely programming the algorithms themselves. Research contribution encompassed developing an automated pipeline capable of downloading samples from the Internet and executing "Preliminary analysis" automatically. This preliminary analysis involved extracting strings and categorizing them, checking for certificates in signed samples, and utilizing the PeID tool for detection. The PeID tool was completely rewritten to enable sample analysis in memory. Subsequently, based on the preliminary


Integrations & Workflows

Supercharge your workflow and connect tools you and your team uses every day.

MISP
The MISP is an open source software solution for collecting, storing, distributing and sharing cyber security indicators and threats about cyber security.

TheHive
TheHive is a scalable Security Incident Response Platform, tightly integrated with MISP (Malware Information Sharing Platform), designed to make life easier.

Yara
YARA rules are malware detection patterns that are fully customizable to identify targeted attacks and security threats specific to your environment.

MITRE ATT&CK
MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) is a knowledge base that provides a comprehensive and detailed mapping of tactics, techniques, and procedures used by cyber adversaries.

Malpedia
The primary goal of Malpedia is to provide a resource for rapid identification and actionable context when investigating malware. Please note that Malpedia is operated as an invite-only trust group.

PEiD
PEiD (PE Identifier) is a tool used for identifying packers, compilers, and cryptors used in Windows executable files (PE files). It analyzes the executable's header and characteristics to determine the presence of specific patterns or signatures associated with known packer and compiler tools, aiding in the identification of file obfuscation techniques.

MalwareBazaar
MalwareBazaar is an online repository and platform for sharing information about malware samples and their associated artifacts.

Figure 12. Contributed Integrations to the research

analysis results, a decision was made regarding whether a given sample fell within the scope of this research. The scope of this research was set to analyze samples that were not packed and written by a supported programming language. Contribution in the field of machine learning and experiments is described in detail in the Validations sections 6.2 and 6.3.

5.4.1 Integrations

Integrations played a pivotal role in the research, as they established connections to the data sources crucial for genetic malware analysis and experimentation. Notably, automatic integrations were implemented for MalwareBazaar, Malpedia, and Zeusmuseum. Additionally, the PeID tool was forked and customized for research objectives, enabling the incorporation of custom rulesets. Moreover, integration with the MITRE Att&ck framework was implemented, providing access to a dataset encompassing threat actors, malware families, and TTPs.

Integration with MalwareBazaar can be configured via a web browser to download newly added samples from the Internet every hour. During experimentation with this integration, it was discovered that automated sample post-processing can yield IOCs that have not been publicly shared.

5.4.2 Malware Correlations

This module aims to facilitate visual and percentage-based assessment of correlations between samples categorized under specific malware families. Given the complexity of the database model, certain cases necessitated using raw PostgreSQL queries instead of Django Object Relational Mapping (ORM). An example of such a complex SQL query is provided in Procedure 5.3, which retrieves the top three malware families exhibiting the highest correlation with the sample. The graphically implemented functionality is illustrated in Figure 15.

This functionality is invaluable for a malware analyst who analyzing an unknown sample, aiming to assess correlations swiftly. The malware analyst can thus answer very quickly whether it is custom malware or a known malware family if the dataset of malware samples is sufficient. If it does belong to a known family, leveraging public malware analyses can expedite the overall analysis process. In the case of custom malware, identifying shared Tactics, Techniques, and Procedures (TTP) within the code becomes feasible. Currently, the visual correlation functionality for samples is tailored exclusively for at-scale comparison methods, but there is potential to enhance it with machine learning correlation results.

```sql
SELECT * FROM (
    SELECT DISTINCT ON (sub2.mf_name)
        sub2.s1,
        sub2.s2,
        COUNT(*) OVER (PARTITION BY sub2.s2),
        sub2.mf_name,
        sub2.mf_id
    FROM (
        SELECT DISTINCT
            sub1.sample_id AS s1,
            jt.sample_id AS s2,
            sub1.trait_id ,
            mf.name AS mf_name,
            mf.id as mf_id
        FROM (
            SELECT
                st.sample_id,
                trait.id AS trait_id
            FROM api_sampletrait AS st
```

```
                INNER JOIN api_trait AS trait ON st. trait_id = trait . id AND st.sample_id
                     = %s
                WHERE trait."size" > 0
           ) AS sub1
           INNER JOIN api_sampletrait AS jt ON sub1. trait_id = jt . trait_id AND NOT
                jt.sample_id = %s
           INNER JOIN api_samplemalwarefamily AS smf ON jt.sample_id = smf.sample_id
           INNER JOIN api_malwarefamily mf ON mf.id = smf.malware_family_id
      ) sub2
      ORDER BY sub2.mf_name, COUNT(*) OVER (PARTITION BY sub2.s2) DESC
) AS sub3
ORDER BY count DESC
LIMIT 3;
```

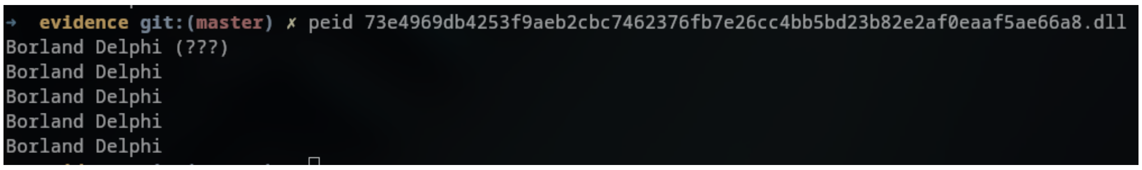
Procedure 5.3. Get top three malware family correlations

6. Validation

This research tries to bring a new method of involving data analysis in malware analysis. For this reason, it is very important to validate the solution, because whether in static or genetic analysis, many borderline cases can occur. Validation is based on the implemented parts in the Contribution section 5. Its goal is to reveal whether this solution works as it should and, above all, to better understand the research problem. This section contains only part of the experiments that were carried out during the research. The experiments were a key point that shifted the value of this research and at the same time opened up other research questions.

6.1 Identification of Custom Packer by Malware Genes

Section 5.2.1 analyzed the possibilities of detecting packers using static PeID signatures. These signatures are open-source and therefore community-created. The figure 13 shows the use of the PeID tool, which did not detect any use of the packer and also detected that the code is written in the Delphi language. The aim of this qualitative research is to validate these results. To validate this, it was reversed engineer sample from the QakBot family. QakBot malware family is categorized as information stealer malware [22]. This malware is often used by threat actors as an entry vector for ransomware [72].



```
→  evidence git:(master) ✗ peid 73e4969db4253f9aeb2cbc7462376fb7e26cc4bb5bd23b82e2af0eaaf5ae66a8.dll
Borland Delphi (???)
Borland Delphi
Borland Delphi
Borland Delphi
Borland Delphi
```

Figure 13. PeID packer detection in QakBot sample

In the table 2 is the identification of the analyzed sample based on the hash value as well as part of its basic static analysis. Executable section with the name ‘CODE’ of the analyzed sample has an entropy at the level of 6.54. Entropy is a property that statistically determines the randomness factor. It is often used to detect packed samples, since compressed or encrypted parts have high entropy [73]. On the contrary, clean code without any protection often has a lower entropy.

Packed samples usually have high entropy at the section level, have few imports and IATs, and also do not contain meaningful strings. This sample does not have a high entropy executable section, but its ‘DATA’ section contains a high entropy (7.95). For this reason, the assumption of the analysis is that the strings are decrypted during the execution runtime. However, this technique is quite common in malware and does not confirm or refute whether the sample is packed. There are enough imported libraries and methods in the IAT structure. However, based on publicly available malware analyses, it can be said that most QakBot samples use the custom packer [74]. Therefore, the next part of this research deals with verifying whether it is really a packed sample and therefore the PeID signatures failed to detect the packer.

Sample Attribute	**Attribute Value**
SHA1 Hash	93b1ab0a9e70a546c4b89dcb20a158dfc90b1421
File Size	2,43MB
Entrypoint	0x551f5c
Machine Type	I386
Subsystem	WINDOWS_GUI
Signed	False
File Type	PE
File Arch	32
Number of malware genes	102108
Number of block genes	96779
Number of function genes	5329
Number of invalid instructions	0

Table 2. Basic Static Analysis of QakBot sample

The analyzed sample is a PE Dynamic-link library (DLL), therefore the program ‘rundll32.exe’ is needed to execute it. Only one entry named ‘DLLRegisterServer’ was found in the Export Address Table. It was used a debugger named x32dbg [1] for unpacking. Breakpoints were set in the debugger for the following Windows API calls:

- VirtualAlloc
- VirtualProtect [2]
- CreateProcessW [3]

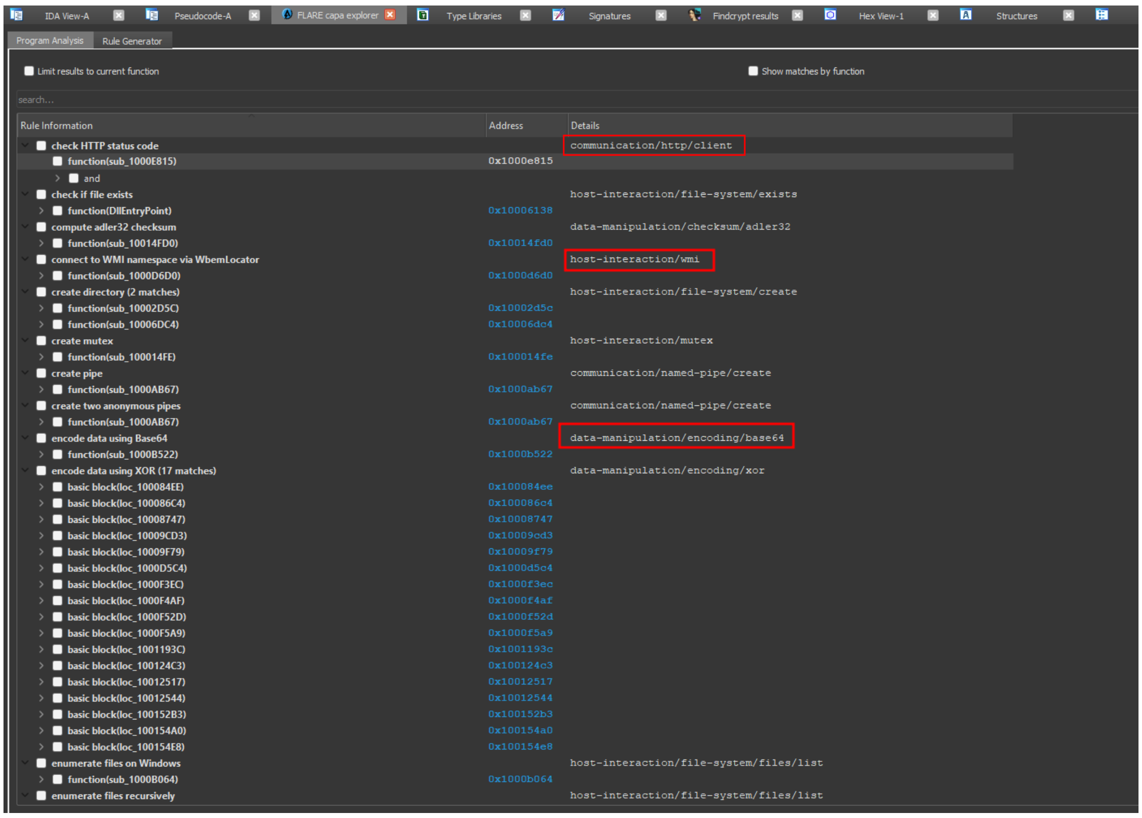
One of the frequently used unpacking strategies is to set a breakpoint on the API called ‘VirtualAlloc’. VirtualAlloc is used when programming to allocate memory on Windows OS and contains four parameters [75]. Based on the arguments for VirtualAlloc, it is possible to find the address, size and privileges of the allocated space. When the program unpacks the malware, it often allocates a memory location where the given code is stored.

[1]https://x64dbg.com/

[2]https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotect

[3]https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw

This strategy also worked in this case, when after the fifth breakpoint on VirtualAlloc it copied an unpacked QakBot sample.



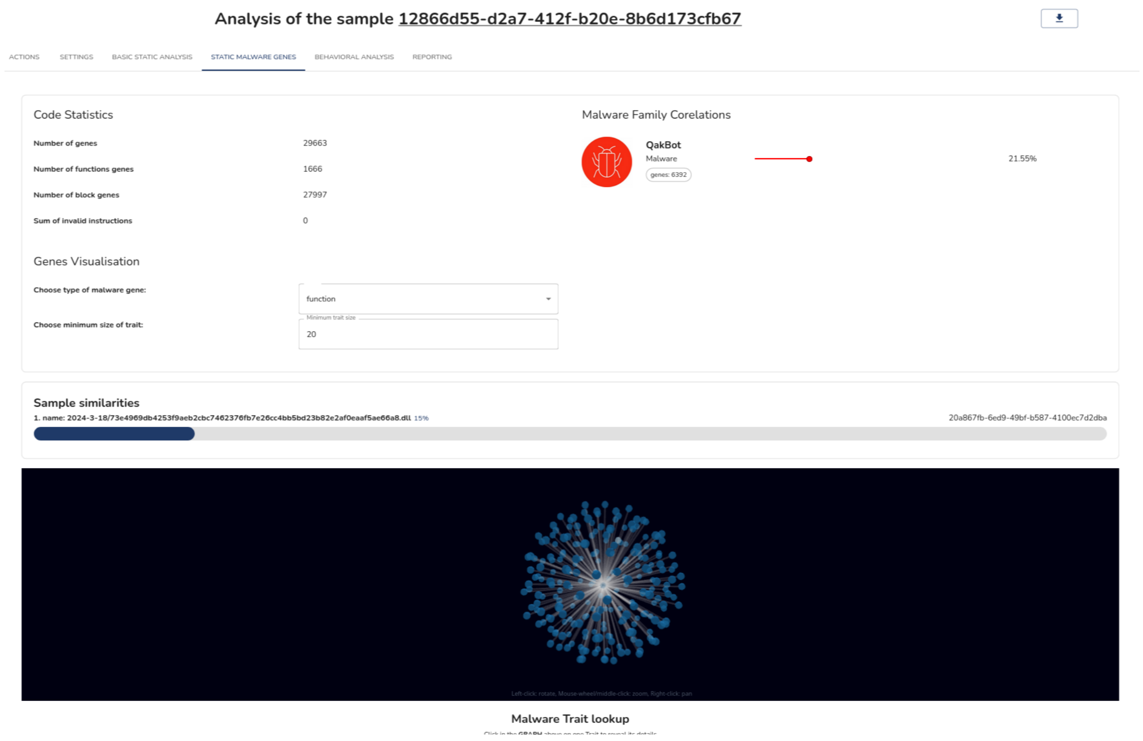
Rule Information	Address	Details
check HTTP status code		communication/http/client
function(sub_1000E815)	0x1000e815	
and		
check if file exists		host-interaction/file-system/exists
function(DllEntryPoint)	0x10006138	
compute adler32 checksum		data-manipulation/checksum/adler32
function(sub_10014FD0)	0x10014fd0	
connect to WMI namespace via WbemLocator		host-interaction/wmi
function(sub_1000D6D0)	0x1000d6d0	
create directory (2 matches)		host-interaction/file-system/create
function(sub_10002D5C)	0x10002d5c	
function(sub_10006DC4)	0x10006dc4	
create mutex		host-interaction/mutex
function(sub_100014FE)	0x100014fe	
create pipe		communication/named-pipe/create
function(sub_1000AB67)	0x1000ab67	
create two anonymous pipes		communication/named-pipe/create
function(sub_1000AB67)	0x1000ab67	
encode data using Base64		data-manipulation/encoding/base64
function(sub_1000B522)	0x1000b522	
encode data using XOR (17 matches)		data-manipulation/encoding/xor
basic block(loc_100084EE)	0x100084ee	
basic block(loc_100086C4)	0x100086c4	
basic block(loc_10008747)	0x10008747	
basic block(loc_10009CD3)	0x10009cd3	
basic block(loc_10009F79)	0x10009f79	
basic block(loc_1000D5C4)	0x1000d5c4	
basic block(loc_1000F3EC)	0x1000f3ec	
basic block(loc_1000F4AF)	0x1000f4af	
basic block(loc_1000F52D)	0x1000f52d	
basic block(loc_1000F5A9)	0x1000f5a9	
basic block(loc_1001193C)	0x1001193c	
basic block(loc_100124C3)	0x100124c3	
basic block(loc_10012517)	0x10012517	
basic block(loc_10012544)	0x10012544	
basic block(loc_100152B3)	0x100152b3	
basic block(loc_100154A0)	0x100154a0	
basic block(loc_100154E8)	0x100154e8	
enumerate files on Windows		host-interaction/file-system/files/list
function(sub_1000B064)	0x1000b064	
enumerate files recursively		host-interaction/file-system/files/list

Figure 14. Flare Capa plugin in IDA Pro.

After unpacking the QakBot sample, it was used the IDA Pro tool and an excellent plugin called Flare CAPA [4]. In the results of the plugin, it is possible to clearly see places where HTTP communication, string decryption or dynamic loading of libraries takes place. After the static analysis of the code of the unpacked sample, it can be confirmed that it is the final stage of unpacking. In Appendix 8 it is attached a more detailed analysis of the code with string decryption. This analysis proves that the PeID tool could not capture the custom packer for the QakBot malware.

However, with the help of malware genes, it is also possible to solve this lack of static rules. From public malware analyses it is known that QakBot uses a custom packer written in Delphi [74]. Genetic malware analysis examines correlations at the binary code level and does not distinguish whether it is the packer code or the malware sample itself. For this reason, it is possible to use the fact that it is a custom packer specifically for QakBot. In another experiment, it was verified whether it is possible to correlate QakBot malware based on their custom packer.

[4]https://github.com/mandiant/capa


Analysis of the sample 12866d55-d2a7-412f-b20e-8b6d173cfb67

Code Statistics

Number of genes	29663
Number of functions genes	1666
Number of block genes	27997
Sum of invalid instructions	0

Malware Family Corelations

QakBot
Malware
genes: 6392
21.55%

Genes Visualisation

Choose type of malware gene: function

Choose minimum size of trait: Minimum trait size 20

Sample similarities

1. name: 2024-3-18/73e4969db4253f9aeb2cbc7462376fb7e26cc4bb5bd23b82e2af0eaaf5ae66a8.dll 15% 20a867fb-6ed9-49bf-b587-4100ec7d2dba

Left-click: rotate, Mouse-wheel/middle-click: zoom, Right-click: pan

Malware Trait lookup

Figure 15. QakBot correlation based on genes of custom Packer.

For malware analysis, it has been obtained a sample that has a TLP, which does not allow it to be provided to third parties. Its basic static parameters are in Table 3. Genetic Malware Analysis was used for the initial classification of the sample. The aim of this classification was to find out whether it is a known malware family. The classification on the MalTraits platform is captured in the Figure 15. After a few seconds of automated Genetic Analysis, it was revealed that the sample contains a high correlation factor with the QakBot malware family.

Sample Attribute	Attribute Value
SHA1 Hash	19434176868e295ae703d60e61751d9f755831bd
File Size	712kB
File Type	PE
File Arch	32
Number of malware genes	29663
Number of block genes	27997
Number of function genes	1666
Number of invalid instructions	0

Table 3. Basic Static Analysis of Unknown sample

6.2 Classification of the Zeus Malware Family

Zeus is a complex malware that is categorized as a ‘banking trojan’ [76, 22]. Malicious features of this malware include keylogging, personal data stealing, but also Antivirus

Evasion techniques [77]. This research experiment was inspired by the excellent project 'zeusmuseum' [5]. In 2011, the source of the Zeus malware was published [6], which caused a large increase of its modifications [78]. Several of these modifications were actually given the malware family name. The aim of this experiment is to try to classify sub-families of Zeus malware using Genetic Malware Analysis and KNN algorithm.

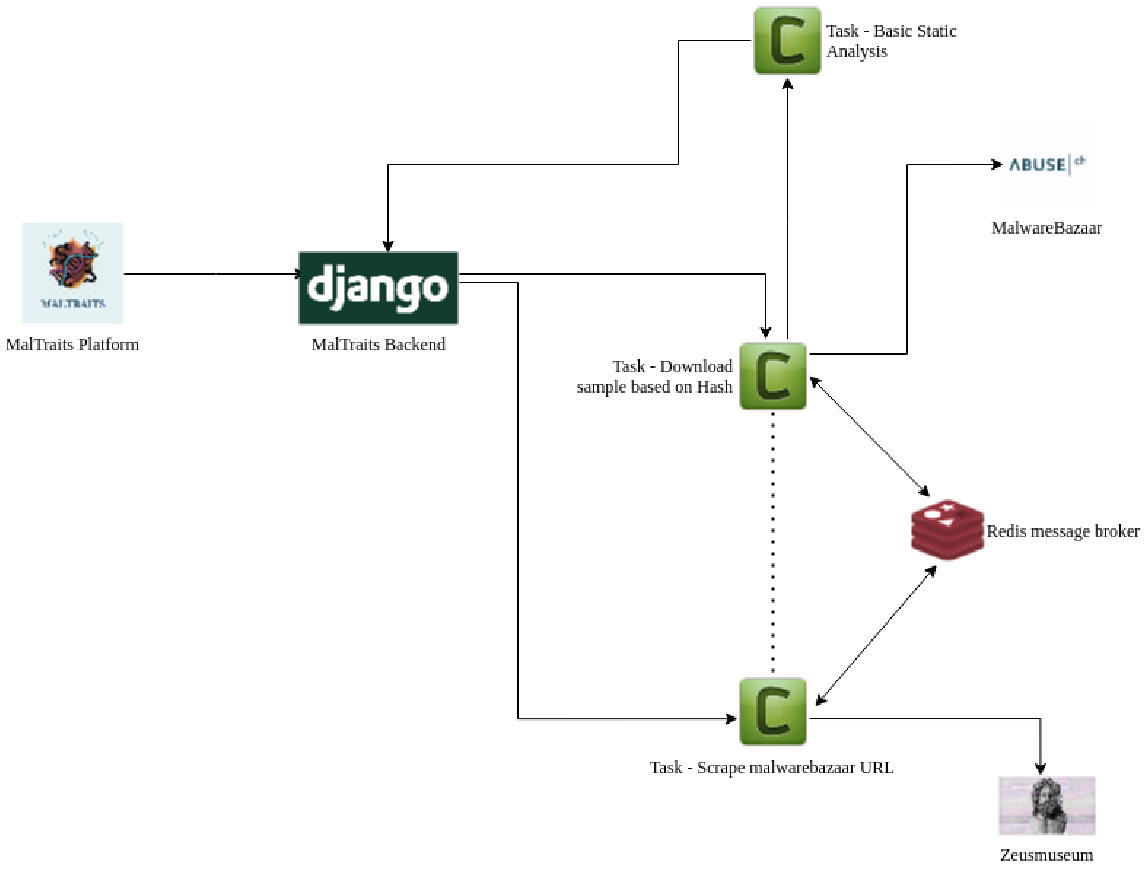
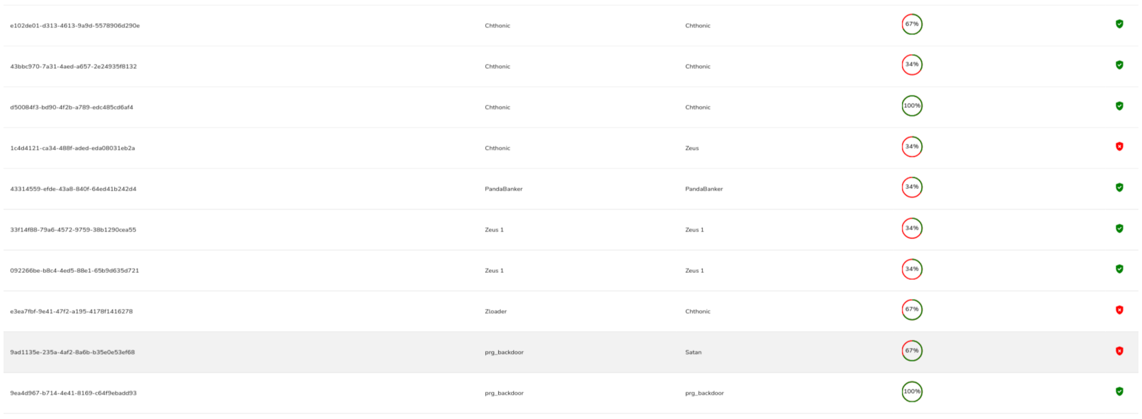

Task - Basic Static Analysis
MalwareBazaar
MalTraits Platform
MalTraits Backend
Task - Download sample based on Hash
Redis message broker
Task - Scrape malwarebazaar URL
Zeusmuseum

Figure 16. Zeus malware experiment High-level design (HLD).

This experiment showed values that are very important for benchmarking the Genetic Malware Analysis itself. In total, it was possible to obtain 566 labeled malware samples belonging to 37 Zeus sub-families. From this dataset, it is possible to obtain up to 1,521,505 unique malware genes after static analysis. Based on the results of the experiment, it was found that one sample has an average of 9,975 malware genes. Static analysis may not always be possible due to several possibilities, such as an unsupported disassembler for .NET binary programs 3.1. In other cases, the sample may not have a standard structure. In this experiment, algorithm managed to process the malware genes of 96.28% of the samples.

[5]https://zeusmuseum.com/
[6]https://github.com/Visgean/Zeus

After downloading the samples and the subsequent static analysis, it is necessary to configure the exclusion parameters and the configuration of the KNN algorithm. The configuration parameters reflect the scope of the research and their design is described in section 5.3.2.



e102de01-d313-4613-9a9d-5578906d290e	Chthonic	Chthonic	67%	🟩
43bbc970-7a31-4aed-a657-2e24935f8132	Chthonic	Chthonic	34%	🟩
d50084f3-bd90-4f2b-a789-edc485cd6af4	Chthonic	Chthonic	100%	🟩
1c4d4121-ca34-488f-aded-eda08031eb2a	Chthonic	Zeus	34%	🟥
43314559-efde-43a8-840f-64ed41b242d4	PandaBanker	PandaBanker	34%	🟩
33f14f88-79a6-4572-9759-38b1290cea55	Zeus 1	Zeus 1	34%	🟩
092266be-b8c4-4ed5-88e1-65b9d635d721	Zeus 1	Zeus 1	34%	🟩
e3ea7fbf-9e41-47f2-a195-4178f1416278	Zloader	Chthonic	67%	🟥
9ad1135e-235a-4af2-8a6b-b35e0e53ef68	prg_backdoor	Satan	67%	🟥
9ea4d967-b714-4e41-8169-c64f9ebadd93	prg_backdoor	prg_backdoor	100%	🟩

Figure 17. Experiment 1. - KNN Classification Zeus Dataset

The results of this experiment indicate only a 43.75% accuracy rate. These results can be caused by several factors:

- The code failed to detect and subsequently exclude packers
- The sub-families are so closely related that it is difficult to find differences, since they come from the same source code.
- The KNN algorithm fails in multidimensional spaces.

In order to answer these questions, in this research continues with further experiments with other datasets and malware correlation methods.

6.3 Malpedia Dataset Experiment

Malpedia offers researchers a malware dataset that contains more than 2000 malware families composed of PE files. One of the great advantages of the dataset from Malpedia is that it already contains a classification of samples that are packed, unpacked and dumped. Only samples marked as unpacked in the Malpedia dataset were included in this experiment.

6.3.1 Selection of Representative Malware Families

The first experiments were conducted on the entire malware family dataset from the Malpedia dataset. However, testing all malware families in a given dataset is not an ideal solution for experimentation for several reasons:

- It is not straightforward to look for corner cases, where it is necessary to use qualitative research and take a closer look at the malware samples and why they are misclassified.
- Malware families with a small number of samples are often misclassified.
- The classification process, particularly the KNN algorithm, places significant demands on Random-access memory (RAM). Consequently, the analysis of thousands of samples at once becomes a memory-intensive operation.

For this reason, ten malware families were selected for the experiments. These families were selected based on the number of samples that meet predefined criteria from the research scope. Selected representative malware families are located in Table 4.

Malware Family Name	**no. Samples**	**Malware Family Category**
QakBot	103	Information Stealer
Gozi ISFB	46	Trojan
Emotet	27	Trojan
DreamBot	19	Trojan
BianLian	15	Ransomware
DarkGate	14	Loader/Dropper
DBatLoader	12	Loader/Dropper
DoppelDridex	11	Trojan
Cobalt Strike	11	Penetration testing toolkit
BazarBackdoor	9	Loader/Dropper

Table 4. Representative Malware Families

The experiment includes 267 malware samples from 10 different malware families. These samples represent different malware categories written by different programming languages. The malware samples of the BianLian family are written in the GoLang language. Other malware families, such as Emotet, are written in C/C++. This set of samples from different malware families represents a broad spectral data sample.

6.3.2 KNN Classification of Whole Dataset

KNN was chosen as the malware classification algorithm. The Euclidian distance algorithm was used to determine the distances and the data representation was categorical. After five experiments, a classification accuracy of 70.08% was achieved. The most accurate experiment run achieved accuracy at the level of 78.19%. Its configuration was as follows:

- Labeled samples: 266
- Unlabeled samples: 135
- k parameter: 3
- Exclusion criteria - Minimal number of "block" malware genes: 1000
- Exclusion criteria - Minimal number of "function" malware genes: 100

The achieved accuracy rate is possibly even improved by applying numerical data representation. The fact that some malware families contain only a small number of samples also has an impact on the results of this experiment.

6.3.3 KNN Classification of Subset Malware Families

In the 6.3.1 section, ten malware families are selected that meet the specified criteria for experimentation. In this dataset, several experiments were conducted to determine the correct configuration of malware genes for machine learning.

The first experiment was performed completely basic, without any optimizations or exclusion criteria for samples. The representation of samples was used in its basic form — as a categorical representation of data. The KNN algorithm was set to three neighbors, i.e., $k = 3$. Labeled samples in this experiment represented 73.78% of the dataset, and the remaining were unlabeled.

The experiment included all 267 representative malware samples and lasted 26 minutes and 55 seconds. The overall accuracy achieved in this experiment was 80%, and with the malware families QakBot, Cobalt Strike, DoppelDridex, and others, it was possible to achieve 100% accuracy.

Figure 18 graphically shows the accuracy of this experiment at the level of individual malware families. This experiment misclassified four malware families. The DreamBot malware family has a classification accuracy of 25%. However, it is important to interpret this result to understand the history and origin of the DreamBot malware family. The reason for misclassification of the DreamBot family is similar to the case of the first Experiment with the Zeus family. DreamBot is a malware family that is a successor and variant of the ISFB family. In 2013, the source code of the ISFB malware was leaked along with detailed [79] documentation. Although the experiment misclassified the malware family, it precisely revealed the similarity of the malware. This information is also very

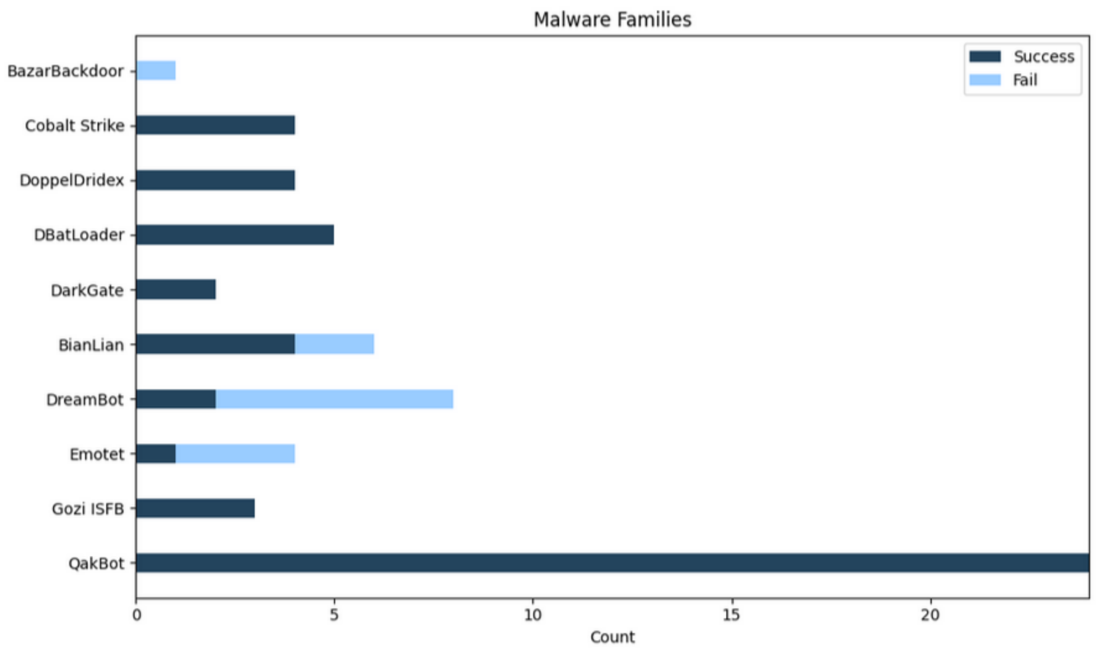


Figure 18. Experiment 2 - Classification results.

important for malware analysis.

Some optimizations described in the 5.1.3 section were also tested in this experiment. The first goal was to reduce the time and memory complexity of the algorithm. During testing, there were 1920 malware samples in the database, which had 4787021. By selecting only the selected dataset, it was possible to reduce the time complexity up to 10 times. The final time complexity of the algorithm was 2 minutes and 41 seconds.

6.4 Performance Benchmarking of the Solution

This study evaluated various actions commonly undertaken during malware analysis through benchmarking. One benchmark assessed the processing time required for conducting Preliminary Analysis on thousands of samples. The benchmarking process was conducted using a personal laptop with specifications outlined in Table 5. It took approximately 55 minutes to perform preliminary analysis on 1000 malware samples.

Hardware Component	**Hardware properties**
CPU model	Intel Core i9 13980HX Raptor Lake
CPU cores	32 cores
RAM	32GB
GPU	NVIDIA GeForce RTX 4080 12GB

Table 5. Benchmarking of the Solution

Constructing a data model from 250 malware samples and subsequently classifying 150 samples generally takes around 30 minutes on average. However, it is important to note that this benchmark accounts for the fact that the Genome Database already contains numerous unrelated samples from other experiments. This fact resulted in the feature vectors being much larger than the sum of the unique malware genes in the experiment.

The optimizations discussed in Sections 5.1.3 and 5.1.3 were also implemented during the solution's verification process. These optimizations significantly reduced the time required for machine learning by more than ten times. Before optimization, the experiment duration was 26 minutes and 55 seconds, whereas with optimization from Section 5.1.3, it was reduced to just 2 minutes and 41 seconds. Following the implementation of the second optimization, the experiment duration totaled 6 minutes and 8 seconds. These results confirm the necessity of optimizations in achieving the objectives outlined in Hypothesis $H0$ from Section 1.1.

7. Discussion

A pivotal aspect of this study involved exploring the potential applications of machine learning for a novel category of malware genes. Given that this type of malware genes had not been utilized in this context previously, it was imperative to establish a comprehensive data representation framework from scratch. By utilizing the forked and modified PeID tool, packed samples were effectively excluded during the Data Pre-processing stage. Additionally, it was discovered that setting a minimum size of 500 malware genes significantly increased the probability of excluding packed samples.

This research identified various potential data representations as well as several optimization techniques. The representation of malware gene feature vectors is grounded in unique genes sourced from the Genome Database. Following the implementation of optimizations, this representation of the input vector for machine learning proves highly effective. While this solution offers advantages, it also presents drawbacks in the form of high-dimensional space. However, the complexity of the high-dimensional space can be mitigated by employing algorithms such as Principal Component Analysis (PCA) [80].

Experimental research achieved notable accuracy rates exceeding 80% in malware classification. After these results, it can be concluded that hypothesis $H1$ can be fulfilled with this solution. The highest accuracy values were achieved by combining the KNN algorithm with the first optimization detailed in Section 4.10. The K parameter was set to three neighbors, and the Euclidean algorithm was employed to calculate the distance between vectors. However, experimental results suggest that genetic malware analysis may not yield high accuracy for malware families sharing the same code base. Low accuracy was evident in the case of the Zeus families from the first experiment and the DreamBot family from the second experiment. These families share a commonality: the root malware family code is public, and many sub-families reuse a significant portion of the initial code.

Over 1000 samples were processed within an hour through automated processing across various integrations. This solution enables the automated processing of 24 thousand samples daily on a standard laptop. However, this benchmarking process only encompasses preliminary analysis and does not incorporate the application of machine learning for correlation. When utilizing machine learning, optimizations become necessary. Employing machine learning with optimizations made it feasible to process 267 samples in less than three minutes. This level of performance enables the fulfillment of the $H0$ hypothesis.

7.1 Future Work

Future work in this research involves designing new approaches to representing malware genes and exploring alternative machine learning methods. A direct continuation of this study would involve efforts to reduce the dimensionality of input vector features. While proposed optimizations may contribute to this goal by selecting representative genes, it will also be essential to systematically and algorithmically reduce the number of malware features in future research endeavors.

The primary focus of this research was to explore the potential of supervised machine learning. Future endeavors will involve exploring novel applications of artificial intelligence algorithms such as clustering and deep learning. Further research aims to achieve an accuracy rate exceeding 90% overall and 70% for similar malware families. The insights and code developed in subsequent research phases are intended to enrich the MalTraits project further, facilitating the practical application of research findings.

8. Conclusion

Genetic malware analysis is an emerging field that greatly assists cyber teams in malware correlation and classification. This research analyzes the methods of how and when to use the field of Genetic Malware Analysis. The research findings confirm the primary research question, demonstrating the feasibility of applying machine learning to a unique representation of malware genes. Machine learning was assessed using both categorical and numerical data representations. This study encompasses the introduction of entirely novel approaches to malware genes, ranging from feature engineering to the assessment of machine learning experiments.

The analytical part of the research describes the strengths and weaknesses inherent in malware genes.Among the notable weaknesses of Genetic Malware Analysis is its reliance on static analysis, which generates malware genes through disassembly. While static analysis offers rapid processing of malware samples, it falls short in effectively analyzing packed samples.

The research's significance lies in its practical application of advanced technology, such as distributed workers, diverse databases, and machine learning. This solution involved programming several thousand lines of code, analyzing various malware samples, and evaluating multiple experiments. The results were promising, with machine learning achieving an impressive 80% accuracy over malware genes.

Genetic Malware Analysis is a methodology that enables organizations to analyze large quantities of malware within their infrastructure, bridging the fields of data and malware analysis. This integration has notably enhanced the efficiency of malware analysis through classification and correlation processes. Moreover, organizations can leverage malware genes for tasks like threat actor attribution and developing signature detection rules.

References

[1] Doaa Wael and Marianne A. Azer. “Malware Incident Handling and Analysis Workflow”. In: *2018 14th International Computer Engineering Conference (ICENCO)*. 2018, pp. 242–248. DOI: 10.1109/ICENCO.2018.8636144.

[2] The Independent IT-Security Institute. *Malware statistics amp; trends report: AV-TEST*. URL: https://www.av-test.org/en/statistics/malware/.

[3] Martin Mihalovič. *Malware Analysis of Cyber Attacks*. May 2022. URL: https://opac.crzp.sk/?fn=detailBiblioFormChildI22150&amp;sid=4F55101800366CEF55F6C3DB109A&amp;seo=CRZP-detail-kniha.

[4] A.D. George. “An overview of RISC vs. CISC”. In: *[1990] Proceedings. The Twenty-Second Southeastern Symposium on System Theory*. 1990, pp. 436–438. DOI: 10.1109/SSST.1990.138185.

[5] Tran Nghi Phu et al. “A Novel Framework to Classify Malware in MIPS Architecture-Based IoT Devices”. In: *Sec. and Commun. Netw.* 2019 (Jan. 2019). ISSN: 1939-0114. DOI: 10.1155/2019/4073940. URL: https://doi.org/10.1155/2019/4073940.

[6] Patrick Wardle. Oct. 2021. URL: https://vblocalhost.com/uploads/VB2021-Wardle.pdf.

[7] Mar. 2024. URL: https://www.virustotal.com/gui/stats.

[8] Anandharaju Durai Raju et al. “A Survey on Cross-Architectural IoT Malware Threat Hunting”. In: *IEEE Access* 9 (2021), pp. 91686–91709. DOI: 10.1109/ACCESS.2021.3091427.

[9] Francesca Arcelli Fontana, Davide Franzosi, and Claudia Raibulet. “.NET reverse engineering with MARPLE”. In: Aug. 2010, pp. 227–231. DOI: 10.1109/ICSEA.2010.41.

[10] Valentina Bellini et al. “Understanding basic principles of artificial intelligence: a practical guide for intensivists”. In: *Acta Bio Medica: Atenei Parmensis* 93.5 (2022).

[11] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O’Reilly Media, Inc., 2019. ISBN: 1492032646.

[12] Mehadi Hassen and Philip K. Chan. "Scalable Function Call Graph-Based Malware Classification". In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. CODASPY '17. Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pp. 239–248. ISBN: 9781450345231. DOI: 10.1145/3029806.3029824. URL: https://doi.org/10.1145/3029806.3029824.

[13] Jianwei Ding et al. "MGeT: Malware Gene-Based Malware Dynamic Analyses". In: *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*. ICCSP '17. Wuhan, China: Association for Computing Machinery, 2017, pp. 96–101. ISBN: 9781450348676. DOI: 10.1145/3058060.3058065. URL: https://doi-org.ezproxy.utlib.ut.ee/10.1145/3058060.3058065.

[14] Christian Collberg et al. "Slinky: Static Linking Reloaded". In: (Mar. 2004).

[15] Mamoru Mimura. "Impact of benign sample size on binary classification accuracy". In: *Expert Systems with Applications* 211 (2023), p. 118630. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.118630. URL: https://www.sciencedirect.com/science/article/pii/S0957417422016773.

[16] Roy Halevi. *What is genetic malware analysis?* Jan. 2019. URL: https://intezer.com/blog/malware-analysis/defining-genetic-malware-analysis/.

[17] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. 1st. USA: No Starch Press, 2012. ISBN: 1593272901.

[18] Erik Guttman and Nevil Brownlee. *RFC FT-IETF-grip-framework-IRT: Expectations for computer security incident response*. June 1998. URL: https://datatracker.ietf.org/doc/html/rfc2350#section-3.5.1.

[19] Software Engineering Institute. *CSIRT Services - Carnegie Mellon University*. Jan. 2017. URL: https://resources.sei.cmu.edu/asset_files/WhitePaper/2022_019_001_884490.pdf.

[20] Paul Cichonski et al. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-61r2.pdf.

[21] Anjali Udasi. *The Incident Response Lifecycle: Strategies for Effective Incident Management*. July 2023. URL: https://www.zenduty.com/blog/incident-response-lifecycle/.

[22] Fraunhofer FKIE. *Malpedia General Statistics*. URL: https://malpedia.caad.fkie.fraunhofer.de/stats/general.

[23] Abdulaziz Ali Alkandari, Imad Fakhri Al-Shaikhli, and Mohammad A. Alahmad. “Cryptographic Hash Function: A High Level View”. In: *2013 International Conference on Informatics and Creative Multimedia*. 2013, pp. 128–134. DOI: 10.1109/ICICM.2013.29.

[24] Darshana Upadhyay et al. “Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications”. In: *IEEE Access* 10 (2022), pp. 112472–112486. DOI: 10.1109/ACCESS.2022.3215778.

[25] Kamran Saifullah. *The Crown Jewels and the Pyramid of Pain*. July 2021. URL: https://www.gispp.org/2021/07/25/the-crown-jewels-and-the-pyramid-of-pain/.

[26] Ivan Firdausi et al. “Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection”. In: *Advances in Computing, Control, and Telecommunication Technologies, International Conference on* 0 (Dec. 2010), pp. 201–203. DOI: 10.1109/ACT.2010.33.

[27] Alexey Bukhteyev. *Invisible sandbox evasion*. Mar. 2023. URL: https://research.checkpoint.com/2022/invisible-cuckoo-cape-sandbox-evasion/.

[28] Ondřej Maňhal. “Evading CAPE Sandbox Detection”. PhD thesis. CVUT DSpace, 2022, pp. 16–18.

[29] Dr. Mafaz. “Generic packing detection using several complexity analysis for accurate malware detection”. In: *International Journal of Advanced Computer Science and Applications* 5.1 (2014). DOI: 10.14569/ijacsa.2014.050102.

[30] Rodrigo Branco, Gabriel Barbosa, and Pedro Neto. *Scientific but not academical overview of malware anti-debugging, anti ...* Dec. 2012. URL: https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf.

[31] Ricardo J. Rodríguez, Juan Antonio Artal, and Jose Merseguer. “Performance Evaluation of Dynamic Binary Instrumentation Frameworks”. In: *IEEE Latin America Transactions* 12.8 (2014), pp. 1572–1580. DOI: 10.1109/TLA.2014.7014530.

[32] c3rb3ru5d3d53c. *C3RB3RU5D3D53C/binlex: A binary genetic traits lexer framework*. URL: https://github.com/c3rb3ru5d3d53c/binlex.

[33] Ranjith G Hegde. “Low Latency Message Brokers”. In: 2020. URL: https://api.semanticscholar.org/CorpusID:235816029.

[34] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (Jan. 2012).

[35] Kilian Q. Weinberger and Lawrence K. Saul. “Distance Metric Learning for Large Margin Nearest Neighbor Classification”. In: *J. Mach. Learn. Res.* 10 (June 2009), pp. 207–244. ISSN: 1532-4435.

[36] Zhongheng Zhang. “Introduction to machine learning: K-nearest neighbors”. In: *Annals of Translational Medicine* 4 (June 2016), pp. 218–218. DOI: 10.21037/atm.2016.03.37.

[37] S. Zhang, J. Li, and Y. Li. “Reachable Distance Function for KNN Classification”. In: *IEEE Transactions on Knowledge amp; Data Engineering* 35.07 (July 2023), pp. 7382–7396. ISSN: 1558-2191. DOI: 10.1109/TKDE.2022.3185149.

[38] M. Raj Shekhar Rao, Deepanshu Yadav, and V. Anbarasu. “An Improvised Machine Learning Model KNN for Malware Detection and Classification”. In: *2023 International Conference on Computer Communication and Informatics (ICCCI)*. 2023, pp. 1–4. DOI: 10.1109/ICCCI56745.2023.10128189.

[39] Julio López and Sebastián Maldonado. “Redefining nearest neighbor classification in high-dimensional settings”. In: *Pattern Recognition Letters* 110 (2018), pp. 36–43. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2018.03.023. URL: https://www.sciencedirect.com/science/article/pii/S0167865518301028.

[40] Feiping Nie et al. “An Effective and Efficient Algorithm for K-Means Clustering With New Formulation”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.4 (2023), pp. 3433–3443. DOI: 10.1109/TKDE.2022.3155450.

[41] Hussein Al Khansa, Fadi Yamout, and Fatima Salam. “Deriving Centroids for K-means Algorithm”. In: *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2018, pp. 266–269. DOI: 10.1109/CSCI46756.2018.00058.

[42] URL: https://scikit-learn.org/stable/modules/clustering.html.

[43] Shruti Sehgal et al. “Data analysis using principal component analysis”. In: *2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom)*. 2014, pp. 45–48. DOI: 10.1109/MedCom.2014.7005973.

[44] Rima Asmar Awad and Kirk D. Sayre. “Automatic clustering of malware variants”. In: *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. 2016, pp. 298–303. DOI: 10.1109/ISI.2016.7745494.

[45] Yipin Zhang et al. “Exploring Function Call Graph Vectorization and File Statistical Features in Malicious PE File Classification”. In: *IEEE Access* 8 (2020), pp. 44652–44660. DOI: 10.1109/ACCESS.2020.2978335.

[46] Jianwen Fu et al. “Malware Visualization for Fine-Grained Classification”. In: *IEEE Access* 6 (2018), pp. 14510–14523. DOI: 10.1109/ACCESS.2018.2805301.

[47] Putu Sukma Dharmalaksana et al. “Improved Malware Detection Results using Visualization-Based Detection Techniques ant Convolutional Neural Network”. In: *2022 IEEE 8th International Conference on Computing, Engineering and Design (ICCED)*. 2022, pp. 1–5. DOI: 10.1109/ICCED56140.2022.10010439.

[48] Guosong Sun and Quan Qian. “Deep Learning and Visualization for Identifying Malware Families”. In: *IEEE Transactions on Dependable and Secure Computing* 18.1 (2021), pp. 283–295. DOI: 10.1109/TDSC.2018.2884928.

[49] Richard Zak, Edward Raff, and Charles Nicholas. “What can N-grams learn for malware detection?” In: *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*. 2017, pp. 109–118. DOI: 10.1109/MALWARE.2017.8323963.

[50] N. Visweswaran et al. “Automated PE32 Threat Classification using Import Table and Deep Neural Networks”. In: *2019 IEEE International Conference on Clean Energy and Energy Efficient Electronics Circuit for Sustainable Development (INCCES)*. 2019, pp. 1–5. DOI: 10.1109/INCCES47820.2019.9167732.

[51] Michal Kida and Oluwafemi Olukoya. “Nation-State Threat Actor Attribution Using Fuzzy Hashing”. In: *IEEE Access* 11 (2023), pp. 1148–1165. DOI: 10.1109/ACCESS.2022.3233403.

[52] Jason Upchurch and Xiaobo Zhou. *Malware provenance: code reuse detection in malicious software at scale*. 2016. DOI: 10.1109/MALWARE.2016.7888735.

[53] Muhammad Ijaz, Muhammad Hanif Durad, and Maliha Ismail. “Static and Dynamic Malware Analysis Using Machine Learning”. In: *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. 2019, pp. 687–691. DOI: 10.1109/IBCAST.2019.8667136.

[54] Stavros D. Nikolopoulos and Iosif Polenakis. “A graph-based model for malware detection and classification using system-call groups”. In: *Journal of Computer Virology and Hacking Techniques* 13.1 (2016), pp. 29–46. DOI: 10.1007/s11416-016-0267-1.

[55] Joren Vrancken. *Detecting Capabilities in Malware Binaries by Searching for Function Calls*. URL: https://www.ru.nl/publish/pages/769526/joren_vrancken.pdf.

[56] O’Meara and Kyle CERT Insider Threat Center. *API Hashing Tool, imagine that*. Mar. 2019. URL: https://insights.sei.cmu.edu/blog/api-hashing-tool-imagine-that/.

[57] Coen Boot. *Applying supervised learning on malware authorship attribution - ru.nl.* URL: https://www.ru.nl/publish/pages/769526/b_coen_boot.pdf.

[58] Omid Mirzaei et al. "SCRUTINIZER: Detecting Code Reuse in Malware via Decompilation and Machine Learning". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Leyla Bilge et al. Cham: Springer International Publishing, 2021, pp. 130–150. ISBN: 978-3-030-80825-9.

[59] Yong Fang et al. "Semi-Supervised Malware Clustering Based on the Weight of Bytecode and API". In: *IEEE Access* 8 (2020), pp. 2313–2326. DOI: 10.1109/ACCESS.2019.2962198.

[60] Yihang Chen et al. "A Gene-Inspired Malware Detection Approach". In: *Journal of Physics: Conference Series* 1168 (Feb. 2019), p. 062004. DOI: 10.1088/1742-6596/1168/6/062004.

[61] Ishai Rosenberg, Guillaume Sicard, and Eli David. "End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware". In: *Entropy* 20 (May 2018), p. 390. DOI: 10.3390/e20050390.

[62] Ilay Cordonsky et al. "DeepOrigin: End-To-End Deep Learning For Detection Of New Malware Families". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–7. DOI: 10.1109/IJCNN.2018.8489667.

[63] Martin Mihalovic, Matej Las, and Michal Minar. URL: https://drive.google.com/drive/folders/1OFqLiP9GBt3cuwJB0IXJ6b87-XBwcuvu?usp=sharing.

[64] Masafumi Oyamada. "Extracting Feature Engineering Knowledge from Data Science Notebooks". In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 6172–6173. DOI: 10.1109/BigData47090.2019.9006522.

[65] Debajyoti Bera, Rameshwar Pratap, and Bhisham Dev Verma. "Dimensionality Reduction for Categorical Data". In: *IEEE Trans. on Knowl. and Data Eng.* 35.4 (Apr. 2023), pp. 3658–3671. ISSN: 1041-4347. DOI: 10.1109/TKDE.2021.3132373. URL: https://doi.org/10.1109/TKDE.2021.3132373.

[66] Ranjani S. Anitha et al. "Categorical Data Clustering using Cosine based similarity for Enhancing the Accuracy of Squeezer Algorithm". In: *International Journal of Computer Applications* 45 (2012), pp. 41–45. URL: https://api.semanticscholar.org/CorpusID:1464229.

[67] Luciano da Fontoura Costa. "Further Generalizations of the Jaccard Index". In: *ArXiv* abs/2110.09619 (2021). URL: https://api.semanticscholar.org/CorpusID:239024336.

[68] Karl-Bridge-Microsoft. *PE format - win32 apps*. Feb. 2024. URL: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[69] Trivikram Muralidharan et al. “File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements”. In: *ACM Comput. Surv.* 55.5 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3530810. URL: https://doi.org/10.1145/3530810.

[70] Fahad Alswaina and Khaled Elleithy. “Android Malware Family Classification and Analysis: Current Status and Future Directions”. In: *Electronics* 9.6 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9060942. URL: https://www.mdpi.com/2079-9292/9/6/942.

[71] Médéric Hurier et al. “Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware”. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017, pp. 425–435. DOI: 10.1109/MSR.2017.57.

[72] Europol. *Qakbot botnet infrastructure shattered after international operation*. URL: https://www.europol.europa.eu/media-press/newsroom/news/qakbot-botnet-infrastructure-shattered-after-international-operation.

[73] J. Hamrock and R. Lyda. “Using Entropy Analysis to Find Encrypted and Packed Malware”. In: *IEEE Security amp; Privacy* 5.02 (Mar. 2007), pp. 40–45. ISSN: 1558-4046. DOI: 10.1109/MSP.2007.48.

[74] Matt Green. *Automating qakbot decode at scale*. Apr. 2023. URL: https://docs.velociraptor.app/blog/2023/2023-04-05-qakbot/.

[75] Karl-Bridge-Microsoft. *VirtualAlloc function (memoryapi.h) - win32 apps*. Feb. 2024. URL: https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc.

[76] Najla Etaher, George R.S. Weir, and Mamoun Alazab. “From ZeuS to Zitmo: Trends in Banking Malware”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 1386–1391. DOI: 10.1109/Trustcom.2015.535.

[77] Dennis Schwarz. *Zeusmuseum*. URL: https://zeusmuseum.com/.

[78] Kurt Baker. *What is zeus trojan malware? - crowdstrike*. Jan. 2024. URL: https://www.crowdstrike.com/cybersecurity-101/malware/trojan-zeus-malware/.

[79] Jerome Cruz. *Dreambot 2017 vs. ISFB 2013*. Mar. 2018. URL: https://www.fortinet.com/blog/threat-research/dreambot-2017-vs-isfb-2013.

[80] Abdulrahman Alkandari and Soha Jaber Aljaber. "Principle Component Analysis algorithm (PCA) for image recognition". In: *2015 Second International Conference on Computing Technology and Information Management (ICCTIM)*. 2015, pp. 76–80. DOI: 10.1109/ICCTIM.2015.7224596.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Experiment QakBot Malware Analysis

In section 6.1, the Qakbot malware was discussed in more detail. This analysis aims to verify whether QakBot uses a custom packer and whether the sample is packed at all. This qualitative research results in a better understanding of custom packers in general and how they can be detected using malware genes. The first part of the malware analysis consists of a Basic static analysis.

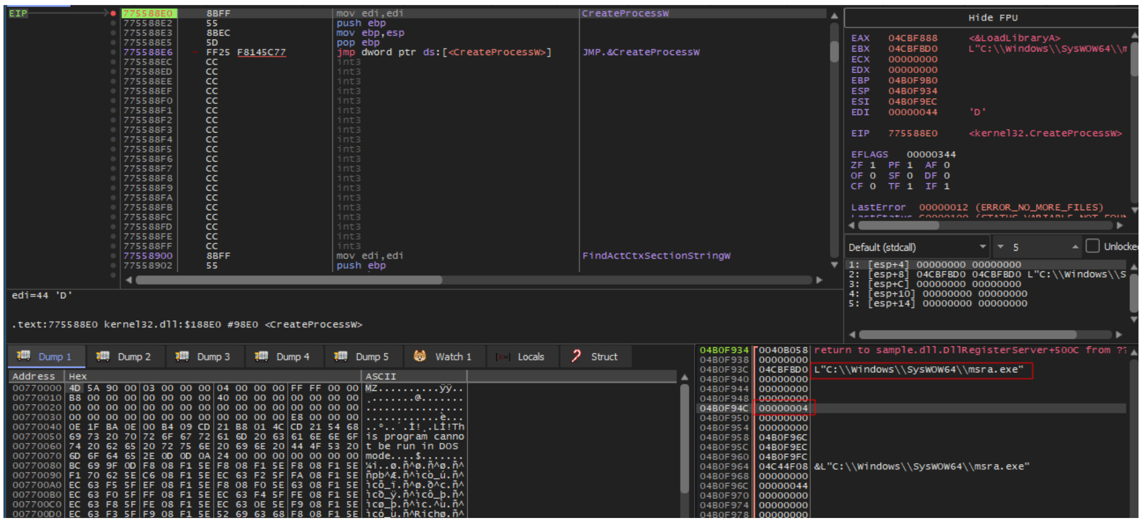
Attribute	**Value**
Analysis Name	QakBot ExploitReversing
Analysis ID/Ticket	PAT24MAR08
Analysis Result	Malicious
TLP	White
TTP	
	- T1083: File and Directory Discovery
	- T1057: Process Discovery
Taxonomies	Malware-type="Loader"
Type of Analysis	
	- Code Analysis
File Type	PE64
PDB Info	-
SHA256	73e4969db4253f9aeb2cbc7462376fb7e26cc4bb5bd23b82e2af0eaaf5ae66a8

Table 6. Qakbot Basic Static Analysis

Based on the small number of meaningful strings, as well as the API function, it was suspected that the sample is packed. This suspicion is also indicated by a higher entropy. Therefore, the sample was debugged in the x64dbg tool, where the unpacking strategy was chosen using breakpoints on several selected API calls.

- VirtualAlloc
- CreateProcessW
- WriteProcessMemory
- ZwUnmapViewOfSection

After a thorough analysis, it was found that the sample wants to be unpacked by injecting the unpacked code into the process named "msra.exe". The unpacking process was revealed after the 6th hit of the VirtualAlloc breakpoint.



```
EIP  775588E0   8BFF           mov edi,edi                         CreateProcessW
     775588E2   55             push ebp
     775588E3   8BEC           mov ebp,esp
     775588E5   5D             pop ebp
     775588E6   FF25 F8145C77  jmp dword ptr ds:[<CreateProcessW>]  JMP.&CreateProcessW
     775588EC   CC             int3
     775588ED   CC             int3
     775588EE   CC             int3
     775588EF   CC             int3
     775588F0   CC             int3
     775588F1   CC             int3
     775588F2   CC             int3
     775588F3   CC             int3
     775588F4   CC             int3
     775588F5   CC             int3
     775588F6   CC             int3
     775588F7   CC             int3
     775588F8   CC             int3
     775588F9   CC             int3
     775588FA   CC             int3
     775588FB   CC             int3
     775588FC   CC             int3
     775588FD   CC             int3
     775588FE   CC             int3
     775588FF   CC             int3
     77558900   8BFF           mov edi,edi                         FindActCtxSectionStringW
     77558902   55             push ebp

edi=44 'D'

.text:775588E0 kernel32.dll:$188E0 #98E0 <CreateProcessW>
```

```
Hide FPU

EAX   04CBF888    <&LoadLibraryA>
EBX   04CBFBD0    L"C:\\Windows\\SysWOW64\\m
ECX   00000000
EDX   00000000
EBP   04B0F9B0
ESP   04B0F934
ESI   04B0F9EC
EDI   00000044    'D'

EIP   775588E0    <kernel32.CreateProcessW>

EFLAGS   00000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000012 (ERROR_NO_MORE_FILES)

Default (stdcall)   5   Unlocke
1: [esp+4] 00000000 00000000
2: [esp+8] 04CBFBD0 04CBFBD0 L"C:\\Windows\\S
3: [esp+C] 00000000 00000000
4: [esp+10] 00000000 00000000
5: [esp+14] 00000000 00000000
```

```
Dump 1   Dump 2   Dump 3   Dump 4   Dump 5   Watch 1   Locals   Struct

Address   Hex                                               ASCII
00770000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00770010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00770020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00770030  00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00  ............è...
00770040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00770050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00770060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00770070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00770080  BC 69 9F 0D F8 08 F1 5E F8 08 F1 5E F8 08 F1 5E  ¼i..ø.ñ^ø.ñ^ø.ñ^
00770090  F1 70 62 5E C6 08 F1 5E EC 63 F2 5F FA 08 F1 5E  ñpb^Æ.ñ^ìcò_ú.ñ^
007700A0  EC 63 F5 5F EF 08 F1 5E F8 08 F0 5E 63 08 F1 5E  ìcõ_ï.ñ^ø.ð^c.ñ^
007700B0  EC 63 F0 5F FF 08 F1 5E EC 63 F4 5F FE 08 F1 5E  ìcð_ÿ.ñ^ìcô_þ.ñ^
007700C0  EC 63 F8 5F FE 08 F1 5E EC 63 0E 5E F9 08 F1 5E  ìcø_þ.ñ^ìc.^ù.ñ^
007700D0  EC 63 F3 5F F9 08 F1 5E 52 69 63 68 F8 08 F1 5E  ìcó_ù.ñ^Richø.ñ^
```

```
04B0F934  0040B058  return to sample.dll.DllRegisterServer+500C from ??
04B0F938  00000000
04B0F93C  04CBFBD0  L"C:\\Windows\\SysWOW64\\msra.exe"
04B0F940  00000000
04B0F944  00000000
04B0F948  00000000
04B0F94C  00000004
04B0F950  00000000
04B0F954  00000000
04B0F958  04B0F96C
04B0F95C  04B0F9EC
04B0F960  04B0F9FC
04B0F964  04C44F08  &L"C:\\Windows\\SysWOW64\\msra.exe"
04B0F968  00000000
04B0F96C  00000044
04B0F970  00000000
04B0F974  00000000
04B0F978  00000000
```

Figure 19. Debugging and unpacking QakBot malware

After unpacking the sample, the code was analyzed statically using the IDA Pro tool. Among other things, a decryption routine was found in the code, which decrypted the string table of the malware. Python script was written by us to decipher the strings

```python
import binascii
import pefile


def calc(num1, num2) -> int:
    return int(num1, 16) - int(num2, 16)


def getPE(file_pathname: str):
    return pefile.PE(file_pathname)


def extractSection(pe, section_name: str):
    for section in pe.sections:
        name = section.Name.decode()

        if section_name in name:
            return section.VirtualAddress, section.get_data(
                section.VirtualAddress, section.SizeOfRawData
            )
    return None


```

```python
def getImageBase(pe):
    return pe.OPTIONAL_HEADER.ImageBase


def decrypt(enc_data, key, offset=0):
    sz_key = len(key)
    dec_data = []
    dec_string = ""

    for i, byte in enumerate(enc_data[offset:], start=offset):
        dec_char = byte ^ key[i % sz_key]
        if dec_char == 0:
            if offset != 0:
                return dec_string
            dec_data.append(dec_string)
            dec_string = ""
        else:
            dec_string += chr(dec_char)

    return dec_data


def getEndOfByteBlob(section_data, blob_offset):
    if b"\x00\x00" in section_data[blob_offset:]:
        return section_data[blob_offset:].index(b"\x00\x00")
    return -1


def stringDecrypt(string_table_addr, key_addr):
    pe = getPE("./evidence/va5_rundll32_04FE0000.bin")
    base_addr = getImageBase(pe)

    data_section_offset, data_section = extractSection(pe,
        ".data")
    data_section_addr = base_addr + data_section_offset

    string_table_offset = string_table_addr -
        data_section_addr
    key_offset = key_addr - data_section_addr

    string_table_end_offset = getEndOfByteBlob(data_section,
        string_table_offset)
    key_end_offset = getEndOfByteBlob(data_section,
        key_offset)

    if string_table_end_offset == -1 or key_offset == -1:
        return

    enc_string_table = data_section[
        string_table_offset : string_table_offset +
            string_table_end_offset
    ]
    xor_key = data_section[key_offset : key_offset +
        key_end_offset]

    dec_string_table = decrypt(enc_string_table, xor_key)
    return enc_string_table, xor_key, dec_string_table


def main():
    enc_string_table, xor_key, dec_string_table =
        stringDecrypt(0x1001D5A8, 0x1001E3F8)

    iat = [1486, 2912, 531, 1533, 1645, 764, 3648, 3042]
    for library in iat:
        library_string = decrypt(enc_string_table, xor_key,
            library)
        print(f"Decrypted library {library_string} on offset
            {library}")

    print(f"Decrypted string table: {dec_string_table}")


main()
```

After decryption, several library names were revealed, which were then used for the API Hashing technique. A more detailed static analysis was not performed, because the goal was to verify whether the sample is really packed. This suspicion was confirmed by dynamic analysis of the code using a debugger.