Malware Evasion Techniques

By
Ryan Smallcalder

Project unit: PJE40
Supervisor: Tobi Fajana

May 2023

Abstract

Malware's capabilities of devastating and disrupting modern security systems are everlasting, with modern iterations proving more sophisticated and adaptive, commonly thwarting heuristic antivirus software; with these prevention measures trailing behind and innovative and high-level malware heavily impacting institutions, it is evident that a solution to identify these techniques is synthesised. This dissertation proposed a methodological framework for automating malware collection, explicitly focusing on evasion, achieving this through VirusTotal's API and a dataset retrieved from the University of New Brunswick titled "CIC-MalMem-2022". The applied approach resulted in a scalable, replicable, and highly repeatable solution which will prove insightful by providing security researchers and professionals with a taxonomy of present-day evasion techniques, illustrating popular and obscure trends amongst malware, and demonstrating the effectiveness and practicality of evasion techniques.

Keywords: Malware Detection, Evasion, Obfuscation, Anti-Sandboxing, Anti-Analyst

Acknowledgements

First, I would like to express my gratitude to my supervisor, Tobi Fajana, for providing guidance in choosing the project, giving up his time for weekly meetings, and giving direction and advice to my final year project and to other modules.

Contents

Chapter 1: Introduction

1.1 Malware Introduction

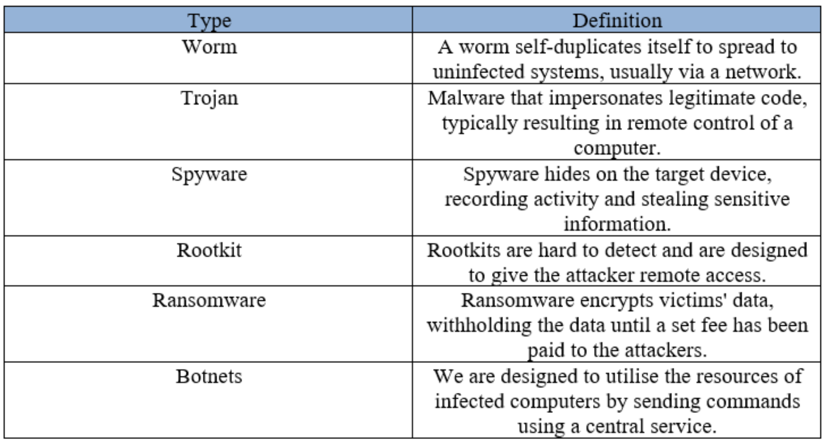
Malware’s exponential growth from primitive viruses to more advanced, sophisticated, and targeted attacks is a concern for modern security. Malware authors are typically advantageous in outsmarting current detection algorithms (Marpaung et al., 2012). The internet facilitates this growth, consequently offering a desirable environment for rapid malicious transmissions (Gao et al., 2015).

Malware’s characterised as software designed to alter a device’s behaviour, specifically in a way that allows unauthorised access or interrupts or damages a computer system (Kramer & Bradfield, 2009). Whilst this is a generalised definition for malware, there are various subsets of computer viruses, split into malware types that exhibit different behaviours, resulting in different outcomes depending on the author’s intentions.

Malware’s behavioural characteristics depend on its type and family; therefore, successful comprehension of malware types is vital to removing and identifying unclassified malware. Due to the extensive amount of malware types and families, it is essential to implement a scope of malware that will be discussed and analysed; Figure 1 illustrates this, displaying common malware types alongside a definition.

Figure 1.0

Common Malware Types



Type	Definition
Worm	A worm self-duplicates itself to spread to uninfected systems, usually via a network.
Trojan	Malware that impersonates legitimate code, typically resulting in remote control of a computer.
Spyware	Spyware hides on the target device, recording activity and stealing sensitive information.
Rootkit	Rootkits are hard to detect and are designed to give the attacker remote access.
Ransomware	Ransomware encrypts victims' data, withholding the data until a set fee has been paid to the attackers.
Botnets	We are designed to utilise the resources of infected computers by sending commands using a central service.

Precisely, 15.37% of Kaspersky users experienced at least one malware attack within 2022, with 43.68% of all infections originating from malicious URLs (Kaspersky, 2022), exhibiting the lack of cybersecurity awareness amongst users and ease of infection from more sophisticated malware iterations. Paired alongside similar yearly statistics, such as the discovery of 41 families and the identification of more than 23,807 new ransomware variants, emphasises the need for a combination of up-to-date evasion techniques to aid in the analysis of malware by cutting down examination times.

This highlights the need for a study and analysis of evasive techniques, with comprehensive surveys including (You & Yim, 2010; Marpaung et al., 2012; Schrittwieser et al., 2017; Galloroa et al. 2021) demonstrating a collection of evasion techniques, indicating a small gap within the literature. This project will differentiate by aiming to provide a framework for automating malware collection and analysis, aiding in analysis and triage by cutting down mass analysis times.

1.2 Project Aims & Objectives

This project aims to develop a framework for large-scale automated malware collection and analysis, specifically focusing on evasion, to evaluate evasion techniques utilised by malware authors critically. Specifically, deconstructing the project into smaller subsections outlined by the following objectives:

- Critically review existing literature, highlighting any research gaps.
- Develop and test an automated tool for collecting malware data.
- Conduct an analysis of evasion tactics on a dataset.
- Illustrate obscure and sophisticated evasion techniques.
- Showcase the evolution and practicality of these techniques in the wild.
- Evaluate the viability of older evasion techniques.

Due to the extensive amount of malware types and families, it is essential to implement a scope of malware that will be discussed and analysed. Therefore, the applied scope will be Windows-based malware, excluding malware that infects macOS, Android and IOS platforms.

1.3 Research Questions

To further understand the project context, developing a research question allows for increased clarity and determining what exactly the project would achieve. The primary goal is to answer the question: How has Malware Evasion evolved and changed over time? This research questions provided depth and structure to the report and were synthesised after a preliminary scan of literature; there was minimal research on the discussion and analysis of evasion techniques, particularly a timeline of techniques encapsulating their advantages, disadvantages, and usability against modern detection systems. The research questions were chosen based on novelty, feasibility, and relevance; this will allow for insight into the original outlook on malware evasion, presenting a viable approach to the analysis of malware evasion.

1.4 Project Constraints

Main project constraints include time; the project start date was September, and will commence in early May. This constraint will be managed by creating a Gantt chart and using google calendar to book meetings and set strict timeframes. Whilst this is a simplistic method, it provides an effective strategy for managing time whilst also being cost-effective and easy to set up. In addition, the quality of work should remain at a high standard. Implementing a scope will ensure this whilst avoiding wasting time and guaranteeing that the project remained relevant to the topic. Therefore, the scope will include malware evasion techniques concerning computers utilising Windows operating systems. A log of risks shown in Figure 1.1 also includes more constraints appropriate to the project with proper mitigation strategies in place.

Figure 1.1

Log of Risks

Description	Impact	Mitigation
Hardware failure	Low – Slight delay in progress.	Frequent Backups to documents Use of University upon an incident
Malware Attack	Medium – Could Result in corrupted data or encrypted data	Frequent backups to all data used Frequent antivirus scans after each analysis. Use of a virtual machine when analysing malware
Lack of Knowledge In Subject Area	Medium - Delayed time scale	Attend lectures Frequent library visits Frequent meetings/emails to fill gaps in knowledge.
Internet connectivity issues	Low - Delays to project time	Work in the Library/Lab facilities.
Supervisor absence	Low - Delays to project. It would have a higher impact depending on the time of the year	Weekly meetings to stay on top of the project, with an opportunity to exceed the project timeline.

1.5 Project Deliverables

Expected project outcomes are a dataset that will be found or created based on prevalent malware within the last ten years, the number of which will depend on the time scale. The second deliverable will be the report, including a dataset analysis. The report will contain the following:

- Chapter 1 - Introduction
- Chapter 2 - Literature Review
- Chapter 3 - Methodology
- Chapter 4 - Design
- Chapter 5 - Implementation
- Chapter 6 - Analysis
- Chapter 7 – Conclusion

1.6 Project Approach

Categorising the project into subsections, each given a deadline for completion managed by Google Planner due to its familiarity, ease of use and ability to sync with lectures to avoid missing other deadlines for other assignments. The execution of critical path analysis to systematically break down tasks into essential tasks will ensure the project recognises critical steps, illustrated in Figure 1.2.

Figure 1.2

Critical Path Analysis Timeline

Activity	Duration	Preceding Activities
A – Introduction	2 Weeks	-
B – Literature Review	12 Weeks	-
C – Methodology	2 Week	B
D – Data Collection	3 Weeks	C
E – Analysis	8 Weeks	D
F - Conclusion	2 Week	A, B, C, D, E

1.7 Legal and Ethical Issues

Legal and ethical issues arise when dealing with any dataset; the dataset may contain malware that accesses files from the target computer, and analysing this from a sandboxing website may involve the collection of personal data, which GDPR regulates, and therefore must comply with this and ensure the confidentiality of any sensitive data. As the project will not directly analyse malware, the implications of research will not be as severe; there will be no infringement on any intellectual property due to the absence of reverse engineering and avoiding using live malware samples.

1.8 Conclusion

This chapter explored the initiation of the project, ensuring that research questions, aims, and objects are established to enforce an achievable scope, demonstrating the complexity of the project whilst ensuring that proper constraints, approaches, and issues with legality are considered to guarantee a streamlined project schedule with the introduction of critical path analysis

Chapter 2: Literature Review

2.1 Introduction

Modern malware Iterations depend on sophisticated evasion techniques to thwart antivirus tools and analysis (Sharif et al., 2008). The advancement in these techniques highlights outdated detection mechanisms, precisely the weakness within signature detection and its inability to prevent attacks originating from unknown signatures (O’Kane et al., 2011). Therefore, this chapter will review current evasion techniques, providing a definition, explanation and possible implementation methods for a given strategy. Upon comprehension of the topic area, the synthesis and evaluation of the progression of evasion techniques allowed for recognising more innovative evasion strategies and if older techniques remain viable.

2.2 Search Strategy

A preliminary plan for the search strategy involved the identification of keywords, commencing with “evasion techniques’, after reading relevant articles and several subcategories of evasion techniques, allowing for the identification of additional relevant literature, illustrated in Figure 1.3.

Figure 1.3

Search Strategy

Search	Query	Results Retrieved
1	Evasion techniques of malware	22,200 Results
2	Malware evasion OR anti-analysis OR encryption OR obfuscation -“android”	73,000 Results
3	Malware environmental awareness	29,400 Results
4	Malware Anti-Sandbox	256 Results
5	Malware polymorphic OR metamorphic OR Oligomorphic	3,220 Results

The targeting of the most recent papers allowed for an up-to-date definition of evasion techniques. However, avoiding the dismissal of older papers as the evasion techniques are mainly adapted versions and are defined similarly. Therefore, some older research remains as it provides a detailed background into early evasion.

The primary search engine utilised was Google scholar; due to the topic's extensive size, it was essential to use Google's advanced search operands to remove sub-genres that do not contribute to the project's aims. Google Scholar was used as a primary research strategy due to its extensive collage of literature, encapsulating other online libraries such as IEEE Xplore, Elsevier, and ACM Digital Library, thus ensuring a wide range of viable literature.

All searches within the chosen library resulted in many results retrieved; regardless, several of them would not be helpful to the project, forefronting an exclusion strategy. The exclusion strategy focused on sorting the articles from relevancy based on title, then reading the abstract alongside the conclusion. If the article were not excluded, it would be thoroughly read and then excluded/included based on relevance. Ensuring only studies that contributed to the project's aims would be included. In addition, literature initially deemed relevant to the project could also be excluded by its quality, usually determined by the number of quality references, the inclusion of related works and effective formatting. The collection of literature typically concluded when information became redundant.

Comprehension of notable literature within the subject area led to obtaining their primary references to find additional references, repeating the process until the literature became redundant. These additional references ensured that the quality of references remained at a high standard whilst also ensuring the inclusion of literature about sub-genres of evasion techniques that would have otherwise been overlooked.

2.3 Malware Defined

Malware is software with the intent designed to create havoc, damage, or disrupt systems and resources connected to them (Tahir, 2018). Whilst malware removal can be a novel task, evasive malware proves a difficult task for the industry, continuing to evolve into undetectable variants contributing to day-one attacks (Alsmadi & Alqudah, 2021).

Kirat & Vigna (2015) defines evasive malware as malware whose behaviour deviates from an executable environment, simply as malware that thwarts detection.

2.4 Detection Strategies

Understanding how malware is detected through various strategies will provide a foundation for comprehending the theory behind malware evasion techniques; these strategies are dissected into subsections detailed below.

A) Signature-Based Detection

A file encases a program structure, aiding in identifying the contents of a file; known malicious files can therefore be matched to this signature. Whilst effectively used in several commercial antivirus software, it has difficulty detecting unknown signatures, and detected malware can evade detection by generating a new signature (Aslan & Samet, 2020).

B) Static Analysis

Static analysis pertains to analysing a binary before execution (Monnappa, 2018). Whilst it has its limitations in retrieving data, it is commonly used to outline the malware's capabilities, divulged by the API calls, function names, variables, or hash values (Barker, 2021). This technique is commonly used with dynamic analysis to comprehend the malware's essential functionality.

C) Dynamic Malware Analysis

Malware analysis employs dynamic analysis to gain a deeper insight into the malware's capabilities, a technique that involves the execution of malware on a system. Dynamic analysis involves monitoring processes during execution, including process monitoring, file system monitoring, network monitoring and registry monitoring (Monnappa, 2018).

D) Dynamic Monitoring of Files Operations

Dynamic monitoring of file operations involves observing operations taken within the computer, usually completed through an automatic framework that detects a substantial number of commands used, such as delete or copy, displaying signs of a security breach.

E) File Extension Blocklist

Blocklists are commonly utilised in the industry; they involve blocking specific file extensions, frequently blocklisting .exe or .pdf files, typically extensions associated with malicious activity.

F) Application Allow List.

Contradictory to blocklists, allow lists are a set of principles in place to allow specific applications, typically through a list of properties that the application must adhere to prevent it from being blocklisted.

G) Honeypots

Honeypots are implemented as vulnerable systems within a network to gain intel on adversaries and understand the tools and techniques repeatedly used by malware authors. These newly examined techniques contribute to improving anti-malware solutions (Matin & Rahardjo, 2020).

H) Integrity Checking

Integrity Checking refers to the identification of a file via fingerprinting – typically through the use of hashing to ensure the integrity of a file, ensuring the file remains untampered within the transmission.

I) File Entropy

Malicious files tend to have a high file entropy, determined by the file's uncertainty and randomness of its structure, resulting in an incoherent series of bytes; this discrepancy often is used as a metric for detecting packed or encrypted files (Marak, 2015).

J) Machine Learning Behavioural Analysis

Machine learning can aid the detection of malware paired with a detection algorithm or dataset. Matin & Rahardjo. (2020) presents a working illustration of detection by classifying a class within a given dataset, outlining an algorithm that detects malware based on features and behaviour with high accuracy.

2.5 Evasion Techniques

A) Oligomorphic & Polymorphic

Oligomorphic malware variants can mutate their decryptor. Despite this capability, they remain detected via signature-based detection. Polymorphic malware aims to fix this limitation by creating countless unique decryptors paired with other obfuscation techniques, such as dead-code insertion and register reassignment (You & Yim, 2010).

B) Metamorphic

Metamorphic malware variants rely on constant content changes within the malware itself rather than encryption methods present with a mutation engine changing the structure of the malware (Tahir, 2018). Therefore, metamorphic malware can evade antivirus solutions due to its constantly changing signature. (Alsmadi & Alqudah, 2021)

C) Encryption

Encryption is essential for ensuring the confidentiality of data by concealing plain text data into ciphertext; unfortunately, utilised by malware authors to achieve evasion. Typically, the main body of malware is encrypted, and the decryptor runs upon execution (Rad et al., 2012).

D) Dead Code Insertion

Dead Code Insertion, often referred to as junk code, is an easy way to change the signature of a malicious file without changing its execution routine (Tahir, 2018). It is typically achieved by inserting NOP – a mnemonic utilised within C to insert no operation. The instruction does not change the state of any registers or memory, allowing malware authors to implement an uncomplicated methodology to bypass signature-based detection (Rad et al., 2012).

Barria et al. (2016) illustrated a working implementation of dead code insertion, applying the procedure alongside replacing the byte's signature using AvFucker, repeating the process until successful evasion.

E) Register Reassignment

This technique utilises the registers, changing or reassigning the value of a register; this could be by reassigning the EAX register to the EBX (Sihwail et al., 2018). Whilst this is another simple evasion technique, applied alongside other evasion techniques can make it difficult to detect (Tahir, 2018). Like Dead Code Insertion, it does not affect the program's behaviour (You & Yim, 2010).

F) Instruction Replacement

This technique involves the replacement of standard instructions with those of equivalent meaning – similar to the use of synonyms and, therefore, does not change the behaviour of the malware, making it difficult to detect. (Tahir, 2018)

G) Subroutine reordering

This technique involves the rearrangement of subroutines, typically in a random order, ensuring that the appearance of the malware changes, but the behaviour remains consistent amongst generations (Sihwail et al., 2018).

Although a substantial number of techniques viruses can manipulate for successful evasion, these outlined techniques are most commonly utilised amongst malware, often paired concurrently with another to ensure evasion is achieved; upon discovering additional evasion techniques, a definition will be given.

2.6 Anti-Analysis

Anti-VM Techniques can be broken down into subcategories and involve validating that the host system is not emulated by checking interaction and artefacts based on timings. Malware uses these indicators to masquerade behaviours; upon detection of unique artefacts, including registry keys or characteristics of CPU instructions, the malware can detect the emulated environment and changes its behaviour to prevent analysis (Chen et al., 2016). Anti-Debugging techniques prevent the malware from running upon detecting a debugger, which involves checking Windows APIs, flags or instruction sets. Upon detection, the malware could change behaviours or terminate itself (Olaimat et al., 2021).

2.7 Related Works

Previous studies have established the analysis of evasive techniques, and comprehensive surveys include (You & Yim, 2010; Marpaung et al., 2012; Schrittwieser et al., 2017) synthesising a collection of malware evasion techniques, whilst the only constraint of them is being dated, highlighting a gap in the literature, paving the importance of an up-to-date combination of current evasion techniques. Galloroa et al. (2021) aim to address the gap in the literature with a systematic review of 92 evasion techniques; whilst the literature is an excellent combination of malware techniques, its limitation lies within the chosen dataset containing software not deemed malicious, as it is common for benign software to use obfuscation methods to protect intellectual property, thus contributing to the dataset.

Extensive research accomplished within the industry precisely detailed detection methods for malware obfuscation, typically applying a chosen methodology for malware detection. Literature about the detection was initially excluded due to its lack of theoretical definition of evasion strategies and its focus on its implemented detection method. However, it must still be acknowledged due to its relevance within the industry.

Figure 1.4

Reference	Detection Method/Algorithm	Date
(Li et al., 2009)	Maximal Patterns	2009
(Lu et al., 2013)	ENDMal	2013
(Narouei et al., 2015)	DLLMiner	2015
(Kirat & Vigna, 2015)	MalGene	2015
(N. Khasawneh et al., 2017)	RHMD	2017
(Kim et al., 2017)	DynODet	2017
(Dai et al., 2019)	SMASH	2019
(Carrier et al., 2022)	Memory Feature Engineering	2022

Detection Algorithms Utilised

Li et al. (2009) evaluates the efficacy of applying detection via maximal patterns, and the pattern is extracted from system calls during runtime and then compared to another pattern sequence. Whilst showing effective results, the limitations are using small family sizes and only using worms, trojans and backdoors. In addition, Lu et al. (2013) also propose a detection algorithm based on malware behaviours, focusing on system calls. Whilst both illustrate an effective way of detecting obfuscated malware, the caveat is the age of the algorithms. However, Narouei et al. (2015) & Carrier et al. (2022) exhibit more modern iterations of these detection algorithms, still utilising detection based on features, such as system calls and API sequences. Carrier et al. (2022) exhibit promising results using a dataset containing samples from modern malware families, including spyware, ransomwares and trojan horses, using memory feature engineering, suggesting that machine learning improves the accuracy when detecting evasive malware. Dai et al. (2019) reinforces the use of API calls to detect obfuscated malware, displaying the effectiveness of behaviour-based detection methods using SMASH, combining software and hardware features paired with a neural network to detect malware based on its features.

A combination of these malware detection methods conveys the advantages of using a detection method based on the behaviours malware exhibit's displaying the effectiveness of algorithms over the industry-implemented signature-based techniques; however, they frequently show the limitations and difficulty of malware analysis, as solutions are regularly proposed, malware authors are already ahead of the curve, generating new effective means of evading detection.

2.8 Conclusion

In conclusion, this chapter carried out the search strategy leading to the successful identification of literature. It proposed an identification phase that generated a keyword list, allowing for a comprehensive literature search. An overwhelming amount of search results preceded a critical search strategy that led to excluding literature based on strict criteria. The chapter also defined standard malware evasion techniques, comparing them against existing evasion detection methods whilst showing popular implementations of these evasion strategies. Literature about the implementation of algorithms was analysed, resulting in a way to identify evasive malware based on behaviours with high success and accuracy.

Chapter 3: Methodology

3.1 Introduction

In-house sandboxes provide several overwhelmingly positive attributes for malware research; malware intelligence reports can be generated within seconds, providing the user with a detailed report on the malware's family and behaviours, with it even generating reports on network traffic, evasion and even dropped files, with a plethora of sites to choose from, each with their advantages and disadvantages. This chapter will therefore present an approach for gathering primary research, justifying design choices, and highlighting any issues and limitations within the approach.

3.2 Research Design

In order to format a research design, the research question outlined in the Introduction needed to be apprehended:

- How has Malware Evasion evolved and changed over time?

This outlined research question would effectively be answered using an experimental analysis approach, applying a mixed method approach of quantitative and qualitative data. This approach is appropriate due to the nature of data collection; the retrieved results from the used dataset would return factual statistical data, which would need human interpretation to synthesise the results. This decision was made because it would provide specific results, and the produced artefact could be reproduced for similar research ideas. Additionally, the design allows for a dataset over manual analysis, abolishing any ethical implications of the project and allowing the project to maintain the given timeframe. Whilst advantageous, limitations arise when considering the validity and comprehension of the results. Incorrect statistical analysis will invalidate any significant research findings while being prone to bias upon evaluating results. Despite this, the chosen design is the best given the project's aims, as it will provide specific and consistent results while producing a generalisable and replicable artefact, generating new insights into malware research. The project will select an experiential research style involving the evaluation of secondary sources. This is the best approach because the project involves collecting secondary data via an automated sandbox, allowing for specific conclusions to be drawn whilst providing a high level of control, permitting the isolation of specific variables to enrich the analysis. However, disadvantages arise when considering it is a taxing task and prone to human error; despite this, mitigation involves following a timeline, allowing breaks to prevent misinterpretation of the data. This approach is used in the existing literature; replicating it will ensure an impressive congregation of evasion techniques capable of answering research questions and meeting requirements.

3.3 Approaches

Virus Share, a collaboration of over 58 million malware samples, was chosen to create a large dataset of impartial malware samples, these samples whilst malicious, they were not explicitly tagged as evasive, therefore providing enrichment to the final dataset as the results would show a clear contrast in evaluating the effectiveness of specific evasion techniques. This approach was accomplished via a looped wget command displayed in Figure 1.5. However, this dataset quickly became overambitious, creating an enormous dataset that promptly exceeded any method of automatic data collection within the project’s timeframe, quickly prompting other techniques to gather a usable dataset.

Figure 1.5

First Data Collection Attempt

```
"for i in {00000..00445}; do wget
https://virusshare.com/hashfiles/VirusShare_$i\.md5; done"
```

Another approach was to use MalwareBazaar’s inbuilt “Tags” method to search the site for malware tagged with “Evasive” as MalwareBazaar has an easy-to-use API that could automate the process, with the ability to acquire a daily CSV, quickly providing a dataset for analysis. Theoretically, this was a good approach, if not for MalwareBazaar’s seemingly randomly tagged samples. In addition, there was no way to ensure that the project remained on scope due to the site containing malware including previously excluded operating systems. Although these approaches contained weaknesses, they provided a structure for the chosen approach, albeit with a smaller dataset.

The final chosen approach was to obtain a dataset from the University of New Brunswick titled “CIC-MalMem-2022”, a modern dataset frequently used via malware researchers whilst also being commonly applied to review the accuracy of modern detection systems (Talukder et al., 2022; Mezina & Burget, 2022). Ensuring key factors, guaranteeing the data was of industry-standard whilst also being modern, and certifying that the project remained in scope due to the dataset containing families of spyware, trojan horses and ransomwares, and avoiding out-of-scope operating systems. The dataset contains 58,596 records, with 29,298 benign and 29,298 malicious. To create a smaller, more targeted sample size, only the 29,298 malicious samples would be considered, opting to remove benign samples as they would provide unnecessary results. The decision was appropriate because it aligns with the research design and project aims, providing a quality dataset that can be investigated using an applied methodology. The next stage of the methodology was to find a suitable way of analysing the chosen dataset. As the sample size is too large for manual static analysis, an online sandboxing site will be used, as they typically contain an API that can be manipulated to gather the required data. Whilst there is a magnitude of sandboxing sites, only three were considered: VirusTotal, Malware Bazaar and Hybrid Analysis, displayed in Figure 1.6

Figure 1.6

Malware Sandbox Characteristics

Characteristic	VirusTotal	Malware Bazaar	Hybrid Analysis
Easy to Use	✓	✓	✓
Good Documentation	✓	✓	✓
Reliability	✓	✓	✓
Aligns With Scope	✓	X	✓
Sign Up Required	✓	✓	✓
Limits	500/Day	2,000/Day	200/Hour
Sandbox Reports	✓	✓	✓
General Information Reports	✓	✓	✓
Behavioural Reports	✓	X	✓

From an initial review of the characteristics exhibited by each API, it was clear that either VirusTotal or Hybrid Analysis would be employed to retrieve results, as they better aligned with the project's scope; whilst Hybrid Analysis does generate required results, sample hashes sent to the website from the chosen dataset divulged no results, opposing to VirusTotal's rich output of dynamic behavioural analysis paired with its output of general results such as upload date, name, file size and type. In addition, the introduction of VirusTotal's new API v3 allows for specific queries containing well-document object attributes, permitting a straightforward way to hand-pick attributes directly correlating with the project's scope. Whilst removing irrelevant results and facilitating the development of legible, well-formatted output.

The query to gather data was “/behaviours”, selected from a viable option of choices. This is because the endpoint returns behavioural data from each sandbox in a JSON format, returning 200 upon a successful result. The main reason for this choice is that it returns all behavioural data. If any design decisions occur, the data will be present and must be appended to the final output file rather than re-analysing each hash from the dataset. This query also returns mitre attack techniques, a taxonomy of techniques representing how an adversary achieves an objective (ATT&CK, 2018 ; Chierzi & Mercês, 2022), providing an opportunity to extract additional defensive evasion techniques.

The penultimate task for a completed approach was to find a way of processing the JSON data in a presentable way, as it needs to be done in bulk; due to the project timeframe, it was necessary to use a programming language as a parser for the data. This is also needed to automate requests sent to VirusTotal. The well-documented API has an exemplary script in various languages. One of them is Python, a familiar language that is easy to learn and read, with popular community support and documented libraries for converting JSON data into a CSV format. For these reasons, Python will be used to create the script capable of formatting the dataset. The script must meet the requirements outlined in the Design chapter to ensure the project’s goals are met. The outlined requirements within Chapter 4 will enable the artefact to fulfil the project’s goals; the dataset will be collected from the initial requirements, and a smaller dataset will be produced from the following requirement. The requirements directly correlate with the research design, enabling the results to be collected from the dataset and allowing for the analysis and critical discussion of the results.

3.4 Conclusion

Overall, this chapter theorised an applicable methodology that can be applied to any chosen dataset. The chosen methodology was to utilise VirusTotal’s API v3 to retrieve a behavioural report on the requested hash; the applied process would be automated due to the size of the dataset. This chapter also outlined the requirements and clear objectives of the script, justifying the reasoning for each objective and selecting Python as the primary programming language due to its familiarity and application within the scenario.

Chapter 4: Design

4.1 Introduction

Data science is extracting meaningful information from a given data set. Python was employed as it is the preferred language for dealing with statistics due to its expansive libraries, including pandas. This library can convert data into set data frames, allowing for conversion to a CSV format (Dixit, 2022). This chapter will discuss the design decisions of the previously chosen methodology, the development of the artefact and how it gathers, processes, and parses the inputted data.

4.2 Software Methodology

Consulting changes and delays within the project timeline, it was worth considering the implementation of a software methodology with a focus on implementation rather than planning, as it needed to be developed quickly to gather the necessary information in time. Therefore, a focus was put on coding based on trial and error rather than one involving significant planning, opting for implementing rapid application development (RAD) (GEAMBAŞU et al., 2011). The project's requirements were defined, and then the coding quickly began to make a prototype capable of gathering data. This decision was due to VirusTotal's well-documented API; alongside the community support available for Python's panda's library, it was assumed the program could be made within two weeks. To get the most out of this methodology, a small plan of what the final dataset should look like was created and displayed in Appendix C.

4.3 Design of Script

The intended approach was to have one large Python script achieving all of the requirements within the methodology. However, this would have made a situational script that could only be used if certain conditions were met, leading to multiple scripts and splitting the requirements into specific sections. This approach is better suited as it allows for more flexibility. If the dataset yields no information, a new dataset can be quickly put in place without errors, as certain parts of the script would be redundant based on other scenarios. This approach also allows the methodology to be replicable across any given dataset. There are few research scripts involving VirusTotal's newer API v3 that gather specific data and convert it to a CSV file meaning that the application of this script will be widely applicable within malware research.

Figure 1.7

Script Requirements

Requirement	Script
• Read Data from a disk • Identify Hash Type • Scrape the "CIC-MalMem-2022" dataset for hashes and output them into a text file, allowing for the removal of non-malicious samples. • Filter out specific Hashes • Removing duplicate hashes within the dataset, preventing redundant data and the waste of daily API quota.	getHashes.py
• Read Data From a disk • Understand API documentation • Obtain API key • Handle API Response • Choose an HTTP client • Query VirusTotal for the behaviours of inputted hash • Keep a counter of how many times the script has run • Use a counter to keep track of hashes left in the text file, updating after each iteration	queryVirusTotal.py
• Read JSON data from a disc • Convert JSON to CSV • Identification of headings/headers • Ensure data integrity • Ensure data matches up with heading • Exporting JSON data to CSV, providing a presentable way to review results.	createCSV.py

Figure 1.6 illustrates the requirements and the script created to meet the requirement; this approach allows for the timeframe to meet as VirusTotal only allowed for 500 requests a day. The data collection could start without knowledge of the panda’s library, as the JSON output would vary depending on the query. The first two scripts were based on the scenario of the chosen dataset, and after they were run once, they were not needed again, therefore opting to leave them out of the main scripts as they quickly became redundant. Their main goal was achieved through regular expression, using ‘[a-f0-9]{64}’ to gather the SHA256 strings from the dataset. The file is then returned to a function called ‘removeRedundacies’. It achieved its requirement through a loop, initialising a variable, gathering the unique lines from the text file writing these two in the text file called “final_datset.txt.”.

The script utilised to gather the data was “queryVirusTotal.py”, retrieving the behaviours of the hashes within the dataset; this was easily achieved by reading the dataset that removeRedundacies produced within a loop, using hash as a variable to read lines from the text file, the hash variable was then passed into the “request.get” function and was able to query VirusTotal successfully. An approximate timeline for how long data collection would be around seven days, leaving additional time for perfecting a solution for data outputting and formatting. As the “final_dataset.txt” had around 2300 hashes, it would take around five days to query based on the daily quota. Additional time was added to deal with any issues and test a solution. The final iterations of the script are demonstrated in the implementation chapter.

The final stage was to parse the JSON data using pandas. This was achieved by understanding the JSON response, using Jupyter Notebooks as a testing platform to understand how the data would be presented using panda’s library (Molin, 2021). As the JSON files varied in size, several IF statements were used; as not all JSON files contained the same data, the script would crash if specific data requested was not present, and therefore it was logical to implement. VirusTotal’s well-documented API made this straightforward, as the objects within the JSON file were already known and would need to be selected based on relevancy. These objects will allow for data filtering by their analysis date, answering the research questions and extracting objects that malware authors typically manipulate to achieve evasion. To ensure a rich final dataset, the following objects were extracted:

- analysis_date
- modules_loaded
- mutexes_created
- mutexes_opened
- registry_keys_set
- modules_loaded
- calls_highlighted
- processes_terminated
- ip_traffic

4.4 Conclusion

This chapter documents the chosen software methodology, detailing why it was picked and alternatives considered before discussing the tested implementations of the scripts. Following the justifying of the script’s performance, discussing its segmentation into smaller scripts allows for the queries to be carried out to ensure that the timeframe was met. Concluding by detailing the conversion to JSON and the objects that would be extracted from the JSON files.

Chapter 5: Implementation

5.1 Introduction

This chapter will briefly review and justify scripts used in the retrieval and manipulation of the data, explaining its function, how it meets the requirements and aiding in the reproducibility of the project.

5.2 Retrieving Hashes

Figure 1.8

getHashes.py

```
def retriveHash():
    with open('dataset.csv', 'r') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        file = open("dataset.txt", "r+")
        for column in reader:
            try:
                hashes = (re.search("[a-f0-9]{64}", column[0]))
                file.write(hashes[0]+ "\n")
            except:
                pass
    return file

def removeRedundancies():
    with open('dataset.txt', 'r') as file:
        lines = file.readlines()
        unique_lines = set(lines)
        with open('final_dataset.txt', 'w') as f:
            for line in unique_lines:
                f.write(line)
```

Figure 1.8 illustrates the first script, "getHashes.py", with the main function "retriveHash" reading a csv file, and searching for a SHA256 hash, achieved through a for loop nested within a try and except block, due to not all records containing SHA256 hashes, therefore passing if a hash was not found. However, if a hash was found, it would be written to a file. The dataset contained duplicates, requiring a solution to remove them. "removeRedundancies" was employed to ensure that unique hashes were put into the final dataset. An explanation for this implementation is that the dataset contained "Begin" records; these records were not needed in the final dataset as they were seemingly harmless applications and therefore would not enrich the dataset, whilst removing the redundancies would mean no surplus data, and ensured the project timeline remained, due to VirusTotal’s API limitations.

5.3 Challenges

Challenge 1: Status Codes

The first initial challenge with the script came from response handling. As the script was in a for loop, it would need to have a function to stop the script after 500 runs; initially, a counter function was implemented, and upon reaching count = 500, the script would terminate. However, this led to the script continuing to write JSON files, despite containing no data. A solution was achieved by limiting the script’s runtime based on the server’s response. For example, if it were equal to 200, indicating a successful response, it would be written to a JSON file. Whilst 429, indicating that the daily quota was met, would mean the script’s termination. This was implemented to prevent empty JSON files from being written, as they would crash when converting to CSV due to “JSONDecodeError” whilst also preventing similar problems with status codes, as anything other than 20x status codes would result in an error.

Challenge 2: Integrity of Information

Initially, how the script was implemented caused an error in which two responses would roll into one response file, causing an invalid JSON file and limiting its use to be parsed. This error was not initially known due to how error handling was handled, and it would skip any JSONDecodeErrors and only present files containing a valid JSON. This was a severe issue and would heavily limit the project results. Whilst an easy fix, it raised questions regarding the integrity of responses. A simple experiment was conducted using curl to confirm that the data contents were the same as that of the implemented script. The same query would be used for both the script and the curl command, and three JSON files were chosen at random; the files were then compared using a script to confirm that the contents of the files were identical.

5.4 Querying VirusTotal

Figure 1.9

getHashes.py

```
def queryVirusTotal():
        counter = 1
        with open("final_dataset.txt", 'r') as file:
                for fileHash in file:
                response =
requests.get("https://www.virustotal.com/api/v3/files/"+
str(fileHash).rstrip()+ "/behaviours?limit=10", headers=headers)

                with open(str(fileHash).rstrip() + '.json', 'w') as f:
                        f.write(response.text)
                        print("Response: ", response, "Hash Name: ",
fileHash, "Samples Collected : ", counter)
                        counter += 1
                        if response.status_code == 429:
                        updateTxt(counter)
                        break

def updateTxt(counter):
        with open("final_dataset.txt", "r") as file:
        removeLines = file.readlines()
        with open("final_dataset1.txt", "w+") as file2:
        file2.writelines(removeLines[counter - 2:])
```

Figure 1.8 depicts the Python script utilised for data collection, opting to choose the behaviours endpoint when using VirusTotal's API, as it contained all behaviour reports for the given hash. This was easily achieved by reading the dataset that "removeRedundacies" produced within a loop, passing the hash value from the list into the URL parameter, and then writing it to a JSON file. Additional code prints the progress by printing the status code, hash name, and counter. Typically, the loop would stop at around 200-300 hashes and need to be run twice a day. The function "updateTxt" was used to keep count of the script's progress; each time the loop would run, it would add to the counter. Upon any status code other than 200, it would stop the loop and pass the counter to "updateTxt", ensuring that no hash was queried twice, using string slicing to cut the txt file.

5.5 Results To CSV

The JSON response received from Virus Total was embedded, making it harder to access each file due to each file containing different information, as some files had data at different levels of the JSON file. This meant that the data was harder to extract than initially thought; a flexible solution was to gather the data needed by using regex, as this would ignore any embedded levels and allow a solution that could be applied to all of the JSON files while not the best strategy for parsing JSON, it did allow for a successful way to gather the data. This can be seen as a project limitation as it is not an appropriate way to parse JSON data and could lead to inconsistencies. However, the files were manually checked to ensure that the data matched that in the CSV file. Figure 2.1 demonstrates the final script used to gather data, with each data having multiple variables being passed to cleanData, to format each item in the list, displayed in Figure 2.0

Figure 2.0

cleanData Function

```python
def cleanData(data):
        noSpace = []
        line = [i.replace('\n  ',"") for i in data]
        noSpace = [x.strip(' ') for x in line]
        line1 = [i.replace("[ ", "")for i in noSpace]
        line2 = [i.replace(" ", "")for i in line1]
        line3 = [i.replace(",", "'\r\n'")for i in line2]
        newString = "\r\n".join(line3)
        return newString
```

Figure 2.1

createCSV.py

```
def getData():
        dataFrame = pd.DataFrame(columns=["Hash", 
"AnalysisDate","modulesLoaded", "mutexesCreated", "mutexesOpened", 
"registryKeysSet", "callsHighlighted", "processes_terminated", 
"ipTraffic"])
        with open("final_dataset10.txt", 'r') as file:
        for hash in file:
                with open(str(hash).rstrip() + '.json', 'r') as access_json:
                access_jsone = json.load(access_json)
                prettyJSON = json.dumps(access_jsone, indent=4, 
separators=(',', ': '), sort_keys=True)
                analysisDate =  r'"last_modification_date":\s(\d\s*){10}'
                analysisDateData = re.findall(analysisDate, prettyJSON)
                modulesLoaded = r'"modules_loaded"\s*:\s\[([^]]+)\s]'
                modulesLoadedData = re.findall(modulesLoaded, prettyJSON)
                mutexesCreated = r'"mutexes_created"\s*:\s\[([^]]+)\s]'
                mutexesCreatedData = re.findall(mutexesCreated, prettyJSON)
                mutexesOpened = r'"mutexes_opened"\s*:\s\[([^]]+)\s]'
                mutexesOpenedData = re.findall(mutexesOpened, prettyJSON)
                registryKeysSet = r'"registry_keys_set"\s*:\s\[([^]]+)\s]'
                registryKeysSetData = re.findall(registryKeysSet, 
prettyJSON)
                callsHighlighted = r'"calls_highlighted"\s*:\s\[([^]]+)\s]'
                callsHighlightedData = re.findall(callsHighlighted, 
prettyJSON)
                processesTerminated = 
r'processes_terminated"\s*:\s\[([^]]+)\s]'
                processesTerminatedData = re.findall(processesTerminated, 
prettyJSON)
                ipTraffic = r'"ip_traffic"\s*:\s\[([^]]+)\s]'
                ipTrafficData = re.findall(ipTraffic, prettyJSON)
                cleanModulesLoadedData = cleanData(modulesLoadedData)
                cleanMutexesCreatedData = cleanData(mutexesCreatedData)
                cleanaMutexesOpenedData = cleanData(mutexesOpenedData)
                cleanRegistryKeysSetData = cleanData(registryKeysSetData)
                cleanCallsHighlightedData = cleanData(callsHighlightedData)
                cleanProcessesTerminatedData = 
cleanData(processesTerminatedData)
                cleanipTrafficData = cleanData(ipTrafficData)
                data = {"Hash" : hash,
                        "AnalysisDate" : analysisDateData,
                        "modulesLoaded" : cleanModulesLoadedData,
                        "mutexesCreated" : cleanMutexesCreatedData,
                        "mutexesOpened" : cleanaMutexesOpenedData,
                        "registryKeysSet" : cleanRegistryKeysSetData,
                        "callsHighlighted" : cleanCallsHighlightedData,
                        "processes_terminated" : 
cleanProcessesTerminatedData,
                        "ipTraffic" : cleanipTrafficData
                        }

                dataFrame.loc[len(dataFrame)] = data
                print(dataFrame)
        dataFrame.to_csv('list.csv',index=False)
```

5.6 Conclusion

This chapter discussed the implementation of all the scripts used within data collection, detailing their usage and justifying design choices. Following this, the discussion of exploring the challenges encountered when programming, how these problems were solved, and ensuring that any similar problems were also resolved. Finally, the chapter concluded by discussing extracted objects from the JSON files, the use of regex and how data would be displayed within the CSV file.

Chapter 6: Analysis

6.1 Introduction

This chapter will focus solely on interpreting results from the created dataset discussed within the implementation; creating appropriate tables was necessary to present the obtained results, commencing with ensuring data integrity by validating the results.

6.2 Data Validation

To ensure data validation, three JSON files were picked randomly, opened within JSON format, and then compared findings to the CSV file and VirusTotal findings. To ensure a fair test, the constant variable "ip_traffic" was selected, as this was present in most files, illustrated in Figure 2.2. Upon successfully validating the results, the csv undertook sequential examination, commencing with an analysis of the highlighted calls.

Figure 2.2

Validation of Data

File	IP Address In CSV	IP Address In JSON	IP Address on VirusTotal	Valid?
1	142.250.184.110	142.250.184.110	142.250.184.110	✓
2	209.85.145.101	209.85.145.101	209.85.145.101	✓
3	218.54.31.226	218.54.31.226	218.54.31.226	✓

The CSV file was analysed over time, focusing on critically evaluating malware evasion techniques, reinstating the research question: How has Malware Evasion evolved and changed over time? Therefore, the data selected for presentation would directly seek to answer the given question, choosing the data, and sorting it by popularity, novelty, obscurity and usage, highlighting its changes over time.

6.3 Calls Highlighted

System calls are present within programs to request a service from any given source, typically the system kernel; whilst not initially malicious, they can give a succinct overview of a programs application; therefore, static analysis is often employed to check these imports (Sikorski et al., 2012).

Figure 2.3

Most Common Calls Highlighted

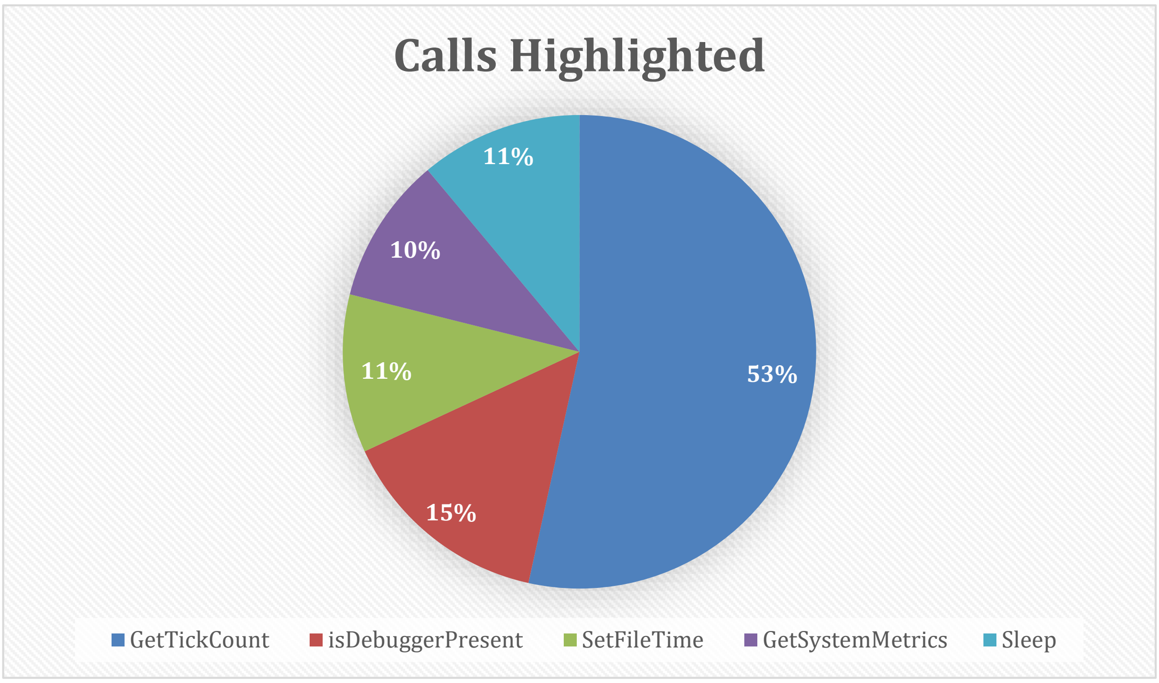


Figure 2.3 illustrates the most popular calls highlighted from the dataset, with 53% of samples containing "GetTickCount", a standard anti-analyst method that determines system run time, returning a value in milliseconds if the value is less than a specific time stamp, the system will exit (Wartell, 2015). Whilst "Sleep" has a magnitude of uses within the malware, it is commonly used as delitescence, avoiding early detection by waiting for an event of trigger (Oyama, 2018); with both techniques demonstrating simple timing-based evasion, its practicality comes malware not demonstrating its intent within the timeframe, which is usually a restrictive characteristic of online sandboxes with them typically running less than 10 minutes (Nunes et al., 2022). Following this, "isDebuggerPresent", another anti-analysis technique that

detects the presence of a debugger, with a multitude of bypass options and thus an inadequate method of evasion, utilised 15% of the time with its implementation attempting to thwart dynamic analysis, terminating execution if the EAX = 1 (Apostolopoulos et al., 2020). Characteristics of online sandboxes are often exploited to evade detection; "GetSystemMetrics" enumerates system information, for example, enumerating screen size, as sandboxes rarely use multiple screens; upon detection of unusual resolution metrics, the application will determine that it is being executed in a virtualised environment.

SetFileTime is used to conceal file access (Nunes et al., 2019). Additionally, employed as an anti-forensic technique called timestomping, the act of changing metadata, usually to a time before infection, delaying detection (Bhargavi & Rai, 2022).

Figure 2.4

Obscure Calls Highlighted

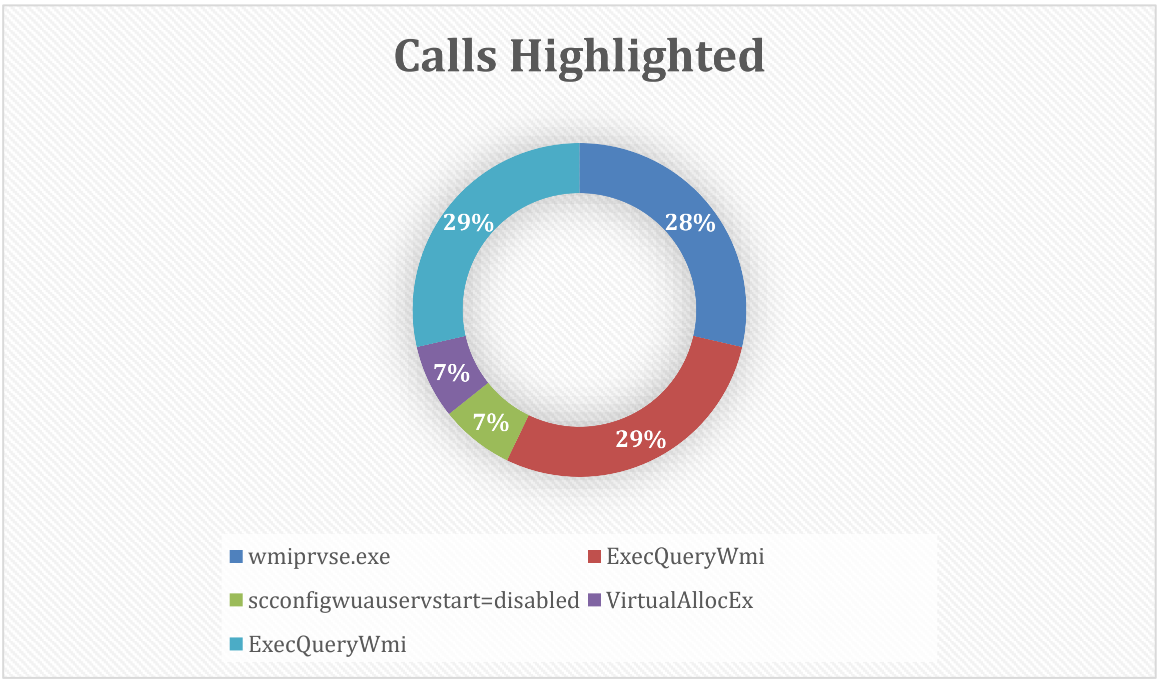


Figure 2.4 exemplifies more obscure malware evasion techniques, with only around 14 appearances from the dataset exhibiting these behaviours. "VirtualAllocEx" enables malware to hide its presence by running in the context of another process by accessing benign applications' virtual memory, thus evading detection. It does this by suspending a target

application, copying the executable code into the address space and resuming the thread, or creating a new thread within the target process, replacing the start address with the executable code (Willems et al., 2007).

Hook injection is a technique used to alter the usage of internal instructions, “SetWindowsHookExW” is used for this case, logging input events enabling malware’s functionality, whilst it can also be used to alter system messages, avoiding detection (Jeong et al., 2021). Hook injection is effective because it injects into the memory space of a legitimate process, modifying the injected code to achieve its objective and is often difficult to detect as a result.

Whilst not specifically a function call, “scconfigwuauservstart=disabled” was detected; this would be inputted after calling cmd.exe, enabling the malware to disable Windows update services, leaving the machine exploitable, whilst potentially preventing the removal of malware, whilst only being present once in the dataset.

WMI (Windows Management Instrumentation) is the management of Windows software and hardware, using standard functions to query processes; “ExecQueryWmi” arose around 29% of the 14 more obscure samples. This is used to query a specific parameter and can be anything from processor information to video controllers and therefore evade detection based on specific parameters (Ladutska, 2022). Examples include checking if the number of processes is smaller than usual and checking hard disk size and networking settings. These indicators would reveal a virtual machine, and the sample would evade based on an unusual characteristic, demonstrating another anti-sandboxing technique.

Other popular methods involved replacing common API calls for their kernel counterparts for an attempt at novelty; for example, " isDebuggerPresent " appears 222 times within the dataset and also appears as “kernel32.isDebuggerPresent” whilst they are not the same function, they do perform the same operation.

Figure 2.5

Calls Highlighted Countermeasures

Calls Highlighted	Countermeasure
Sleep	Sleep Skipping
SetFileTime	Monitor anomalies/inconsistencies in the file metadata
isDebuggerPresent	Set EAX = 0 after IsDebuggerPresent is called Fill with NOP
GetSystemMetrics	Match Screen resolution to host

Figure 2.5 encapsulates countermeasures that can be used to prevent each discussed evasive strategy. Sleep skipping is the process of reducing the duration of sleep, achieved through a debugger. The implementation is not always usable if the malware exhibits nondeterminism as it changes program behaviour and will affect analysis (Oyama, 2019). In addition, “GetTickCount64”, used around 2.88% of total system calls, can be employed to detect sleep skipping by including the amount of time that the system was asleep, highlighting the ability to evade any implemented countermeasures further. Other strategies involve minor countermeasures that include patching the program via a debugger or properly implementing a functioning sandbox. A majority of calls highlighted were aimed at evading sandboxes, an expected finding due to their prevalence in malware research.

Figure 2.6

Calls Highlighted Trends

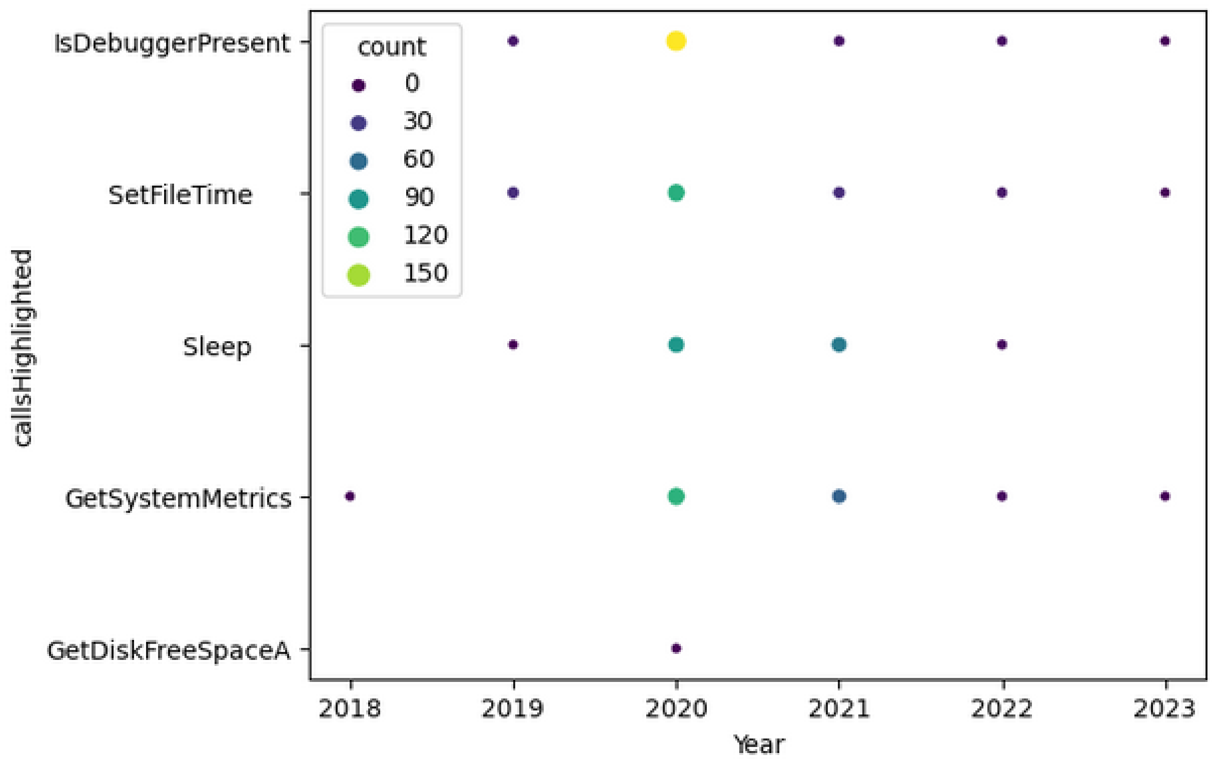


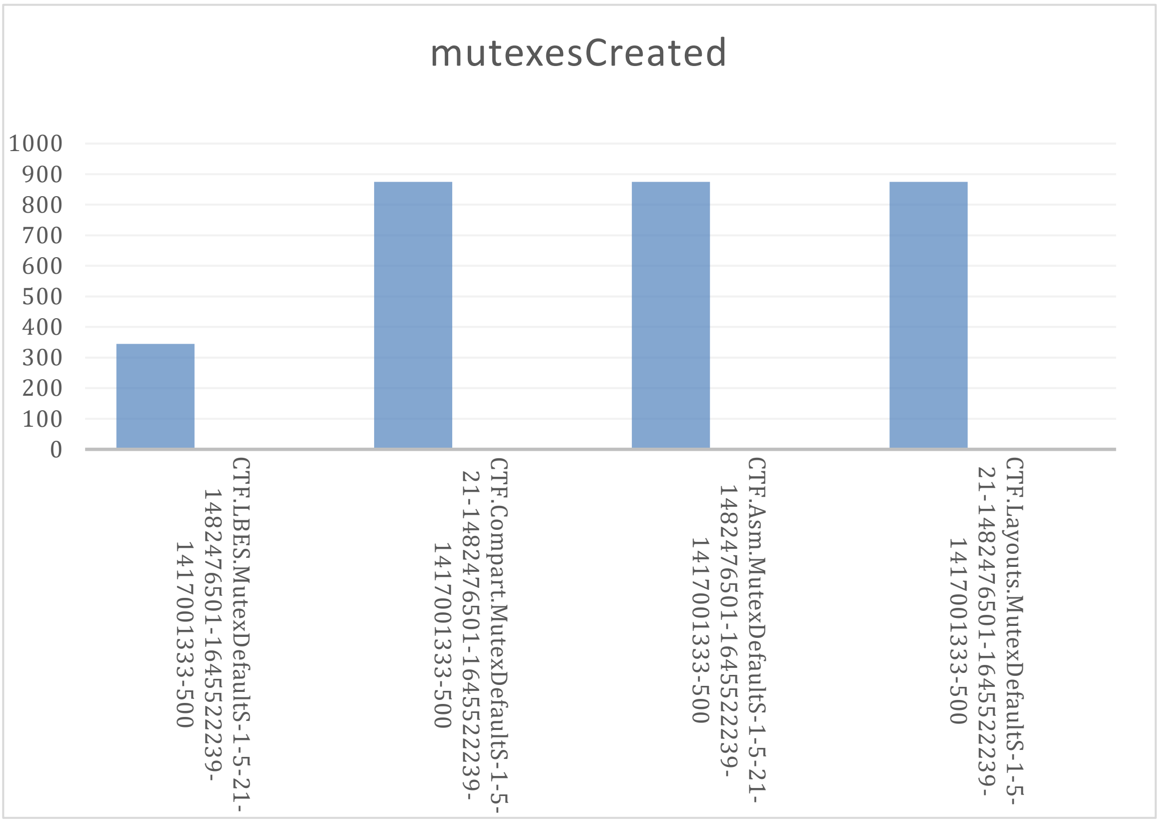
Figure 2.6 illustrates a scatterplot containing popular calls highlighted and their trends overtime; as the dataset is relatively new, paired with the difficulty of obtaining new malware samples, it was evident that there would be insufficient data to conduct this side of analysis; despite this, the most consistently used call was “setFiletime”, with its usage, much like others peaking in 2020. Contrastingly, it illustrates “GetDiskFreeSpaceA” as the least consistently used call, with its only usage implemented in 2020. These results are more likely to indicate limited data rather than demonstrate a trend amongst evasion techniques. Despite this limitation, “Sleep”, “SetFileTime”, and “IsDebuggerPresent” were consistently used throughout the dataset, indicating their influential usage in the creation and evasion of malware.

6.4 Mutexes

Mutexes are used to obtain mutual exclusion, achieving this goal by providing exclusive access to a resource (Golab & Ramaraju, 2016). Malware utilises mutexes to ensure only one instance is running at a time, avoiding reinfection; whilst this is not inherently evasive, evasion comes from naming conventions upon creating and loading these mutants.

Figure 2.7

Most Popular Mutexes Created



Previous analysis of calls highlighted reveals the use of “kernel32.CreateMutexA” whilst its usage in benign applications is known, it can be an indicator of checking for mutexes that are present within virtual environments; an example of this is “Sandboxie_SingleInstanceMutex_Control”, by detecting unique artefacts exhibited by sandboxes malware can seek to avoid divulging is malicious characteristics (Rin, 2013). The mutexes shown in Figure 2.7 are Windows mutex objects containing the SID of the user that owns the mutex, appearing around 875 times each aside from “CTF.LBES.MutexDefaultS-1-

5-21-1482476501-1645522239-1417001333-500" which appears 345 times; due to their regular occurrence within the dataset, it is suspected that they are used to detect the existence of a virtualised environment, rather than as usage for mutual exclusion. Whilst not definitive, it can be inferred that they are trying to avoid a specific sandboxing service due to the SIDs remaining constant within the dataset; there are little to no versions of the same "CTF.LBES.MutexDefaultS" string occurring with differing SID values.

Figure 2.8

Most Popular Mutexes Created

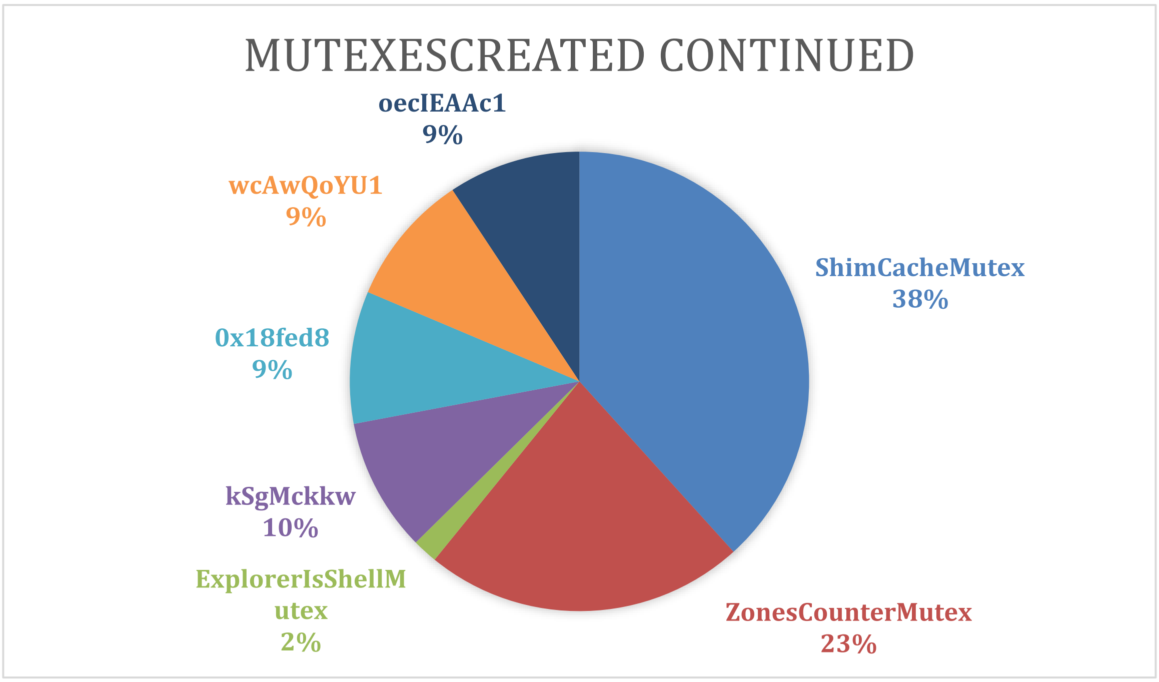


Figure 2.8 illustrates a continued, most popular mutexes created, excluding Windows mutex objects, further consolidating the usage of random arbitrary string names to evade detection, adding an element of unpredictability, contributing to the difficulty of analysis – whilst also potentially being non-deterministic depending on the variant or version of the malware. (Stiborek et al., 2017). Whilst a majority of malware – around 63% used commonly used mutexes, much like the "CTF" mutexes, they are often used to check instances of system mutexes and evade based on these parameters.

Whilst not the most evasive characteristic a malware can use, it has implications for the usage of machine learning for malware detection; a list of polymorphic mutex names can be assembled, employing them as a means for malware detection.

Figure 2.9

Modules Loaded Trends

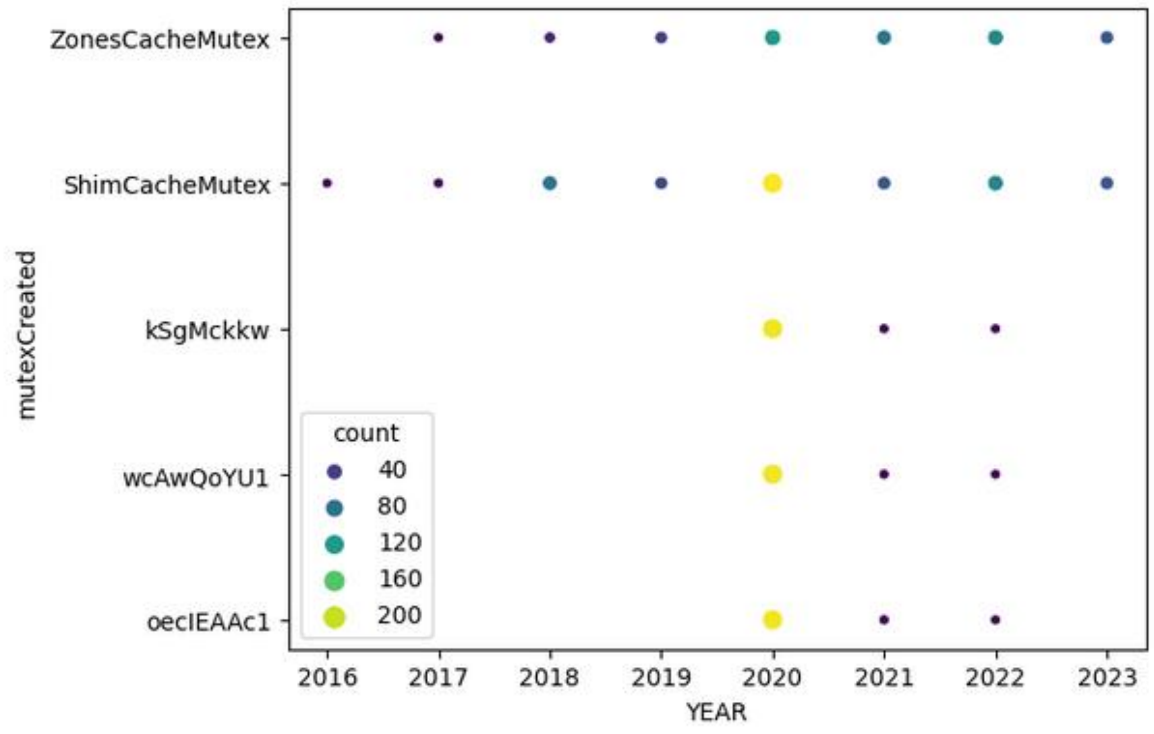


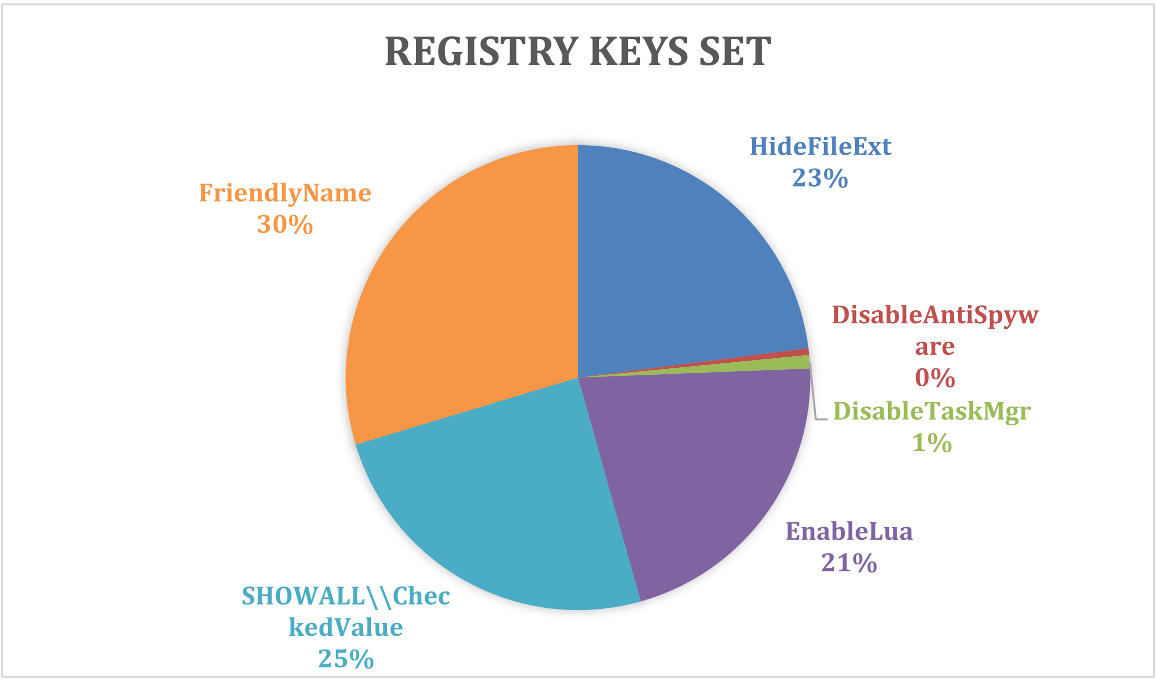
Figure 2.9 annotates trends seen within the modulesLoaded data, showing the “ShimCacheMutex” being the most consistently used from 2016 – 2023, shortly joined by “ZonesCacheMutex” in 2017. 2020 displayed the first usage of arbitrary strings within the dataset; whilst seeing a decline, it is more likely due to the lack of data rather than their usage as a whole, indicating malware authors could opt to use it for future implementations. However, more data is needed to be certain.

6.5 Registry Keys Set

Malware's manipulation of registry keys is well documented; typically used to instil persistence (Sikorski, M., & Honig, A. 2012). Their usage in the context of evasive malware concerns their ability to make system changes without user acknowledgement via disabling UAC or LUA.

Figure 3.0

Most Common Registry Keys Set



Whilst several registry entries were implemented to implant persistence, privilege escalation or as a unique identifier, a limited number of evasion techniques were exhibited, shown in Figure 3.0. "FriendlyName" is a name used to refer to a device or application. It was often given the value of 'admin', used in around 30% of the 644 identified evasive strategies, likely used to hide the technical identifier, replacing it with a trusted privilege account.

Tactics about the observation of file systems involve "HideFileExt", "Hidden", and "ShowSuperHidden", with their usage controlling the behaviour of hidden files and hiding file extensions. "HideFileExt" was the most popular of the three, appearing 23% of the time, unlike "ShowSuperHidden", appearing only four times (Erdelyi, 2004), allowing malware to hide within the file system, remaining undetected. EnableLUA utilised around 21% and was set to dword: 0, disabling approval for the execution of administrative tasks. This is a form of user account bypass, allowing malware to make system-wide changes without proper permissions or alerting the user to these changes. Windows defender was also disabled three times from the dataset, setting the value of “DisableAntiSpyware” to false, thwarting the user’s ability to remove or quarantine malicious applications, whilst it is important to note that this is now a legacy setting, explaining its conserved usage (Microsoft, 2022). Alongside this was “DisableTaskMgr”, with around 1% usage, disabling the user’s ability to terminate processes with high or unusual usage, commonly utilised by ransomware samples to avoid the user terminating any created processes.

The dataset contained six files that used hex stream to obfuscate changes made to the registry, shown in Figure 3.1.

Figure 3.1

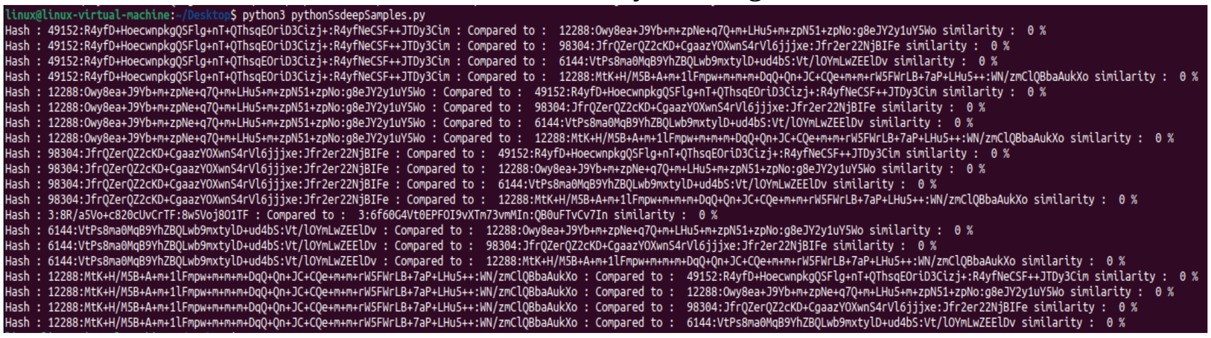
Ransomware Hex Stream Hashes

Time/Date	Hash
29/11/2019	1b3db282a06d549d7a16f2b2e2b4a69c29e110a0d094218d7c735ca013c84f45
29/11/2019	0d7964a2b78bdc1fabe5cdde85f238bc5306d57d6f1e7c1775924b0aea1a56f4
29/11/2019	0bdeebc278e53f93e220d030b88e9adb56fabe79ae615014c6b5357947a2acf4
29/11/2019	4d9c9e13e77dad4dd6d05f53c56109591f03ec4960678939f228aca9514ec99a
30/11/2019	00a671318b564a76e8306eb07828306473e10d17950fddad93ed9417143c34d1
27/01/2020	0b618e33d392e1e60ecdcc1e7183756f08e02add5d0a96e05318c34447b1a258

Whilst not an infallible means of evasion, used in collaboration with other techniques can provide an effective strategy to achieve evasion through obscurity. An example hex stream is “SystemCertificates\\trust\\HexStream = 430052004C007300", with the hex stream being ASCII encoded; whilst making it difficult to analyse the occurrences of hex streams manually, it does not guarantee evasion due to detection mechanisms such as signature-based detection. Interestingly, 5 of the six hashes share the same date. Whilst not an accurate indicator due to time stomping and obfuscation via packing, it could imply the generation of malware or how it was created.

Figure 3.2

SSDEEP Fuzzy Hashing



```
linux@linux-virtual-machine:~/Desktop$ python3 pythonSsdeepSamples.py
Hash : 49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim : Compared to :  12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo similarity :  0 %
Hash : 49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim : Compared to :  98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe similarity :  0 %
Hash : 49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim : Compared to :  6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv similarity :  0 %
Hash : 49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim : Compared to :  12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo similarity :  0 %
Hash : 12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo : Compared to :  49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim similarity :  0 %
Hash : 12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo : Compared to :  98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe similarity :  0 %
Hash : 12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo : Compared to :  6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv similarity :  0 %
Hash : 12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo : Compared to :  12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo similarity :  0 %
Hash : 98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe : Compared to :  49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim similarity :  0 %
Hash : 98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe : Compared to :  12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo similarity :  0 %
Hash : 98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe : Compared to :  6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv similarity :  0 %
Hash : 98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe : Compared to :  12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo similarity :  0 %
Hash : 3:8R/a5Vo+c820cUvCrTF:8w5Voj8O1TF : Compared to :  3:6f60G4Vt0EPFOI9vXTm73vmMIn:QB0uFTvCv7In similarity :  0 %
Hash : 6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv : Compared to :  12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo similarity :  0 %
Hash : 6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv : Compared to :  98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe similarity :  0 %
Hash : 6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv : Compared to :  12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo similarity :  0 %
Hash : 12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo : Compared to :  49152:R4yfD+HoecwnpkgQSFlg+nT+QThsqEOriD3Cizj+:R4yfNeCSF++JTDy3Cim similarity :  0 %
Hash : 12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo : Compared to :  12288:Owy8ea+J9Yb+m+zpNe+q7Q+m+LHu5+m+zpN51+zpNo:g8eJY2y1uY5Wo similarity :  0 %
Hash : 12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo : Compared to :  98304:JfrQZerQZ2cKD+CgaazYOXwnS4rVl6jjjxe:Jfr2er22NjBIFe similarity :  0 %
Hash : 12288:MtK+H/M5B+A+m+1lFmpw+m+m+m+DqQ+Qn+JC+CQe+m+m+rW5FWrLB+7aP+LHu5++:WN/zmClQBbaAukXo : Compared to :  6144:VtPs8ma0MqB9YhZBQLwb9mxtylD+ud4bS:Vt/lOYmLwZEElDv similarity :  0 %
```

Further investigation of the files involves using a handcrafted python script to compare the fuzzy hashes. Figure 3.2 expresses a similarity match of 0%, indicating that this is a trend amongst malware rather than a trend amongst variants. Less evasive manoeuvres, typically illustrated by ransomware samples, involve the disabling and overriding crucial system functionality, often used in conjunction with one another (Hamed et al., 2020). Figure 3.3 illustrates this. Typically the employment of one technique subsequently involved the usage of all of them. Whilst it can be effective, using these registry edits is noisy and was frequently applied amongst ransomware to prevent users from terminating the process.

Figure 3.3

Obscure Registry Edits

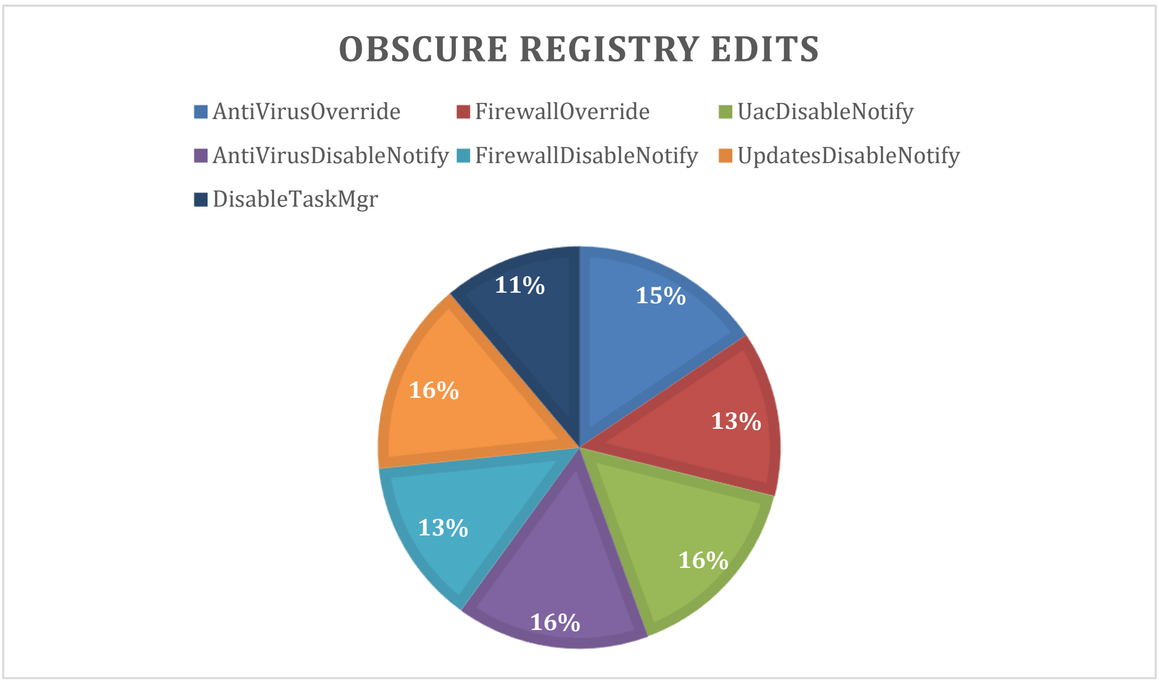


Figure 3.4

Registry Key Trends

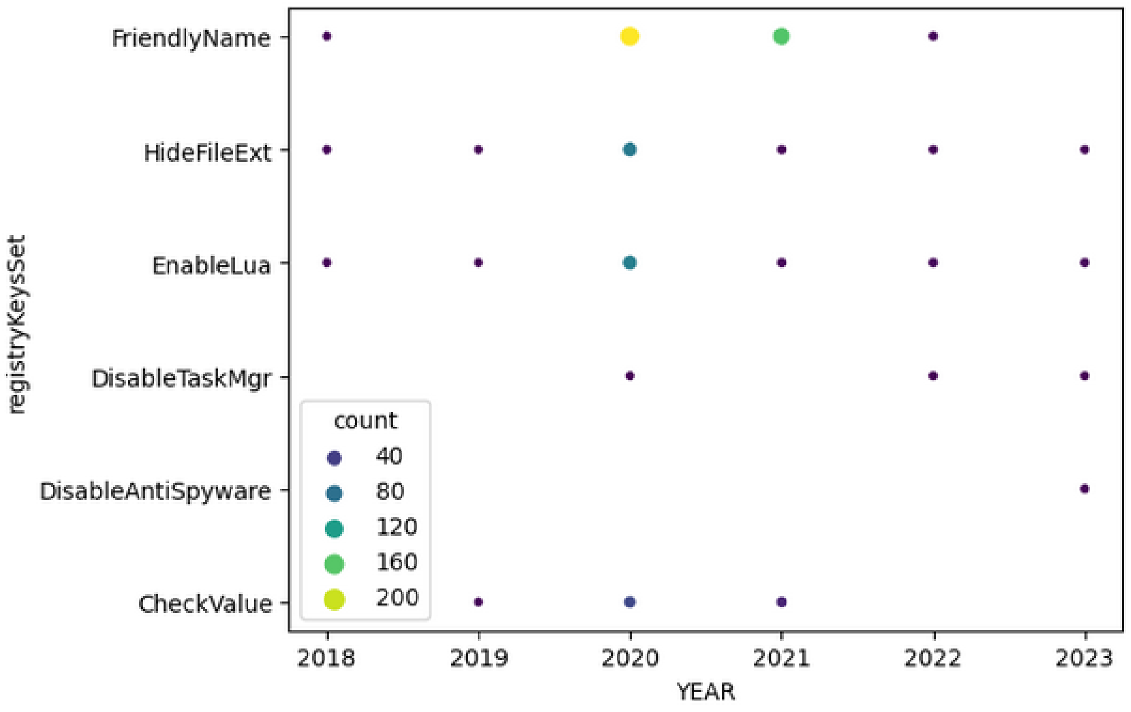


Figure 3.4 encapsulates the usage of registry keys and a given count for each year, with the two most consistently used being ‘HideFileExt’ and ‘EnableLua’ with continuous usage from 2018 through to 2023, highlighting their effectiveness and continued usage in aiding evasive malware. Opposing, ‘DisableAntiSpyware’ was the least set registry key, possibly indicating it as an emerging technique, reinforced by its only usage in 2023, whilst these edits primarily would focus on evading the user’s detection due to the noise generated.

6.6 Modules Loaded

A module is a set or unit of instructions used to aid an application in achieving its goal. In the context of malware, they provide a succinct overview of malware's capabilities whilst also aiding in the detection of variants as a result of the reused modules.

Figure 3.5

Modules Loaded Popular

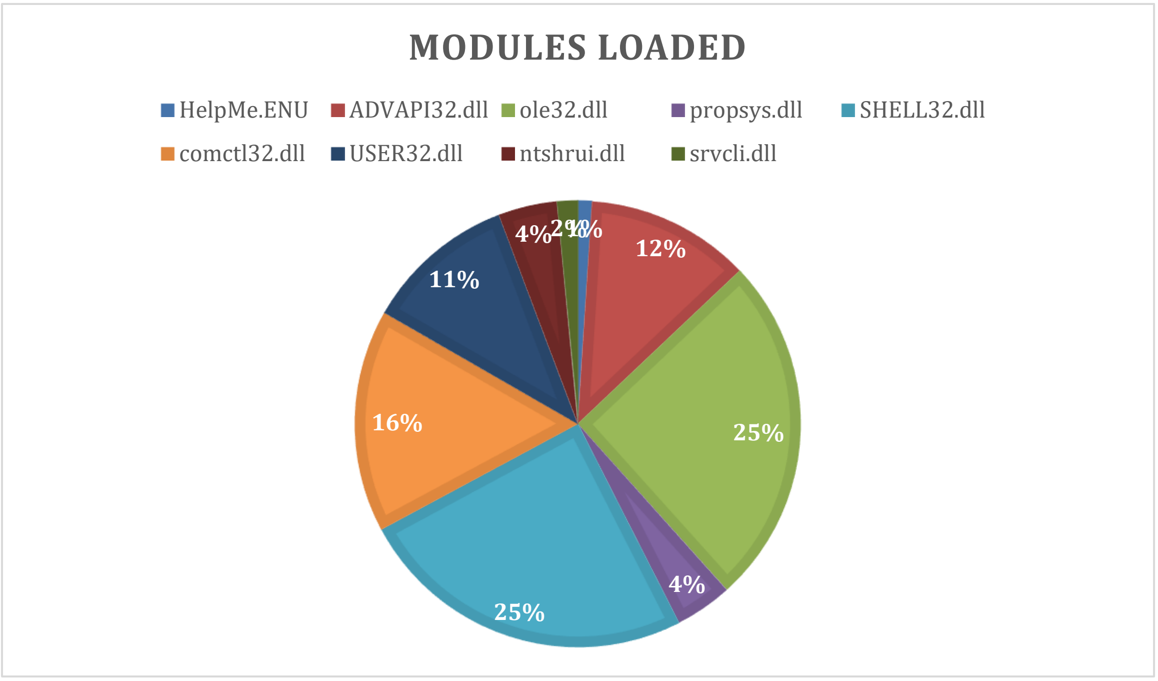


Figure 3.5 illustrates the most used modules within the dataset; it is hard to conclude them as many are used in common applications and are seemingly harmless. These modules do anything from interacting with the registry to managing user input and output operations; applications opt to use modules due to their efficiency as they are loaded at runtime. It could be inferred that DLL hijacking took place. The application loads malicious code within the DLL, replacing the name for a commonly used module such as ADVAPI32.dll, forcing the malicious code to be executed (Fernández-Álvarez & Rodrígu, 2023). Another possibility is the use of dynamic loading, its usefulness coming from applications dynamically loading these modules at runtime evading detection. This strategy requires local file system access on the target host and would likely be a more targeted attack (Kwon & Su, 2010).

Obscure modules loaded within the dataset contained random arbitrary string names shown in Figure 3.6. This could be indicative of dynamic loading occurring; dynamic loading becomes hard to detect as modules are loaded without writing them to a disk. This could bypass detection via traditional antivirus software, modern solutions such as machine learning and behavioural analysis would likely detect this.

Figure 3.6

Modules Loaded Obscure

modulesLoaded
"c:\\Windows\\system\\ykkoqjd.exe"'
'"c:\\Windows\\system\\uzuffyx.exe"'
'"c:\\Windows\\system\\eaukfcs.exe"'
'"c:\\Windows\\system\\vdqtadm.exe"'
'"c:\\Windows\\system\\fbfcpjd.exe"'
'"c:\\Windows\\system\\kxjzvsl.exe"'
'"c:\\Windows\\system\\khvwnkw.exe"'
'"c:\\Windows\\system\\ggfwbvk.exe"'
'"c:\\Windows\\system\\nghvooi.exe"'
'"c:\\Windows\\system\\uoejlgc.exe"'
'"c:\\Windows\\system\\amcyxmr.exe"'
'"c:\\Windows\\system\\tbddsox.exe"'
'"c:\\Windows\\system\\iegmrmz.exe"'

6.7 Processes Terminated

Similar to modules loaded, a malware's process reveals artefacts crucial to analysis and digital investigations. Much like any application, malware uses a process as an instance of an application, encapsulating key details including parent process, start time and general usage (Casey, 2009). Malware's manipulation of processes occurs through process injection or hollowing – creating a benign process in a suspended state, proceeding to unmap its memory (Leitch, 2013). This occurs through API calls such as 'ZwUnmapViewOfSection or 'NtUnmapViewOfSection' and is completed through the use of "VirtualAllocEx" or "WriteProcessMemory" (MITRE, 2020). It can be concluded that some form of process hollowing occurred due to the "VirtualAllocEx" presence within the dataset.

Figure 3.7

Process Terminated Popular

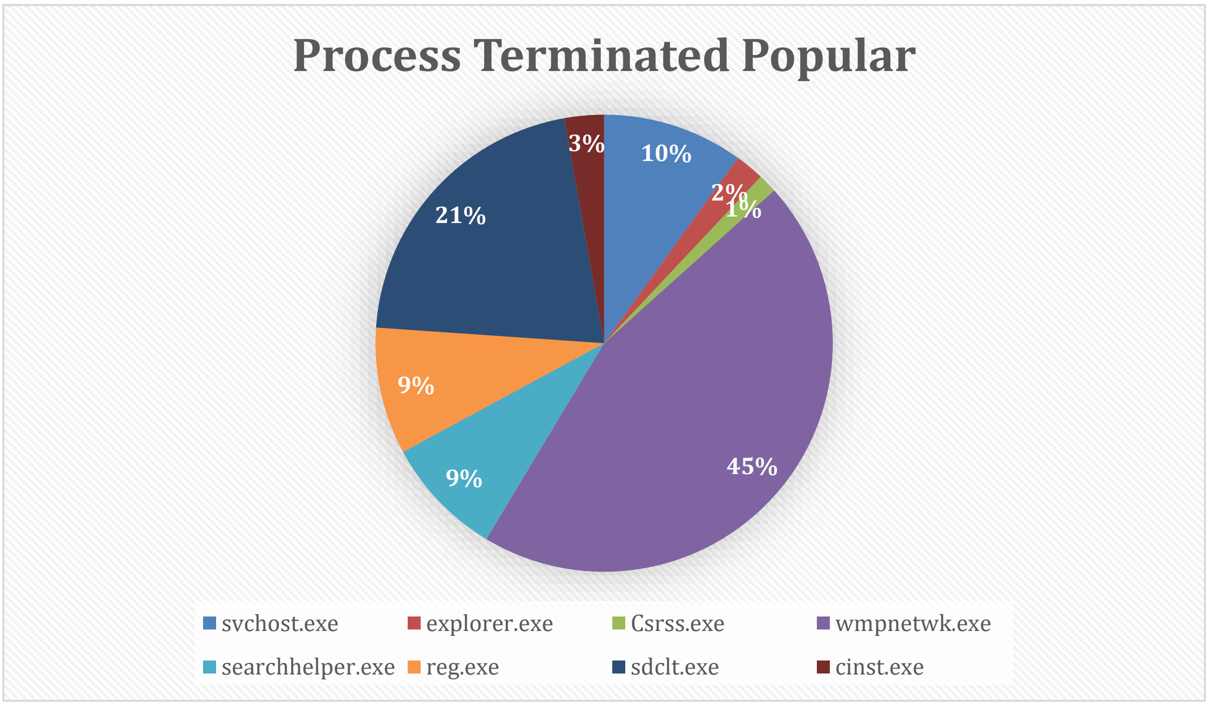


Figure 3.7 highlights the most commonly occurring processes terminated within the dataset, with the most common being “wmpnetwk.exe”, followed by “sdctl.exe”, with 45% and 21%, respectively. Most popular processes are Windows-related and, therefore, would likely be seen as harmless processes that were terminated. They could have been terminated to free up space, allowing the malware to run more efficiently with the additional resources. However, ‘VirtualAllocEx’ usage implies that some process hollowing took place. Statistics within Figure 3.7 could be used as a metric, generating the most popular process names malware uses for process hollowing to aid in machine learning and research. This would directly impact how malware is detected, used in conjunction with other key objects.

Figure 3.8

Obscure Processes Terminated

processTerminated
"""1428-C:\\Windows\\System\\WkbQGiY.exe""": 1
"""2332-C:\\Windows\\System\\IJYUOOi.exe""": 1
"""2432-C:\\Windows\\System\\rJImrwa.exe""": 1
"""2688-C:\\Windows\\System\\zecMXCK.exe""": 1
"""2412-\""C:\\Users\\admin\\Downloads\\d1ac8a4ccc8dee13a523b00525dccdaa.virus.exe\"""""": 1
"""2960-\""C:\\Users\\admin\\Downloads\\d697980b2fe1915c34db95dadf0704da.virus.exe\"""""": 1
"""%CONHOST%\""-541109544-821547718125166532033609812311500448-15025454501187004975992584004""": 1
"""2700-c:\\xnxxn.exe""": 1
"""2636-c:\\llplxnh.exe""": 1
"""2256-c:\\thrxxx.exe""": 1
"""1968-c:\\vrrxjn.exe""": 1
"""2880-c:\\lphpf.exe""": 1
"""800-c:\\dnhnd.exe""": 1

Similarly to Modules Loaded, processes terminated contained a list of seemingly random string names, likely terminating the loaded modules to avoid detection.

6.8 Conclusion

This chapter analysed results, opting to leave out analysis on IP addresses as the other objects focused on host-based evasion strategies, therefore outlining it in future work. Generally, the findings indicate malware is opting to evade sandboxing, most discovered techniques involved anti-sandboxing techniques. This is likely due to their prevalence within the industry, with other techniques focusing on evading user detection by hiding files or extensions. The general trend seems to be a rise in anti-sandboxing techniques, with some evidence of the generation of polymorphic strings within mutexes, modules loaded, and processes terminated, likely to evade signature-based detection. The two most valuable objects were calls highlighted and registry keys set, demonstrating an abundance of evasive strategies used by modern malware, with calls highlighted generally being the best metric for displaying these evasion strategies. The most challenging two metrics were modules loaded and processes terminated, with minimal evidence that any strategies had been used. However, it was hypothesised that some techniques could have been implemented based on other real-world samples. Evasion techniques like API Hooking, DLL Injection and Process hollowing had implied usage within the dataset; these techniques are known to be effective and still used in the wild and, in conjunction with each other, have an increased chance of achieving evasion.

Chapter 7: Conclusion

7.1 Introduction

This chapter aimed to discuss the project critically, review how the project achieved its goal, review the proposed research questions, discuss strengths and weaknesses of the project, identify areas of improvement, and highlighting any future work.

7.2 Limitations

The limitations of the project included an insufficient sample size for the analysis of trends – whilst the dataset itself was sizeable, the issue arose from its limited timeframe, with samples ranging from 2016 – 2023; a majority were taken from 2022, impacting how the analysis of trends was conducted. Therefore, impacting the results and outcome of the project. However, amendments could have been made without the limited timeframe; another dataset could have been accumulated from 5 years ago. This would have given a better indication of how malware evasion has changed and improved. Despite this, the project proved insightful, proving techniques currently used amongst modern malware samples.

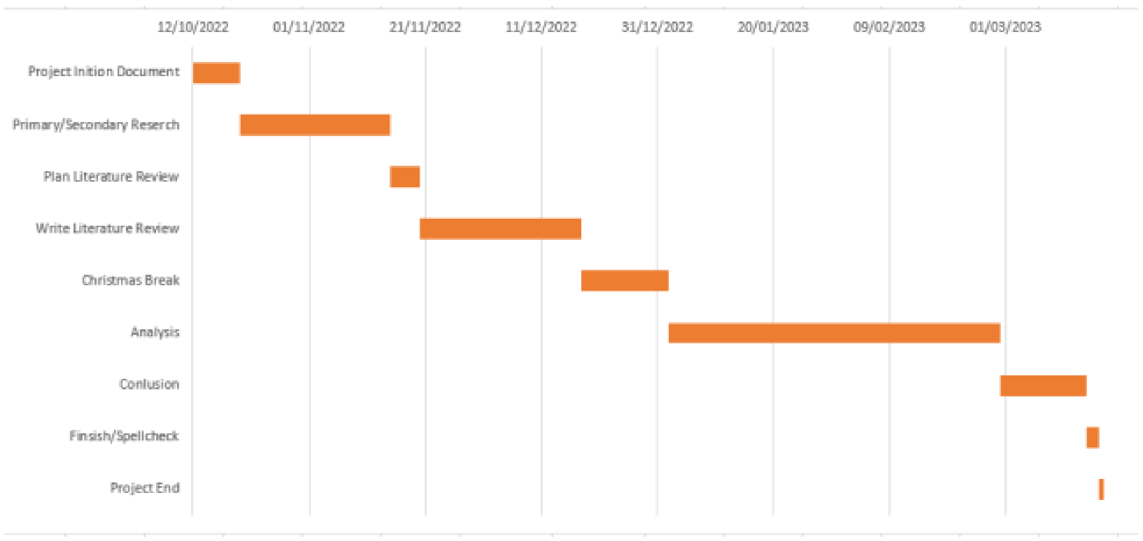
7.3 Future Work

VirusTotal's API encapsulates a series of objects; whilst this project hand-selected a few, a collection of file behaviours could be extracted from any given dataset, focusing on other malware features. A series of network-based behaviours could be analysed, such as 'ids_alerts', 'dest_ports', 'hostname', 'src_ip' and 'src_port' with the primary objective of analysing network-based evasion, associating a malicious IP address with the TOR consensus list, aiding in the identification of malicious relays through the use of traffic analysis. Alternatively, the same approach can be applied but with a dataset encapsulating an extensive timeframe, allowing the trends to be accurately portrayed.

7.4 Project Planning & Management

Figure 3.9

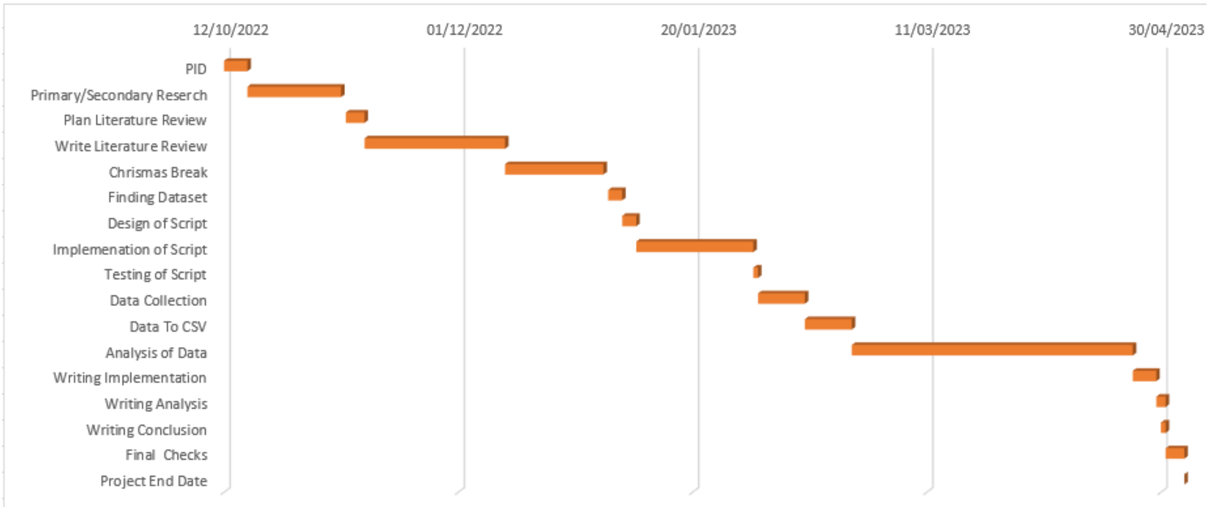
Project Gantt Chart



Commencement of the project involved the creation of a Gantt chart, displayed in Figure 3.9, alongside the critical path analysis discussed in Chapter 1. The project needed to adhere to this time scale due to other commitments and timeframes. It was closely followed, accompanied by implementation of RAD as a software methodology to ensure data collection was possible in the timeframe. Utilised project management methodologies were highly suitable because they allowed for a seamless and smooth transition from one chapter to the next, consequently meeting the deadlines for the first half of the project. However, deviations occurred due to a lack of understanding of the handling and parsing of JSON data, taking longer than initially thought. Despite this, the methodologies chosen were the favourable options, reinstating a new chart focusing on handling deviations illustrated in Figure 4.0.

Figure 4.0

Followed Gantt Chart



7.5 Review of Aims, Objectives & Requirements

Reinstating the project's aims of developing a framework for automated malware analysis and critically analysing malware evasion techniques permits the revision of these objectives, understanding if they were achieved. The project created a series of scripts allowing for the collection, storing and analysis of data, directly achieving the project‘s first objective, this was also a scalable, replicable and adaptable script capable of usage on other malware datasets, meeting the requirements outlined within the Design chapter. This directly contributed to the next objective, allowing for the critical analysis of malware evasion techniques through analysis of the produced artefact, finding currently circulating malware evasion techniques used in the wild dating as far back as 2016. In conclusion, the project provides an application capable of mass automatic malware analysis, completing the outlined aims and objectives; compiling an up-to-date revision of evasion techniques that were present within the "CIC-MalMem-2022" dataset, critically analysing techniques and how malware has changed over time.

7.6 Recommendations

For future use, it is recommended that amendments be made to the script, opting to use JSON parsing over the regex implementation that was used. Whilst it was effective and did work as a solution, it is not the recommended way of parsing data, and therefore revision is needed. Secondly, the use of another dataset is recommended, this one was great given the timeframe, but there is no limit on the number of samples that can be analysed other than disk space. Therefore, proving exceedingly useful to malware researchers.

7.7 Final Conclusion & Self-Reflection

Overall, the project was a challenging and rewarding process, with exciting learning opportunities, allowing Python to be learnt in more depth through the use of Seaborn and pandas, learning how to analyse data at mass, with 2,329 malware samples being analysed extracting objects based on their characteristics, and providing a useful application capable of mass malware analysis, with the opportunity to further problem-solving skills as the development proved difficult. To finalise, the project completed its aims to develop a framework for automated malware analysis and further built on this by analysing a dataset for evasive attributes.

References

Aslan, Ö. A., & Samet, R. S. (2020, January 3). A comprehensive review on malware detection approaches | IEEE journals IEEE Xplore. Retrieved January 27, 2023, from https://ieeexplore.ieee.org/abstract/document/8949524

ATT&CK, M. I. T. R. E. (2018, October 17). Defense evasion. Defense Evasion, Tactic TA0005 - Enterprise | MITRE ATT&CK®. Retrieved March 26, 2023, from https://attack.mitre.org/tactics/TA0005/

Barría, C. B., Cordero, D., Cubillos, C., & Palma, M. (2016, May 14). Proposed classification of malware, based on Obfuscation | ieee ... Proposed classification of malware, based on obfuscation. https://ieeexplore.ieee.org/abstract/document/7496735/

Barria, C., Cordero, D., Cubillos, C., & Osses, R. (2016, June 27). Obfuscation procedure based in dead code insertion into crypter. IEEE Xplore. https://ieeexplore.ieee.org/document/7496733/

Bhargavi, S., & Rai, C. R. (2022, July). Technology for Antiforensic Attacks. https://ijrpr.com/uploads/V3ISSUE7/IJRPR5947.pdf

Biondi, F., Given-Wilson, T., Legay, A., Puodzius, C., & Quilbeuf, J. (2018, January 1). Tutorial: An overview of malware detection and evasion techniques. SpringerLink. https://link.springer.com/chapter/10.1007/978-3-030-03418-4_34

Carrier, T., Victor, P., Tekeoglu, A., & Lashkari, A. H. (2022, February). [PDF] detecting obfuscated malware using memory feature engineering: Semantic scholar. [PDF] Detecting Obfuscated Malware using Memory Feature Engineering | Semantic Scholar.https://www.semanticscholar.org/paper/Detecting-Obfuscated-Malware-using-Memory-Feature-Carrier-Victor/97e4949d31244de2a085bde4b43eb3975851a155

Casey, E. (2009). Handbook of Digital Forensics and Investigation. ScienceDirect. https://www.sciencedirect.com/book/9780123742674/handbook-of-digital-forensics-and-investigation

Chen, L., Ye, Y., & Bourlai, T. (2017, December). Adversarial machine learning in malware detection: Arms race between. IEEE Xploree. https://ieeexplore.ieee.org/abstract/document/8240775

Chen, P., Huygens, C., Desmet, L., & Joosen, W. (2016, May 11). Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware. SpringerLink. https://link.springer.com/chapter/10.1007/978-3-319-33630-5_22

Chierzi, V., & Mercês, F. (2022, March 23). Evolution of IOT linux malware: A Mitre ATT&CK TTP based - IEEE xplore. IEEE Xplore https://ieeexplore.ieee.org/abstract/document/9738756/

Erdelyi, G. (2004, March 31). Hide'n'Seek? anatomy of stealth malware - blackhat.com. https://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-erdelyi/bh-eu-04-erdelyi-paper.pdf

Fernández-Álvarez, P., & Rodrígu, R. J. R. (2023, March). Module extraction and DLL hijacking detection via single or multiple memory dump. Elsevier Enhanced Reader. https://reader.elsevier.com/reader/sd/pii/S2666281723000069?token=BF240383CF9BBAE93AF8298898BEA15D6AF53D02C292F00622A4EEEAF608B3A1A2B8EA36310F68875BB3FC9F4ECF754C&originRegion=eu-west-1&originCreation=20230505070414

Galloroa, N. G., Polino, M., Carminatia, M., Continellab, A., & Zaneroa, S. (2021, November 21). A systematical and longitudinal study of evasive behaviors in Windows malware. Computers & Security. https://reader.elsevier.com/reader/sd/pii/S0167404821003746?token=DF5C476D891A056F41DEC585496F20C7E50F07B07EDBD93A2C76F2D9ACF9E08B5F61C57C8628D8BE9A292EF854FDA2CB&originRegion=eu-west-1&originCreation=20230128123208

Gao, Y., Lu, Z., & Luo, Y. (2015, January 15). Survey on malware anti-analysis | IEEE conference publication - IEEE xplore. IEEE Xplore. https://ieeexplore.ieee.org/document/7010353/

Golab, W., & Ramaraju, A. (2016, July 1). Recoverable mutual exclusion: Proceedings of the 2016 ACM symposium on principles of distributed computing. ACM Conferences. https://dl.acm.org/doi/pdf/10.1145/2933057.2933087

Hamed, Z., M. Ahmed, I., & Ameen, S. Y. (2020, January). Protecting Windows OS against local threats without using antivirus. ResearchGate. https://www.researchgate.net/profile/Ibrahim-Ahme27/publication/352996272_Protecting_Windows_OS_Against_Local_Threats_Without_Using_Antivirus/links/60e31b64a6fdccb74507b61a/Protecting-Windows-OS-Against-Local-Threats-Without-Using-Antivirus.pdf

Jeong, T., Lee, S., Youn, J., & Cho, D. (2021, February 28). Effective feature selection for classification of malware families. Journal of Convergent Research Interchange. http://www.fucos.or.kr/journal/APJCRI/Articles/v7n2/8.pdf

Kim, D., Majlesi-Kupaei, A., Roy, J., Anand, K., ElWazeer, K., Buettner, D., & Barua, R. (2017, September 29). DynODet: Detecting Dynamic Obfuscation in malware. SpringerLink. https://link.springer.com/chapter/10.1007/978-3-319-60876-1_5

Kirat, D., & Vigna, G. (2015). Malgene: Automatic extraction of malware analysis evasion signature. dl.acm. https://dl.acm.org/doi/pdf/10.1145/2810103.2813642

Kirat, D., & Vigna, G. (2015, October 1). Malgene: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM Conferences. https://dl.acm.org/doi/10.1145/2810103.2813642

Kramer, S., & Bradfield, J. C. (2009, September 29). A general definition of malware - journal of computer virology and hacking techniques. https://link.springer.com/article/10.1007/s11416-009-0137-1

Kwon, T., & Su, Z. (2010, July 1). Automatic detection of unsafe component loadings: Proceedings of the 19th International Symposium on Software Testing and Analysis. ACM Conferences. https://dl.acm.org/doi/pdf/10.1145/1831708.1831722

Ladutska, R. (2022, December 20). Evasions. Github. https://github.com/CheckPointSW/Evasions

Leitch, J. (2013). Process Hollowing. https://dl.packetstormsecurity.net/papers/general/Process-Hollowing.pdf

Li , J., Xu, M., Zheng, N., & Xu , J. (2009, November). Malware obfuscation detection via maximal patterns. IEEE Xplore. https://ieeexplore.ieee.org/document/5369393/

Lu, H., Wang, X., Zhao, B., Wang, F., & Su, J. (2013, March 25). Endmal: An anti-obfuscation and collaborative malware detection system using Syscall sequences. ScienceDirect. https://www.sciencedirect.com/science/article/pii/S0895717713001064

Marpaung, J. M. A. P., Sain, M., & Lee, H.-J. (2012, April 3). Survey on malware evasion techniques: State of the art and challenges. IEEE Xplore. https://ieeexplore.ieee.org/document/6174775/

Matin, I. M. M., & Rahardjo, B. (2020, January 23). Malware detection using Honeypot and machine learning. IEEE Xplore. https://ieeexplore.ieee.org/document/8965419

Mezina, A., & Burget, R. (2022, November 18). Obfuscated malware detection using dilated Convolutional Network |. IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9943443/

Microsoft. (2022, March 18). DisableAntiSpyware. Microsoft Learn. https://learn.microsoft.com/en-us/Windows-hardware/customize/desktop/unattend/security-malware-Windows-defender-disableantispyware

N. Khasawneh, K., Abu-Ghazaleh, N., Ponomarev, D., & Yu, L. (2017, October). RHMD: Evasion-resilient hardware malware detectors. ACM DL DIGITAL LIRARY. https://ieeexplore.ieee.org/document/8686636/

Narouei, M., Ahmadi, M., Giacinto, G., Takabi, H., & Sami, A. (2015, April 22). DLLMiner: Structural Mining for Malware detection. Wiley Online Libraryfrom https://onlinelibrary.wiley.com/doi/pdfdirect/10.1002/sec.1255

Noor, M., Abbas, H., & Shahid, W. B. (2018, February 1). Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. Journal of Network and Computer Applications. https://www.sciencedirect.com/science/article/pii/S1084804517303168

Nunes, M., Burnap, P., Rana, O., Reinecke, P., & Lloyd, K. (2019, August 20). Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis. Journal of Information Security and Applications. https://www.sciencedirect.com/science/article/pii/S2214212619300109

Nunes, M., Burnap, P., Reinecke, P., & Lloyd, K. (2022, May 18). Bane or boon: Measuring the effect of evasive malware on system call classifiers. Journal of Information Security and Applications. https://www.sciencedirect.com/science/article/pii/S2214212622000813

O'Kane, P. O. K., Sezar, S., & McLaughlin, K. (2011). Obfuscation: The hidden malware | IEEE Journals & Magazine - IEEE Xplore. IEEE Xplore. https://ieeexplore.ieee.org/document/5975134/

Olaimat, M. N. O. N., Maarof, M., & Al-rimy, B. A. S. A.-rimy. (2021, January). Ransomware anti-analysis and evasion techniques: A survey and research IEEE Xplore, https://ieeexplore.ieee.org/document/9392529

Oyama, Y. (2018, May 6). Investigation of the diverse sleep behavior of malware - 日本郵便. https://www.jstage.jst.go.jp/article/ipsjjip/26/0/26_461/_pdf/-char/en

Oyama, Y. (2019, January 24). Skipping Sleeps in Dynamic Analysis of Multithreaded Malware. Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8625167&tag=1

Process injection: Process hollowing. Enterprise | MITRE ATT&CK®. (2020, January 14). https://attack.mitre.org/techniques/T1055/012/

Rad, B. B., Masrom, M., & Ibrahim, S. (2012). (PDF) camouflage in malware: From encryption to metamorphism.ResearchGate. https://www.researchgate.net/publication/235641122_Camouflage_In_Malware_From_Encryption_To_Metamorphism

Razak, M. F. A., Anuar, N. B. B., Salleh, R., & Firdaus, A. (2016, August 26). The rise of "Malware": Bibliometric Analysis of Malware Study. Journal of Network and Computer Applications. https://www.sciencedirect.com/science/article/pii/S1084804516301904

Rin, N. (2013, November). VMDE. https://www.heise.de/security/downloads/07/1/1/8/3/5/5/9/vmde.pdf

Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., & Weippl, E. (2017, March 1). Protecting software through obfuscation: Can it keep pace with progress in code analysis?: ACM Computing Surveys: Vol 49, no 1. ACM Computing Surveys. https://dl.acm.org/doi/abs/10.1145/2886012

Sihwail, R., Omar, K., & Ariffin, K. A. Z. (2018, September). Vol.8 (2018) No. 4-2 ISSN: 2088-5334A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. Faculty of Information Science & Technology. https://core.ac.uk/download/pdf/325990564.pdf

Sikorski, M., Honig, A., Zhuge, J., Jiang, H., & Zhang, G. (2012, March 3). E Yi Dai Ma Fen Xi Shi Zhan: The hands-on guide to dissecting malicious software = practical malware analysis. Google Books. https://www.amazon.com/Practical-Malware-Analysis-Hands-Dissecting/dp/1593272901

Stiborek, J., Pevný Tomáš, & Rehák, M. (2017, October 16). Multiple instance learning for malware classification. Expert Systems with Applications. https://www.sciencedirect.com/science/article/pii/S0957417417307170

Tahir, R. (2018, March). A study on malware and malware detection techniques - MECS press. MECS. Retrieved January 20, 2023, from https://mecs-press.org/ijeme/ijeme-v8-n2/IJEME-V8-N2-3.pdf

Talukder, A., Khondokar, F. H., Islam, M., Uddin, A., Arnisha, A., Mohammad , A. Y., Fares, A., & Ali, M. M. (2022, December 23). A dependable hybrid machine learning model for network intrusion detection. Journal of Information Security and Applications. https://www.sciencedirect.com/science/article/pii/S2214212622002496

Veerappan, C. S., Keong, P. L. K., Tang, Z., & Tan, F. (2018, February 8). Taxonomy on malware evasion countermeasures techniques | IEEE ... IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8355202/

Wartell, R. (2015, October 6). Ticked off: Upatre Malware's simple anti-analysis trick to defeat sandboxes. Unit 42. https://unit42.paloaltonetworks.com/ticked-off-upatre-malwares-simple-anti-analysis-trick-to-defeat-sandboxes/

Willems , C., Holz , T., & Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. University of Mannheim. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4140988&tag=1

You, I., & Yim, K. (2010, November 11). Malware obfuscation techniques: A brief survey | IEEE conference ... IEEE Xplore. https://ieeexplore.ieee.org/document/5633410

Carrier, T., Victor, P., Tekeoglu, A., & Lashkari, A. H. (2022). Detecting Obfuscated Malware using Memory Feature Engineering. In The 8th International Conference on Information Systems Security and Privacy (ICISSP).

VirusTotal. (n.d.). https://www.virustotal.com/

Appendices

Appendix A – PID

School of Computing
Final Year Research Project

Project Initiation Document

Ryan Smallcalder

Malware Evasion Techniques in the last 10 years

1. Basic details

Student name:	Ryan Smallcalder
Draft project title:	Malware Evasion Techniques in the last 10 years
Course and year:	Cyber Security and Forensic Computing
Project supervisor:	Tobi Fajana
Client organisation:	
Client contact name:	

2. Degree suitability

This project will be highly relevant to the industry, as it involves static/dynamic analysis of malware, particularly the techniques that are used in the evasiveness of malware, uncovering the most commonly used techniques, and more advanced techniques. Within the modern-day malware accounts for a large percentage of attacks; in 2021 there were 5.4 billion record attacks. This project can therefore aim to provide insight into antivirus software, finding common evasion techniques alongside the more advanced techniques. Malware makes up a large portion of cybersecurity and is one of the most important to defend against. This project directly links to the Malware forensics module and is therefore highly suitable for study within my degree.

3. Outline the project environment and problem to be solved

*For **research** or **study** projects:*
The intended audience would be malware analysts, and applied to malware vendors, as new samples are found every day, it would be beneficial to recognize more obscure techniques. The project would aim to provide insight into malware evasion techniques that the industry will find useful, whilst many scholarly examples exist involving the examination of malware - many do not discuss and review the techniques that are utilised by the malware to avoid detection. Therefore this project will attempt to present the evolution of malware techniques, including the most used techniques, and more obscure/advanced ones, and how they have changed over the last 10 years.

4. Project aim and objectives

Malware remains a prevalent part of cyber security and still poses a major threat to the internet, with antivirus companies dealing with unique malware samples every day. The project aims to analyse malware evasion techniques, seeing the progression of the evasion techniques over time, seeing what has changed, or if older techniques remain viable in the evasion of antivirus software. The project can be broken down into crucial stages that will allow me to meet the project's aim.

1. Revision of malware evasion techniques/tactics
2. Writing the Literature review
3. Creation/Obtaining a data set
4. Analysis of data set/samples

This project will use limited resources as most of it is already pre-owned there will be minimal constraints. However, conditions that will hinder my project include dissatisfactory datasets, harsh timescales, illness to myself or staff, and inability to access staff facilities such as the labs or the university library.

5. Project aim and objectives

Malware remains a prevalent part of cyber security and still poses a major threat to the internet, with antivirus companies dealing with unique malware samples every day. The project aims to analyse malware evasion techniques, seeing the progression of the evasion techniques over time, seeing what has changed, or if older techniques remain viable in the evasion of antivirus software. The project can be broken down into crucial stages that will allow me to meet the project's aim.

5. Revision of malware evasion techniques/tactics
6. Writing the Literature review
7. Creation/Obtaining a data set
8. Analysis of data set/samples

This project will use limited resources as most of it is already pre-owned there will be minimal constraints. However, conditions that will hinder my project include dissatisfactory datasets, harsh timescales, illness to myself or staff, and inability to access staff facilities such as the labs or the university library.

6. Log of risks

Description	Impact	Mitigation/Avoidance
COVID-19 outbreak means I cannot get into a lab for usability testing	*Meidum - cause delays to the project timeline*	*Get in while I can and prioritize lab tasks in time. Make an alternate test plan that does not need the lab.*
Other time constraints such as assignment deadlines	*Low - Would need to delay the project for other time constraints*	*Proper time management to mitigate deadline collisions*
Hardware failure	*Low - Will delay, with the potential to make data corrupted.*	*Frequent backups to all data used*
Malware attack	*Medium - Could result in corrupted data/ locked data*	*Frequent backups to all data used* *Frequent anti-virus scans after each analysis.* *Use virtual machine when analyzing malware*
Lack of knowledge in subject area	*Medium - Delayed time scale*	*Attend lectures* *Frequent library visits* *Frequent meetings/emails to fill gaps in knowledge.*
Power Outage	*Low - Potential to corrupt data/loss of data* *Unable to access documents.* *Delays to project time scale.*	*USB drive containing updated data* *Cloud solution such as google docs* *Work in the Library/Lab facilities.*
Low bandwidth	*Low - Delays to project time scale.*	*Work in the Library/Lab facilities.*
Internet Outage	*Low - Delays projecting time scale.*	*Work in the Library/Lab facilities.*
Supervisor absence	*Low - Delays to project. Would have a higher impact depending on the time of the year*	*on Weekly meetings to stay on top of the project, with an opportunity to exceed the project timeline.*
illness/injury to me	*Medium - Delays to project. Would have a higher impact depending on the time of the year*	*Weekly meetings to stay on top of the project, with an opportunity to exceed the project timeline.*

7. Project deliverables

A dataset will need to be created/found on popular malware within the last 10 years, a number will need to be picked on how many samples this will be, and a justification of why x amount of samples was picked. The creation of other documents includes a plan for the literature review, an excel spreadsheet containing the name of the malware, the hash value, the source it was obtained from, and any other additional information. The plan for the literature review will only contain academic literature that will add depth to the final version of my dissertation.

8. Project approach

The project has deadlines that can often be classified into different project stages. These deadlines will be managed using a planner such as a google calendar, this is a suitable solution as it syncs lectures/events and will allow me to see what deadlines are coming up to avoid missing deadlines for other assignments. I will need to ensure that I am managing my time correctly as I have a part-time job and other modules, to ensure that I will set daily goals to complete, such as a daily quota of samples that need to be analysed. I will use critical path analysis to methodically break down my project into essential tasks, this will ensure that the project is done systematically without missing critical steps. The start of my project will involve researching secondary sources, such as existing reports journals and documents. The project may use primary research in the creation of a dataset of malware samples. It could also change direction and use a pre-made dataset for analysis.

Project plan

The project can be broken down into stages outlined below:

Primary/Secondary Research

- ○ Involves the creation/collection of a dataset.
- ○ Collection of literature
  - ■ What has been done before
  - ■ Identify gaps; what have they done well/what's missing
    - ● Timeline: Latest 15th November
  - ■ Resources Used
    - ● Internet, Library
    - ● No other access needed

Plan of Literature review

- ○ Completion of literature review planner
  - ■ Writing of literature review

                - Timeline: Before Christmas Break

Collection/Creation of samples

- Will run in the background of other key project events
- Timeline will generally vary depending on how quickly samples are analysed.
- A number of samples will be analysed and put into an excel document to keep track of them.
- Timelines can be much quicker if a usable dataset is found
    - Resources Used:
        - Internet, Library, Github, Virtual Machines
        - Malware analysis skills needed
            - Will be obtained from frequent lectures (Malware Forensics Module)

Analysis

- Written when x number of malware has been analysed
    - Timeline: After Christmas break - End of February

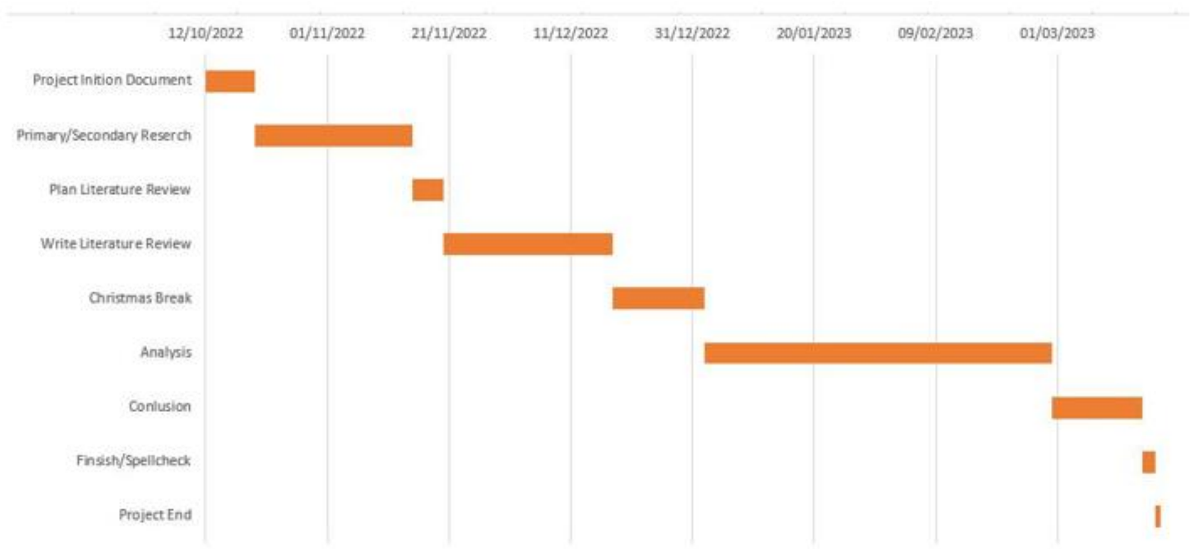
Conclusion/Introduction

- Written after Analysis to ensure key points are mentioned
    - Timeline: End of February - 15th March

Finishing write up

- Envolve cleaning up document
- Any spelling mistakes/referencing issues or

incorrect data. This is further illustrated within this Gantt chart



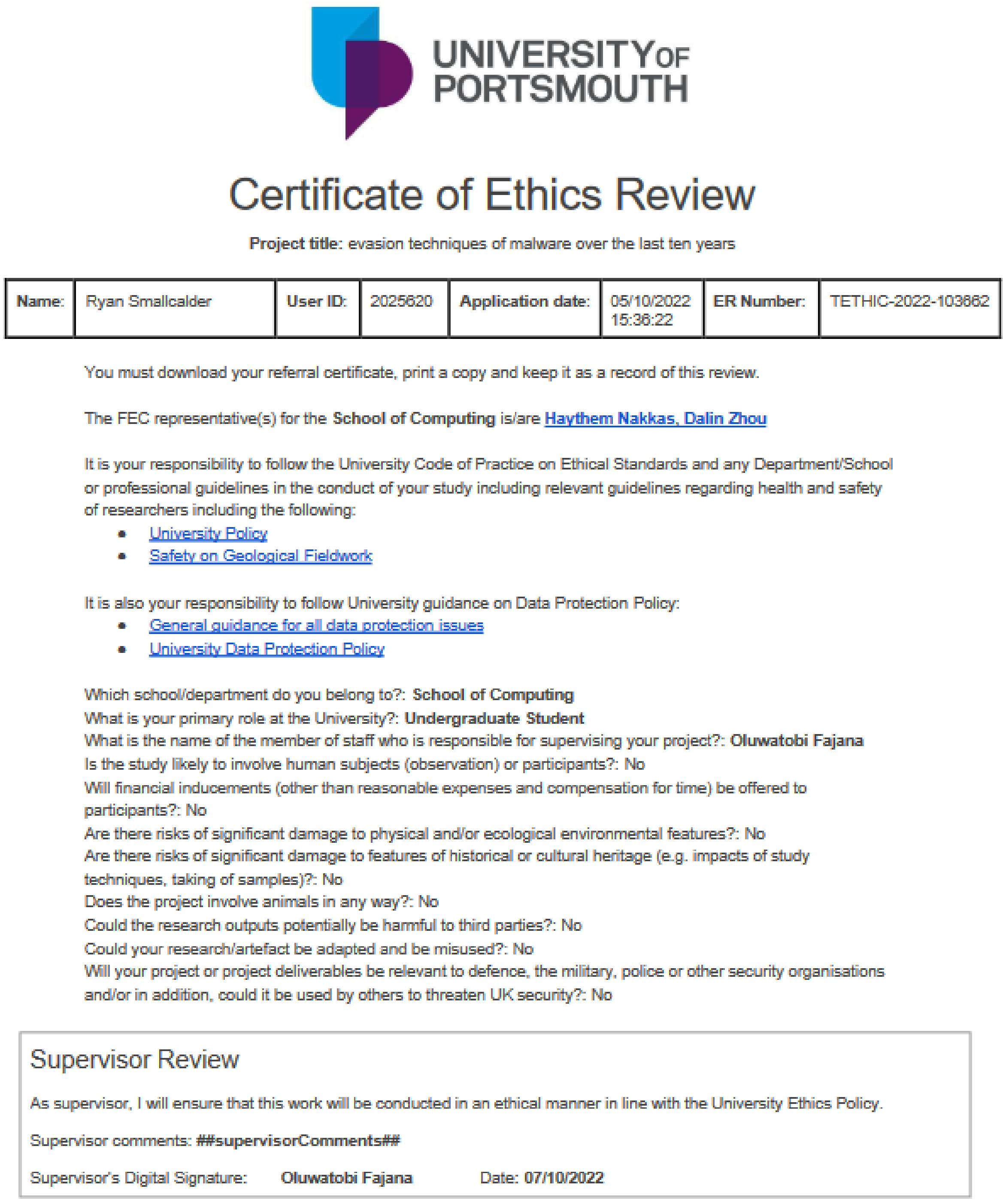
9. Supervisor Meetings

I have agreed with my supervisor to meet every two weeks, this can be either face-to-face or online depending on his availability, I will also email him whenever I can to make sure I am on track. To combat any extended absences of my supervisor, I aim to keep ahead of schedule and will use my Gantt chart as a rough outline, the timeline will be very important to follow as I work weekends and therefore need to manage my time effectively.

10. Legal, ethical, professional, and social issues (mandatory)

Malware would require correct handling, and preservation to ensure that the malware doesn't infect the host system. I would be responsible for the widespread infection that the malware causes, I have taken steps to ensure that the malware is handled correctly. A virtual machine will be used to download the malware, the malware can be renamed to “sample.exe.vir” to prevent accidental execution of the program. Next, it will be put into the virus total, to detect if it is a know malicious program. Antivirus will need to be disabled to prevent the deletion of the malware. The virtual machine will also be regularly snapshotted to prevent it from being unusable, this will prevent any time from being wasted on creating a new virtual machine after every analysis. The first step will be to research how to build a safe environment and this will be included in the malware forensics module. Therefore the main security implication would be an infection into the host/network which can be mitigated by the previous steps. The main law that handles malware is the Computer misuse act as I own the computer, it only becomes an issue if I am unable to correctly handle the malware when analysing it. I will only be infringing on the law when the malware infects systems that I don't own, for example, if I need to work in the library. I will only be analysing malware on my home computer to prevent this. This will guarantee the malware is handled in a safe environment.

Appendix B – Certificate of Ethics Review



UNIVERSITY OF PORTSMOUTH

Certificate of Ethics Review

Project title: evasion techniques of malware over the last ten years

Name:	Ryan Smallcalder	**User ID:**	2025620	**Application date:**	05/10/2022 15:36:22	**ER Number:**	TETHIC-2022-103662

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Haythem Nakkas, Dalin Zhou**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Oluwatobi Fajana**
Is the study likely to involve human subjects (observation) or participants?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor comments: **##supervisorComments##**

Supervisor's Digital Signature: **Oluwatobi Fajana** Date: **07/10/2022**

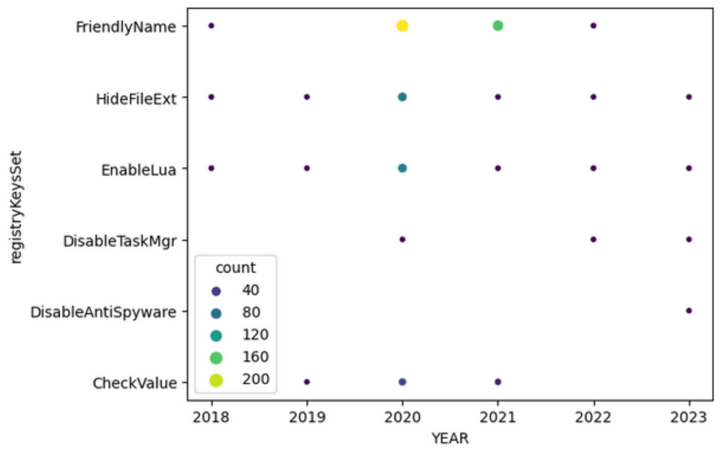
Appendix C – Excel Spreadsheet Plan

Hash	Module Loaded	CallsHighlighed	IP_ADDR	ProcessTerminated
Hash 1	data	data	data	data
Hash 2	data	data	data	data

Appendix D – Seaborn Trends Code Using Jupyter

```
In [48]: df = pd.read_csv('C:/Users/Rsmal/Desktop/DissFinal/ScriptsData/Dissertation/Registry.csv')
         dataComp = df[['YEAR', 'registryKeysSet', 'count']]
         colour = sns.color_palette("viridis", as_cmap=True)
         sns.scatterplot(data=dataComp, x='YEAR', y='registryKeysSet', hue="count", size = "count", palette=colour)

Out[48]: <Axes: xlabel='YEAR', ylabel='registryKeysSet'>
```


FriendlyName
HideFileExt
EnableLua
DisableTaskMgr
DisableAntiSpyware
CheckValue
registryKeysSet
count
YEAR