UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

RanAware, analysis and detection of ransomware on Windows systems

Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

Written By: **Pablo Aznar Puyalto**

Under supervision of: **René Serral Gracià**

Barcelona, June 2022

Abstract

These past years the use of the computers increased significantly with the introduction of the home office policy caused by the pandemic. This grow has been accompanied by malware attacks and ransomware in particular. Therefore, it is mandatory to have a system able to protect, to prevent and to reduce the impact that this type of malware has in an organization.

RanAware is a tool that performs an early ransomware detection based on recording file system operations. This information allows RanAware to monitor activity on the file system, collect and process statistics used to determine the presence of a ransomware in the system. After detection, RanAware handles the termination and isolation of the malicious program as well as the creation of an activity report of the ransomware operations.

In addition, this project performs an evaluation of the impact that RanAware has in a system.

Acknowledgements

I would like to thank several people for this project. First of all, *René Serral,* the director of this project, for guiding, counselling and supporting me during the development of this project.

Secondly, I would like to thank my colleague from the masters, *Rubén Barceló Armada*, for taking time to discuss and give his opinion regarding some of my questions about this project.

Thirdly, to my brother *José Manuel Aznar Puyalto,* for supporting and helping me to take decisions regarding the project and for taking his time review the report contents and structure.

Fourthly, to my mother *Maria Dolores Puyalto Gracia,* for supporting me throughout the development of the project and helping me with all the design related decisions regarding the project report and presentation.

Lastly, to *Daniela Leila Pernas Ferrari* for designing the RanAware logo.

Table of contents

List of Figures

List of Tables

1. Introduction

Due to the COVID-19 pandemic people and companies have become more reliant on computers and online business; thus, leading to a drastic increase of ransomware attacks and other types of malware in general [1].

Ransomware is a class of malware that aims to block or deny the access to a victim's machine and asks for a ransom payment to recover the system. Ransomware attacks can be distinguished in two categories, locker ransomware and crypto ransomware. While the former focuses on locking the access and disabling some services on the victim's machine, the second is characterized by encrypting the files on a device, as a result immobilizing or blocking a system that remains powerless without access to the decryption key [1].

Crypto ransomware uses secure and standard cryptographic algorithms, designed to be reliable and unbreakable, to encrypt the data. This makes them especially dangerous since the recovery of the encrypted data is supposedly impossible without the knowledge of the decryption key. For this reason, crypto ransomware is considered to be more threatening than locker ransomware, since even though the system is available, the information may be lost forever. This is also the reason why crypto ransomware are more used than locker ransomware.

The goal of the cybercriminals that use ransomware is usually the financial gain. For this reason, cybercriminals tend to target big companies or public institutions, as these are more likely to pay a high ransom. However, individuals and small companies are also victims of a ransomware attacks. Therefore, every user should be protected against them.

In this past year, 2021, there has been some noticeable Ransomware attacks, for example; the attack against Colonial Pipelines that took place in May 2021, affected the operations of a major fuel supply chain in the United States of America. The company had to pay around $4.4 million dollars. Another attack that happened in 2021 was the attack performed by the ransomware group PYSA against the Universitat Autònoma de Barcelona (UAB). Even when the public entity did not pay the ransom, the estimated cost to restore the system and functionalities is around €2 million euro. In addition, the consequences of this attack are still affecting the university and its students since the system is not completely functional as it was before.

The cost of being a ransomware victim is not only the economic ransom that is asked to free the system, that often is not paid, but the impact it has on the normal availability and operability of the victim. Usually, a company affected by ransomware will have a long period of downtime until the system is completely recovered that implies an economical loss, often comparable to the amount that is being asked by the cybercriminals.

For these reasons it is crucial to be protected against ransomware attacks, the most optimal protection is an early detection to be able to identify and stop the ransomware before it is able to encrypt the whole system, hence minimizing the loss of the victim.

1.1. Project Goal

The aim of this project, RanAware, is to build a lightweight system that is able to perform an early detection of crypto ransomware and minimise the potential damage that can cause into a system.

This project focuses on crypto Ransomware since they are the most common and critical type of ransomware nowadays. The target system is Windows, because is the most used platform in computers and the one that is mostly targeted by ransomware attacks.

RanAware detects ransomware using different methods; first, by strategically placing and monitoring decoy files, which are specially crafted files that should never be written into, to detect malicious processes trying to modify them. Second, RanAware is able to determine potential malicious processes by monitoring them along with the file system operations that happen in the system. Finally, RanAware stops malicious processes, provides a log of what are the operations performed by the terminated suspicious processes and ensures that these are never executed again in the system.

1.2. Motivation

The damage caused by ransomware attacks in an organization or an individual is incredibly high, not only because it encrypts all the personal or important data, that will be possibly lost, but also because the downtime needed to recover from these attacks is usually as damaging as the loss of data.

During 2019 the computational infrastructure of my father's small business was affected by Ransomware twice. Having seen first-hand the potential damage of ransomware attacks to individuals or small business paired with the increase of ransomware cases during the pandemic motivated me to investigate and develop a system to help anyone that can be affected by ransomware, big companies or individuals.

After researching on the topic, I found that the majority of the work uses Machine Learning as a tool to implement an intelligent system to early detect ransomware based on their behaviour. Despite the good performance that a Machine Learning system provides, the cost of creating and maintaining an algorithm cannot be overlooked. For this reason, RanAware focuses on a simpler approach for an early ransomware detection system, since I believe that ransomware characteristic behaviour can be detected by other means, avoiding the costs of the machine learning algorithm implementation and maintenance.

1.3. Contents

This thesis is divided in several sections as follows: First, I will comment on some proposed products to detect and prevent ransomware. Second, I will expand on the common Ransomware functionalities as well as some Windows system mechanisms. Third, I will explain and develop the work done in this thesis. In the following chapter, I will present the results of the thesis, first the results testing against diverse ransomware families and later an evaluation of the overhead and the cost it implies using this product on a system. Fifth, I will comment on the project planification followed by the budget for the project. Then I will focus on the improvements and future work continued with work that was developed but later discarded. Finishing with the conclusions of the project.

2. Related Work

Ransomware attacks have been growing over the years and so has the research in the academic community. Many studies that investigate the characteristics and behaviour of ransomware [2]–[4] and use them to propose techniques in order to detect them have been presented [5]–[10]. In addition, several survey papers have been redacted to summarize the research endeavours and to provide a general view of the ransomware scene [1], [11]. Moreover, some papers proposing products to early detect ransomware and to prevent and mitigate the damage have been published [12], [13].

In this section I will focus on commenting two papers that propose different systems to detect ransomware as early as possible to mitigate the damage done.

2.1. ShieldFS

ShieldFS is an extension of the filesystem that is able to distinguish ransomware from benign processes during runtime [12].

ShieldFS authors have created a machine learning classifier based on the low-level windows operations, IRPs, that is able to distinguish between a benign process and a ransomware based on the file-system operations.

In addition, ShieldFS is able to work with ransomware that use code injection to mask their operations by maintaining a history of the operations of the processes.

ShieldFS looks for indicators of the use of cryptographic primitives scanning the memory of any process identified as malicious. ShieldFS also proposes a shadowing mechanism to recover the files encrypted by a ransomware.

2.2. RWGuard

RWGuard is a product that employs three different monitoring techniques to detect ransomware at an early stage [13].

The first technique that RWGuard uses are decoy files, which are files that should never be written, placed strategically around the system. The second technique used is monitoring the file operations of the processes at IRP level, RWGuard has a machine learning classifier able to predict if the process is benign or a ransomware based on this data. The third technique consists of monitoring file changes (e.g., create, delete and write operations) to determine anomalous changes.

Furthermore, RWGuard performs library hooking of the CryptoAPI of Windows in order to intercept and monitor all file encryptions. The objective is to store the keys used for encryption by malicious processes in order to recover the encrypted files after a ransomware attack.

RWGuard also implements a file classification mechanism that, after receiving the decision taken by the process monitoring classifier and the file monitoring module, uses the information obtained with the crypto-tool to determine if it is an operation performed by a Ransomware or a legitimate process.

The main difference of RanAware with the rest of the projects found in the research field is that RanAware does not use Machine Learning to detect ransomware processes. It combines state-of-the-art techniques with its own process monitoring strategy to infer the presence of ransomware. Additionally, RanAware implements a new system to ensure that these malicious processes are not executed again in the victim’s computer.

3. Background

In the next sections of the thesis, I will explain in some extent the knowledge required to better understand the work performed in this thesis and the decisions made throughout the development of the project.

I will start expanding on the definition of ransomware and their most common actions. Following this the Input/Output (I/O) model of Windows will be discussed since it is the main focus of RanAware. Finally, the file system filter drivers will be explained since it is important for the understanding of the thesis.

3.1. Ransomware

In order to better understand the project and RanAware functionalities, first it is important to understand the behaviour and characteristics of ransomware and how can we use these in order to identify them.

A ransomware is a type of malware that blocks and/or encrypts all the files in the compromised system, then offers the possibility to recover them if a ransom is paid. Ransomware often use threats like publishing sensitive data or destroying the data in order to blackmail the victim into paying the ransom.

As commented before, two types of ransomware can be distinguished: locker ransomware, that focus on blocking the access to the infected device and crypto ransomware, that focus on encrypting the data on the device. Both types of ransomware ask for a payment in order to recover the system. However, crypto ransomware is the most used, therefore from this point all the ransomware concepts will be focused in crypto ransomware.

A ransomware attack can be divided in several stages described as follows:

- **Infection**, the ransomware infects the machine when the victim executes a malicious program that can be obtained via a spam email, a compromised web access, a security exploit or other methods of malware propagation. This stage usually includes the privilege escalation of a ransomware.
- **Command and Control**, the ransomware establishes communication with a Command and Control (C2) server managed by the attackers. This server is responsible of receiving information of the victim's machine and controlling some actions of the ransomware, for example, when to start the encryption process and to provide cryptographic keys.
- **File Encryption**, once the ransomware is inside the system, proceeds to lock the victim's files or device. Ransomware encryption has two approaches: Either it crawls the system encrypting all the files found in the device or it targets specific critical system structures like the Windows File Table.
- **Extortion**, following the encryption process, a ransomware usually creates a ransom note offering the decryption key in exchange for a ransom.

3.1.1. Common Ransomware Actions

Based on the actions commented above, it can be inferred that ransomware have a remarkable characteristic behaviour and often follow specific patterns of activity [2], [7], [9]–[11].

One of the most indicative behaviours of ransomware is the **file interaction or file system activity**. Ransomware perform several actions related with files, namely, file creation, reading, writing, modification or deletion. Usually, ransomware creates files containing the ransom note in each directory where an encrypted file can be found. When encrypting a file, the encrypted contents are written into a new file or overwriting the contents of the legitimate file. Ransomware also deletes temporary files or some system files. It is also common for ransomware to modify the extension of the encrypted files with the name of the ransomware.

The other most representative aspect of ransomware is the **usage of encryption**. Ransomware uses standard algorithms, like AES, ChaCha or RSA, to encrypt the files of the infected device. Sometimes ransomware uses the cryptographic API available on the system; however, the great majority now uses their own cryptographic implementations of the algorithm.

Another activity characteristic of ransomware is the **communication with a Command-and-Control** server (C2). The ransomware communicates with this server to send information of the infected device and to receive cryptographic keys to be used for the encryption. Usually, the keys received by the C2 server are public keys used to encrypt the symmetric keys used by the ransomware to encrypt the files.

Ransomware also tends to **modify the Windows registry** in order to achieve persistence in the infected device. The most common operation is to register to be executed each time the computer is powered on.

One more action that ransomware performs in Windows systems is the **deletion of the volume shadow copies** [14]. Volume Shadow copies are backups of the critical files of the system created automatically by Windows. Ransomware deletes them in order to hinder the recovery capabilities of the system.

All these operations can be used in order to identify the presence of a ransomware in a device.

3.2. Windows I/O

3.2.1. Windows Input/Output Manager

Input/Output (I/O) operations refer to any operation, program or device that transfers data to or from a computer. Typical devices that perform I/O operations are: keyboards, mice, disk drives, external drives, etc. Device drivers provide the necessary software that serves as a connection between the external devices and the operating system [15].

The Windows kernel-mode I/O manager is in charge of the communication between applications and the interfaces provided by device drivers. The communication between the operating system and device drivers is fundamentally done through I/O request packets (IRPs). IRPs are similar to network packets; they are exchanged from the operating system to the specific drivers and from one driver to another.

3.2.2. Windows I/O Model

Every operating system has an I/O model for handling the data flow to and from the peripheric devices. The Windows I/O model has the following features [16]:

- All I/O requests to drivers are sent as IRPs.
- I/O Operations are layered. User-mode subsystems call I/O services, exported by the I/O manager, so this can carry out the I/O operations in behalf of the application or the user. The I/O manager creates one or more IRPs for each call, and routes them through the corresponding drivers.
- The I/O manager defines a set of standard routines that drivers can support.

3.2.3. IRP Function Codes

I/O Request Packets (IRPs) are kernel structures that device drivers use to communicate with each other and with the operating system. An I/O operation is described by one or more IRPs.

Each IRP is identified by a major function code (IRP_MJ_XXX), which tells the driver or the underlying device driver which operation should carry out to satisfy the I/O request [17].

Some of the IRP major codes that are sent to drivers are:

- IRP_MJ_CREATE: Sent when there is a request to open a handle to a file object or device object [18].
- IRP_MJ_CLOSE: Indicates that the last handle of the file object that is associated with the target device object has been closed and released [19].
- IRP_MJ_READ: Sent any time following the successful completion of a create request. Usually, an application has requested a data transfer to a device. In the context of files, it is sent when reading from a file [20].
- IRP_MJ_WRITE: Is very similar as the previous code. In the context of files, it is sent when modifying the contents of a file [21].
- IRP_MJ_QUERY_INFORMATION: Sent to obtain metadata about a file or file handle [22].
- IRP_MJ_SHUTDOWN: Sent to notice that the system is being shut down [23].

In addition, some of the IRP major function codes are accompanied by IRP minor codes (IRP_MN_XXX) that taken together allow to identify exactly what function is being requested.

3.3. Windows File System

A file system is the structure that an operating system uses to control how data is stored in and retrieved from the storage devices, usually disks. All the operations related with files like creation, modification, deletion and so on, are operations that need to go through the file system as these operations are modifying the storage.

The Windows file systems are implemented as file system drivers working above the storage system.

3.3.1. File System Filter Drivers

A file system filter driver, or minifilter driver, is an optional driver that adds value or modifies the behaviour of the system by intercepting requests targeted at a file system, or another file system filter driver. This interception allows a minifilter to extend, replace or modify the functionality of the original request. Some examples of minifilter drivers are: anti-virus filters, backup drivers, encryption products [24].

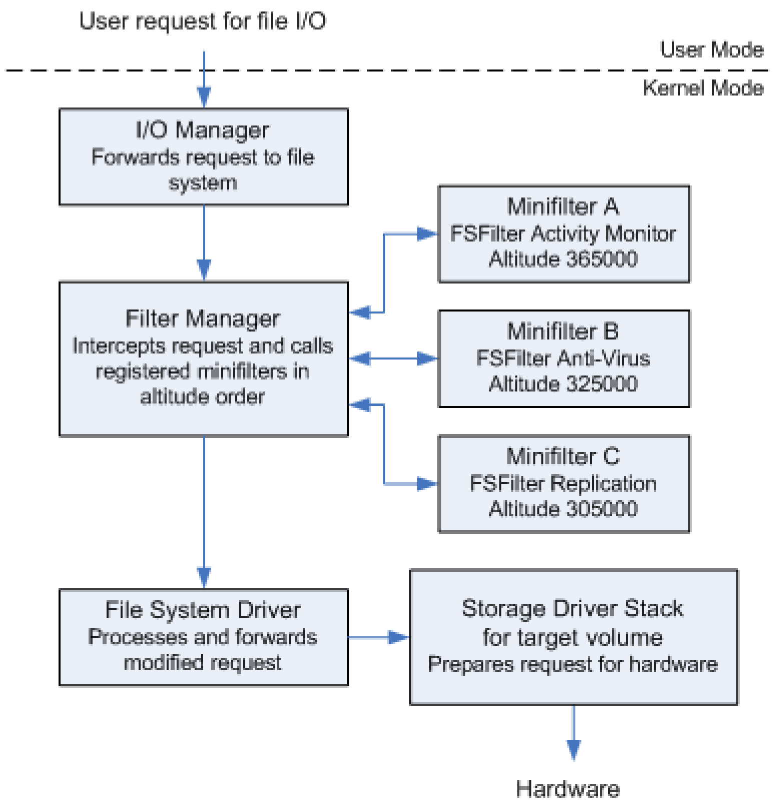
3.3.2. Filter Manager

The filter manager (FltMgr) is a kernel-mode driver that implements and exposes functionalities required in minifilter drivers so these do not have to implement and manage the complexities of file I/O.

The filter manager is automatically attached to the file system stack when a minifilter driver is loaded. A minifilter is attached to the file system stack indirectly, by registering the desired I/O operations it's going to filter to the filter manager [25].

There can be several minifilter drivers attached to a single file system stack, with the objective to determine the load order of the minifilter drivers Windows uses the *altitude*. Each minifilter driver must have a unique *altitude* identifier. The *altitude* of a minifilter driver is used to ensure the load order of the different minifilter instances and to determine the order that the filter manager calls the minifilter drivers to handle the I/O operations [26].

The following figure illustrates a simplified I/O stack with the filter manager and three minifilter drivers.


User request for file I/O
User Mode
Kernel Mode
I/O Manager
Forwards request to file system
Filter Manager
Intercepts request and calls registered minifilters in altitude order
Minifilter A
FSFilter Activity Monitor
Altitude 365000
Minifilter B
FSFilter Anti-Virus
Altitude 325000
Minifilter C
FSFilter Replication
Altitude 305000
File System Driver
Processes and forwards modified request
Storage Driver Stack for target volume
Prepares request for hardware
Hardware

Source: https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

Figure 3.1: Simplified I/O stack

3.3.3. Filtering with Minifilter Drivers

A minifilter driver can filter IRP based I/O operations, fast I/O and file system filter operations. For each of the I/O operations it chooses to filter it registers a PreOperation callback routine and/or a PostOperation callback routine. When the filter manager is handling an I/O operation, it calls the appropriate callback routine for each minifilter driver that has registered for that operation. When the callback returns, the filter manager calls for the appropriate routine of the following minifilter in the file system stack that has registered for the operation. The operations are identified with the corresponding IRP major function code.

Using *Figure 3.1* as an example, assume that all three minifilter drivers have registered for the same I/O operation. The filter manager would call their PreOperation callback routines in order of altitude, from highest to lowest, and then it would forward the I/O to the next file system device driver. When the I/O request is completed, the filter manager receives the request for completion, and it would call the PostOperation callback routines of the registered minifilters in reverse order, from lowest to highest.

3.3.4. Minifilter Drivers and User Mode Applications

The filter manager supports bidirectional communication between user-mode and kernel-mode through communication ports [27].

The minifilter driver creates a communication server port specifying the security parameters, a callback to handle when a client connects, another callback to handle disconnection of clients and a callback to process the received messages.

Once the communication port is created, the minifilter begins to listen for incoming connections to the port. When a user-mode attempts to connect to the port, the connection callback is called to handle the creation of a new connection. A connection is only accepted if the user-mode caller has sufficient permissions as specified by the security descriptor on the port. Each connection with the driver gets its own message queue and private endpoints.

Once the communication is established, a user-mode application can send messages to the minifilter driver that will be handled by the registered callback. The minifilter driver can reply to the received messages and can also send messages to the user application.

Whenever an endpoint closes, either kernel-mode or user-mode, the connection is terminated. When the connection is closed from the user-mode side, the disconnect callback of the minifilter driver is called so it can close its handle to the connection.

3.4. Tools

In this first section I am going to present the different tools I have used in order to develop the project.

Git [28] is used to manage and work with the code. Git is a version control system designed to handle projects fast and efficiently. I paired Git with **GitHub** [29] to store the code online and be able to work using different machines.

Visual Studio 2019 Community Edition [30] is the Integrated Development Environment (IDE) that I used to develop and compile all the versions of the code. I had to use this IDE since it is the one that is compatible with the Windows Driver Kit (WDK)

Windows Driver Kit (WDK) [31], is used to develop, test and deploy drivers for Windows. The WDK provides the tools needed to develop minifilter drivers.

VMWare Workstation Pro [32] is a virtualization software. I used virtual machines in order to test the driver since I had to run ransomware samples and I did not want to deliberately infect my computer with ransomware.

WinDBG Preview [33] is the new version of the WinDbg, the debugger for Windows systems, with graphical interface, renewed visuals and some more functionalities. I used WinDBG Preview to debug the Windows Kernel and the driver.

VirtualKD-Redux [34] is a tool that allows to automatically launch and attach the WinDBG Preview to a remote instance of Windows running in a virtual machine. Debugging the Windows kernel is comparably easier using this tool than preparing all the environment manually.

DebugView [35] is an application that allows to monitor debug output messages of a local or network system. It can show the Win32 debug output and Kernel-mode output. Used to review the correct functionality of the programmed driver.

Process Monitor [36] is a monitoring tool for Windows that shows real-time all the file-system, Registry and process/thread activity in the system. I used this tool to review some Ransomware actions and the actions of RanAware.

Conti source code [37], *Conti* is a ransomware developed by a Russian ransomware group. With the recent Russian conflicts, a lot of cyberattacks were launched and the source code of the *Conti* ransomware was leaked by a twitter user [38]. I used this code to investigate which are the actions of a state-of-the-art ransomware and how are they performed; thus, gaining insight on what operations should RanAware focus to detect ransomware.

4. Project Design

The great majority the ransomware operations mentioned in section 3.1.1 lead to file modification or file system, increasing its activity. As a consequence, a great strategy to detect ransomware is to monitor the file system, particularly file modifications.

For this reason, RanAware focuses on file modifications and file system operations to detect ransomware activity. Initially, the idea was to monitor encryption API calls, which is a common trait of ransomware. However, given that most ransomware implement their own custom encryption methods, the approach was discarded, for more information refer to section 9.1.

RanAware is composed of different modules. The first module is a file system filter driver or minifilter driver that monitors file operations occurring in the system. The second component is a user application that communicates with the minifilter driver, reading the information about the I/O operations and which process performed them. Then, by maintaining process statistics and decoy file monitoring, the application determines whether a process is a ransomware or not. After detecting a ransomware process, RanAware stops it, generates a log of the operations this process has performed and quarantines the executable to ensure it is never executed again, at least by any legitimate user. The third and last component is a decoy file generator that ensures to strategically generate decoy files, that are files that should never be modified, throughout the system. This design is depicted in *Figure 4.1.*

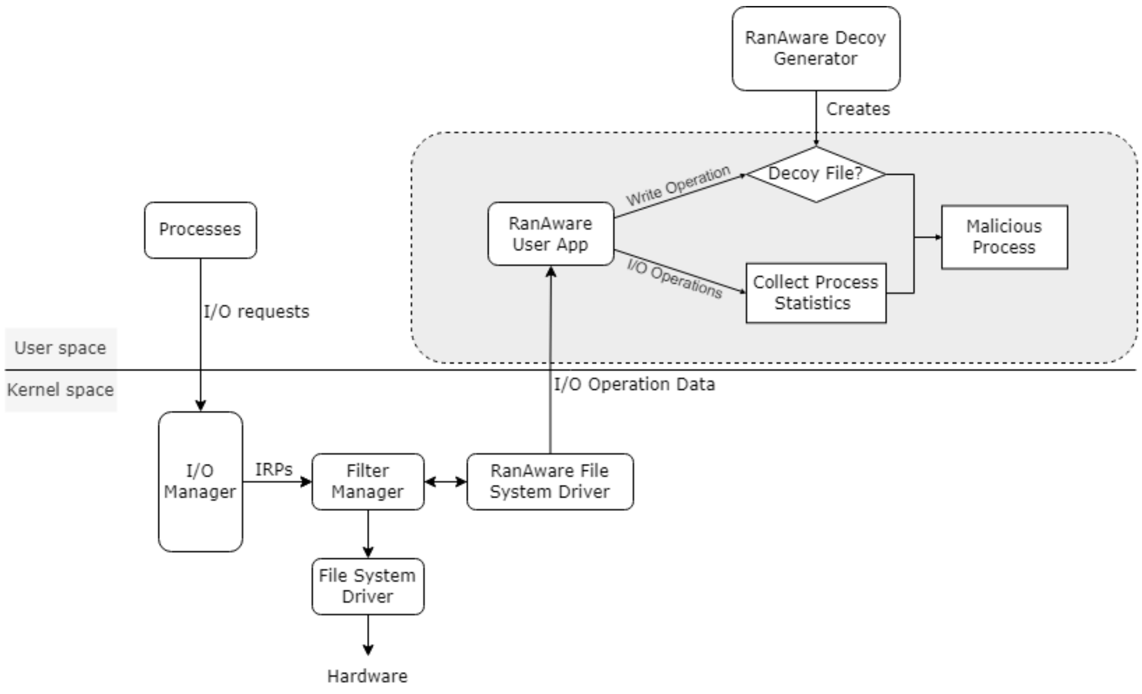


Figure 4.1: RanAware design diagram

In the following sections, I will focus on the details of each one of these components and the reasons I chose to implement them.

4.1. Driver Module

At the end, all of the ransomware operations related with files or file system activities are Input/Output operations directed to disk. In Windows, these I/O operations are translated into IRPs that must be handled by the I/O manager and the corresponding drivers in order for them to be completed.

For this reason, the first module of RanAware is a minifilter driver. This driver is constantly monitoring the I/O operations that occur in the system and logging information about them. The driver stores the data of the requests until the client application reads them from the driver's log.

With this module RanAware will log all the file system operations of any process of the system, including any possible ransomware running in it.

For each I/O operation logged, the minifilter driver stores different pieces of information; the most relevant data stored by the driver is: the time when the request is issued, the time when the request is completed, the ID and the name of the process that issued the petition, the ID of the thread that issued the petition and the IRP major and minor codes that identify the I/O operation. In a particular case, it also stores whether the operation is a rename or not.

The driver registers with a high altitude, specifically 37000, that is the one that corresponds to drivers observing and reporting file I/O [39].

4.1.1. Filtered Operations

As commented before, each I/O operation is identified by different IRP major codes and a driver that wants to filter a given I/O operation must register for the IRP Major code that identifies the I/O operation.

The RanAware driver does not monitor all the I/O operations that happens in the system. The driver focuses on the operations that are more pertinent for detecting the activity of ransomware, these are:

- File reading: Identified by IRP_MJ_READ.
- File writing: Identified by IRP_MJ_WRITE.
- File attribute modification: Identified by IRP_MJ_SET_INFORMATION
- Directory enumeration and modification: Identified by IRP_MJ_DIRECTORY_CONTROL

RanAware only filters these operations since are the most relevant related to ransomware activity in the file system. The file reading and writing is used by ransomware to read the contents of the files before encrypting and to write the encrypted contents back. The file attribute modification is monitored considering that ransomware habitually adds the ransomware name as a file extension after encrypting. Finally, the directory enumeration is a common operation performed by ransomware in order to list all the files of a directory.

4.1.2. Actions

Given that the information that the RanAware minifilter records for each operation is the same, the minifilter driver registers the same PreOperation and PostOperation callback routine for all the I/O operations its filtering.

During the handling of the PreOperation callback, the driver records all the parameters available at the start of the operation. When the PostOperation callback is called, the minifilter driver records the final parameters that could not be stored before the operation is completed, like the status of completion, the completion time or the information returned when completing the operation.

The minifilter driver maintains a double-linked list storing the records of all the logged operations. Whenever a new operation is logged, the driver allocates a new record and stores it into its linked list.

4.1.2.1. Identifying a Rename Operation

There is a special case regarding one operation that needs additional treatment: the rename operation. The IRP_MJ_SET_INFORMATION [40] code represents all of the operations that request to set metadata of a file or file handle; this means that this IRP code covers a great quantity of I/O operations. Since RanAware is interested in the rename operation, whenever the driver receives this IRP code needs to process the request data in order to recognise a rename operation.

To identify the rename operation, RanAware uses one of the parameters of the IRP_MJ_SET_INFORMATION, the *FileInformationClass* parameter that specifies the type of information to be set to the file. This parameter can take several values, for the RanAware driver scope, the key value is *FILE_RENAME_INFORMATION.*

Whenever the driver is processing the IRP of an operation identified by IRP_MJ_SET_INFORMATION and the *FileInformationClass* of this IRP takes the value of *FILE_RENAME_INFORMATION*, the driver understands that this operation corresponds to a rename of a file, therefore it stores in the record of this operation that it is a rename.

4.1.3. Communication with Client

The last action the driver performs is communicating with the client application to send the data of the logged operations.

The driver creates a communication port during initialization time. This port is used by the client to start a communication that lasts until one of the two parties closes the connection.

Once the connection is created, the driver will receive commands from the user application. The only command that the driver responds for the moment is the *GetLogs* command.

When the driver receives the *GetLogs* command it iterates through the double-linked list that stores all the log records and copies them one by one to a client buffer until the buffer is full. If there are no more records left, the driver returns an error code indicating that there is no more information to read.

4.2. Client Application

The second module of RanAware is the client application. This application communicates with the driver, writes a log of all the operations in the system, monitors process and decoy files and is the responsible of taking the final decision to identify ransomware processes.

The first action that the application does is to communicate with the minifilter driver to retrieve the log records of the file system operations. The application analyses this data and by monitoring the activity of decoy files and maintaining statistics of processes identifies ransomware activity. In addition, all the data that the application reads from the driver can be written into a user file or through the screen.

After finding a malicious process the RanAware application will terminate it, store a log of all the operations the process has performed in the system and ensure that the process is never executed in the system again. The logs provide useful information in order to leverage the impact on the system by the malicious process.

4.2.1. Driver Communication

The communication between the driver and the application is a straightforward task. The application initiates a communication with the driver during the start-up.

Then, the application creates a thread that is constantly communicating with the driver to receive log records. Upon receiving a log record the thread analyses the data of the record, looking for suspicions file modifications, updating the statistics of tracked processes and performing any action if necessary.

4.2.2. Process Monitoring

With the information recovered from the logs received by the driver, RanAware application monitors the file activity in order to detect unauthorized access to the decoy files, which is any operation that tries to modify the contents of the file, and maintains statistics of the processes that are useful to identify a ransomware.

The first thing that RanAware application investigates in a file operation log is whether the operation is directed to a decoy file or not. In the case that the operation is a writing operation and is directed to a decoy file, the application will assume that the process that issued the application is a malicious process. For more information refer to section 4.3.

The primary process statistic that RanAware records is the number of file renames in a set period of time. The reasoning behind this is the fact that ransomware normally tends to change the file extension after encrypting a file. The renaming of a file is not a very common operation that an average user does, particularly if the operation is repeated several times in few seconds. This is the reason why the driver needs to indicate which of the recorded I/O operations are rename operations.

The current configuration is set to determine a process as a ransomware if it performs more than 10 renames in a period of 30 seconds, both amount of renames and time interval are configurable parameters of RanAware. Ransomware attacks usually operate as fast as possible and would produce far more renames in a given time period, regardless the extra time that the encryption process of bigger files may imply. Conversely, it is very unlikely for a user to perform this many rename operations in the same time period. Therefore, the current configuration records the benign operations but does not classify them as malicious.

In addition, the process statistics are maintained within a structure that is easy to expand allowing the upgrade of it with new statistics or information that can be useful to detect ransomware.

4.2.3. Avoid detection of system operations

RanAware must be sure that all the system processes can operate without any type of intrusion, for this reason, it maintains a list of benign processes or directories that are considered secure. These are the system directories that usually contain legitimate applications of Windows such as "C:\Windows\System32", "C:\Program Files" or "C:\Program Files (x86)". RanAware avoids monitoring processes that belong to any of these directories since they are considered secure. This way allows RanAware to not record any statistic of Windows internal processes so it uses less resources and does not interfere with the system operations nor the user experience.

All the processes in the system directories are considered safe since malware is usually not located in the system directories. In addition, in order for any malware to be installed within these directories, requires that it is executed with administrator privileges.

Consequently, any malware executed with administrator privileges could perform any action in the infected computer such as: disabling RanAware, disabling the security measures, changing permissions, installing itself into the system directories and so on. This is the reason why RanAware assumes that a "system" process is benign, since if a malware is executed with administrator privileges means that any action taken by RanAware can be circumvented in some way by the malware.

4.2.4. User Alerting, Logs and Quarantine

After identifying a malicious process, the RanAware application will terminate it, alert the user, store logs about the process activity and quarantine the malicious executable.

The log file is stored in the RanAware local report directory located at: "C:\Users\<User>\AppData\Local\RanAware\RanAware_Reports\". The log contains all the activity of the terminated process recorded by the driver. In here we can see all the actions that this process has performed, over which files, the name and route of the process' executable, the time when these were performed, etc.

The alert informs the user that a suspected ransomware process has been found and terminated. It indicates the route of the created log and recommends contacting with the system administrator, if any, and performing a malware analysis in order to detect other possible threats on the computer.

Finally, RanAware maintains a protected quarantine directory destined to store all the malicious samples that has detected. This way, these are never executed again in the victim's computer unless they are deliberately executed by someone with administrator permissions.

The access to this directory is restricted to administrator accounts and only read and write permissions are allowed. Furthermore, whenever a new sample is quarantined in the directory, the *Access Control List* of each sample, the Windows mechanisms to describe the allowed operations for a given object, is modified so only administrators can read and write the file and to ensure that it cannot be executed.

4.2.5. Additional Actions

RanAware application performs some additional actions apart from the mentioned in the previous sections.

In order for RanAware to be able to terminate any process detected as malicious, it must have the correct permissions [41], for this reason, during start-up, the RanAware application modifies its privileges to circumvent any malicious process trying to avoid termination.

In addition, the first time that the user application of RanAware is executed it registers the decoy generator module as a task in the Windows machine so it is executed automatically every time the computers powers on.

Whenever a process is terminated it registers the name of this process and the executable route into a list so that any user or system administrator can take a look at which are the malicious processes identified by RanAware.

4.3. Decoy Generator

Ransomware tends to have an aggressive behaviour; they start to iterate over all the directories of the infected computer encrypting all the files found during the process with the objective of affecting the maximum amount of information of the victim.

In some cases, some ransomware aims to affect only the important files of the victim's computer, these usually are the files that are created or accessed recently. The objective is to avoid infecting files that are i.e., 2 or 3 years old, since they may be not that important for the victim. Although this behaviour is not the one that I have observed during the project testing.

Taking advantage of the aggressive behaviour of ransomware attacks I decided to use a well-known technique known as decoy files. The decoy files are files that are present in the system but should not be modified by any process at all.

This way, when RanAware detects a process that tries to modify a decoy file means that it is with high probability, a ransomware.

All of the operations that are performed on a decoy file can be detected with the logs obtained from the minifilter driver. The user application is able to distinguish which process is performing the access; thus, avoid detecting the RanAware processes as malicious.

The third module of RanAware is a decoy generator that is executed each time the computer is powered on. Once executed, the decoy generator will generate several decoy files throughout the system. The file contents resemble real contents of files that can be found in any personal desktop computer.

The reason behind executing the decoy generator each time the computer powers on is to have a recent creation and access time in the file properties.

5. Results

To ensure that RanAware is working as intended, is crucial to evaluate its functionalities within real scenarios. However, it is also important and often overlooked, to measure the impact that this type of projects have in a system's performance. In the literature, I did not find any information of the overhead that the proposed solutions caused in a system; thus, I find this part of the project really important.

All the tests have been executed in a virtual machine created in VMWare Workstation Pro, with a configuration of 8GB of RAM, 4 processors and running Windows 10 Pro version 21H2 in test mode.

The computer that executes the virtual machine is a laptop model MSI GF75 Thin, running Windows 10 Pro version 21H1, with 16 GB of RAM, an Intel Core i7-10750H CPU @ 2.60GHz (12 CPUs) and a 512 GB SSD disk.

RanAware is always executed the same way; the driver and the RanAware application are running, additionally, the application writes all the records received by the driver in a text file.

In this section I present first, an evaluation of RanAware against real ransomware attacks and then an evaluation of the impact that RanAware has in a system considering different scenarios.

5.1. Ransomware Detection

In order to determine the effectiveness of RanAware I downloaded several samples of ransomware from different sources and I executed all of them in the virtual machine mentioned before with RanAware running.

Previous to executing the ransomware a snapshot with a clean state of the machine is prepared so it is possible to roll back to a clean system after a ransomware execution.

The first two samples I executed are two educational ransomware programs, *EDA2* and *hidden-tear* [42]. These samples perform the typical ransomware operations, they first communicate with a Command and Control (C2) server in order to retrieve a key, then start encrypting the files and finally changing the desktop image background. For executing both samples I had to configure a local server to act as a C2 server. In this case, RanAware is able to identify and terminate both samples.

The third sample I have executed is a sample of the *LockBit* ransomware. The sample has been obtained from an online sandbox called *AnyRun* [43].
The SHA-256 hash of the sample is:
"0545f842ca2eb77bcac0fd17d6d0a8c607d7dbc8669709f3096e5c1828e1c049".
In this case, RanAware is also able to identify and terminate this ransomware before it can encrypt a significant number of files in the system.

BlackMatter is the fourth type of ransomware I executed to test RanAware. I obtained the sample from the same webpage as the previous one, *AnyRun,* the SHA256 hash is: "2f1404af9417dbbbe69d53cb0cc0d6f2fc79138c372ab3c498ec05f60dbdc9f3".

Again, RanAware is able to detect and stop the ransomware before any critical damage occurs in the system.

The last sample I tested RanAware with is a *WannaCry* ransomware. The sample with SHA-256 hash: "ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa" is detected and terminated by RanAware successfully.

The behaviour of these ransomware samples is very aggressive and simple. Both the *LockBit* and *BlackMatter* ransomware start scanning the root system directory, *C:\*, recursively and encrypting all the files found. Although the two of them target different files, it does not seem the ransomware searches for recent files or specific file types. From the behaviour observed, *LockBit* ransomware encrypts everything it finds in alphabetical order and *BlackMatter* in reverse alphabetical order, always excluding the system directories since they do not have any right to modify the contents. *WannaCry* sample attacks with a different approach, it starts encrypting the contents of the directory where it is executed and then backtracks to the root directory encrypting everything it finds. *EDA2* targets a directory specified in its configuration file and encrypts every file that is present in this directory or subdirectories in alphabetical order, in this test *EDA2* is launched against the root directory, C:\. Finally, *Hidden-tear* starts the encryption process from the Desktop of the current user and encrypts every file it finds in alphabetical order.

In *Table 5.1* there is a summary of the ransomware samples launched that specifies whether if they are detected by RanAware or not, the number of encrypted files and the percentage of encrypted files over the total files in the system.

The estimated total number of files in the system is 155.000; this number is the number of files scanned by an antivirus, *Malwarebytes* [44], during a complete system scan. Although the total number of files in the system reported by Windows is higher, around 528.369, a lot of these files are special system files that won't be encrypted by ransomware.

Ransomware	Terminated by RanAware	Nº of encrypted files	% of encrypted files
EDA2	Yes	10	0,006 %
Hidden-tear	Yes	0	0,00 %
LockBit	Yes	10-97	0,06 %
BlackMatter	Yes	10-142	0,09 %
WannaCry	Yes	10-30	0,02 %

Table 5.1: Ransomware test summary

The number of encrypted files shows the minimum and the maximum of encrypted files obtained in several executions of the sample. The maximum number of encrypted files depends on how fast the ransomware operates. Even when the ransomware is detected when it encrypts at most 10 files, the issued operations before detection can be greater, especially if the ransomware spawns several threads to perform the tasks concurrently. Nevertheless, the total percentage of encrypted files is extremely low.

Hidden-tear first file encryption targets a decoy file, as a consequence it is instantly detected by RanAware. If the ransomware does not target a decoy file, it is detected by the amount of rename operations performed in a small period of time; this is the reason the minimum number of encrypted files is sometimes 10, that is the threshold of file renames configured in RanAware.

5.2. Benchmarking

All of the programs that run on a system have an impact in the performance of it in one way or another. Accordingly, RanAware also affects the performance of the system that is protecting; hence, I evaluated the resources RanAware is using and the overhead the minifilter driver implies on the I/O operations.

5.2.1. Overhead in I/O operations

The performance tests evaluate the overhead that the driver causes to the I/O operations, particularly to the write operations that are the most common ones that the driver is monitoring.

The tests consist of computing the elapsed time of writing the same 512 random characters into 10, 100, 1000 and 10000 files. I first executed these tests without RanAware running and then with RanAware enabled to compare the impact of the whole project in the I/O operations.

In addition, I have considered **three different scenarios** regarding the load of the system. In the first scenario the system has a **low workload** and only the benchmark program is running. In the second scenario the system has a **medium workload** with two live stream videos with a quality of 1080p playing in the background using Microsoft Edge. In the third scenario the system is under **heavy workload** using the same configuration as the previous scenario and additionally a game, *Bombslinger,* and a movie are running in the system.

During the executions of the tests, the Real Time Virus Protection of Windows Defender has been deactivated since it could start scanning the system or the files that are being created while the test is executing causing a very significant impact on the data obtained.

All the results that are presented in this section follow some code names described in *Table 5.2*. This is done in order to have shorter names that are easier to represent.

Acronym	Meaning
LL	Low Load
ML	Medium Load
HL	Heavy Load
ND	No Driver running
D	Driver running
XF	X Files created

Table 5.2: Acronym table for the test nomenclature

In the following figures I present the collected data of each experiment. I have collected 100 samples for each number of written files, with and without driver and with different system loads.

The data is represented in a boxplot, displaying the number of seconds needed to write the contents of all the files. The boundaries of the box indicate the lower quartile (25th percentile) and the upper quartile (75th percentile), and the horizontal line inside the box represents the median. The outliers that exceed the upper limit, which is the maximum value of the upper whisker, are discarded. There is one plot for each number of written files and each plot includes the data for the executions with and without a driver and for the different system workloads.

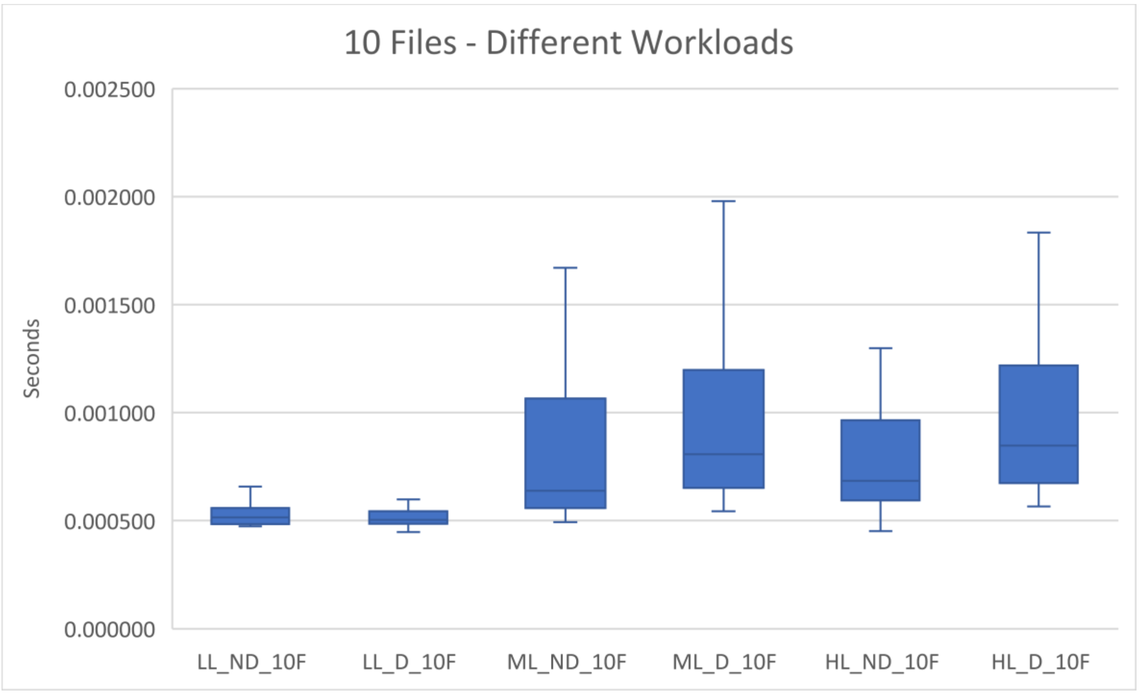


Figure 5.1: Comparison of writing 10 files

Figure 5.1 presents the time it takes to write 10 files. In this case when RanAware is running and the system is under some pressure, the time to write the files increases slightly.

It is important to remark that in this case, the results report that some file writings are slower under a medium load than with a heavy load in the system. This is probably caused because writing 10 files only is a very small amount of work that can be affected by a lot of factors during execution such as cache.

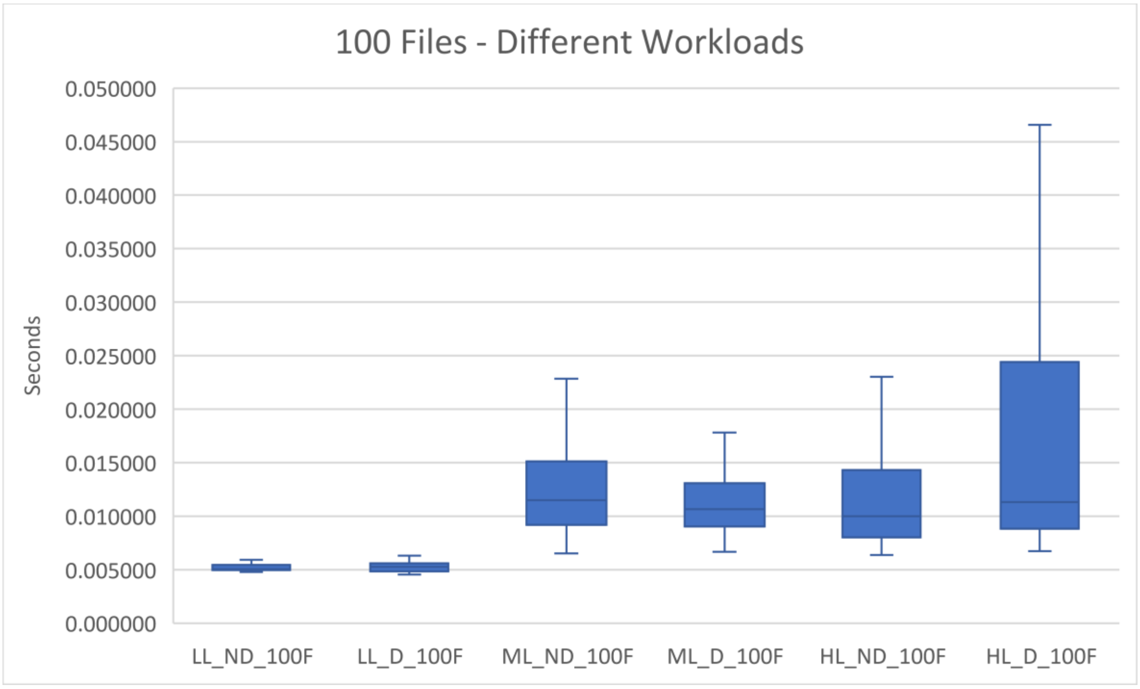


Figure 5.2: Comparison of writing 100 files

Figure 5.2 that corresponds to writing 100 files, shows a similar behaviour regarding the previous iteration of the test; when RanAware is running under a busy system, the execution time reported by the test increases.

In this case, it can also be observed that when the system is under medium load, in some occasions the test reports similar or higher execution times than when the system is under heavy load. This is probably caused by the same reasons as with the first iteration of the test, writing 100 Files is a light task that can be affected by different factors.

At first sight, *Figure 5.1* and *Figure 5.2* start to indicate the beginning of a trend; whenever the system is busy the time it needs to complete the writing requests increases. This is more noticeable if we compare the times reported with a relaxed system and a very busy one.

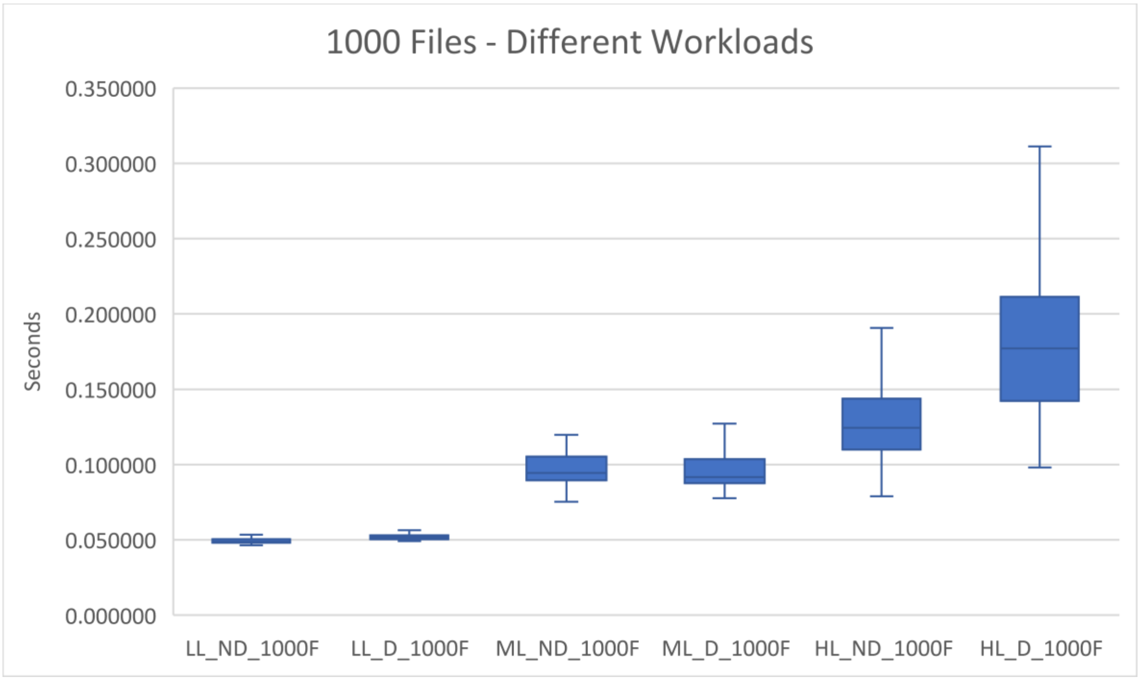


Figure 5.3: Comparison of writing 1.000 files

Figure 5.3 shows the time needed to write 1.000 files. In this test case the execution of RanAware also increases the overall time needed to complete the writing requests, although it is particularly noticeable when the system is under a heavy work load. With the other system work loads, the differences in execution time are minimal.

The trend that started appearing in the previous executions is easier to identify; the busier the system is, the more time it needs to complete the writing requests. Unlike in the previous scenario, the difference between a medium workload system and a heavy one is easily discernible.

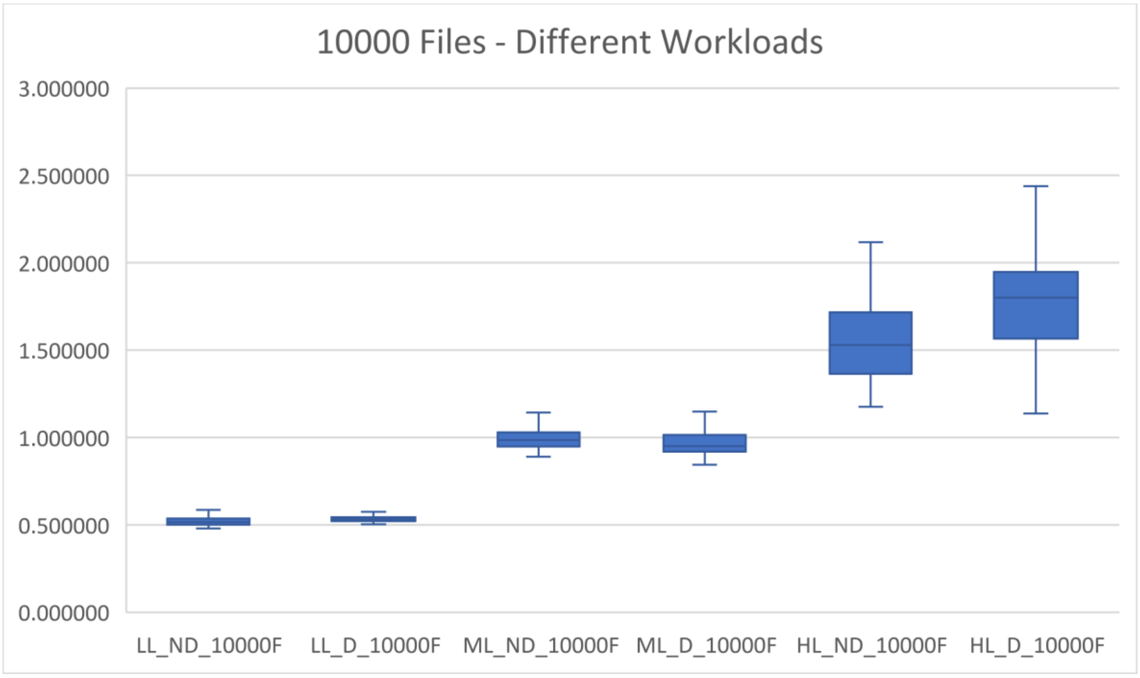


Figure 5.4: Comparison of writing 10.000 files

Figure 5.4 shows the last iteration of the test that consist of writing 10.000 files. Similar to the preceding test cases, in this iteration all of the trends mentioned above are easy to recognize. RanAware influences the execution of the test but only seems to affect when the system is under heavy load. In addition, the difference of execution time between system workload levels is clearly identifiable.

As it can be observed in the *Figure 5.3* and *Figure 5.4* report extremely similar results and show the most logical outcome of the tests; the busier the system is, the more time it needs to complete the file writings. While the results of the two first iterations of the test start showing this tendency, it is not until the test starts to perform a huge number of operations that this is remarked.

Even though, the two first iterations show similar results when the workload of the system is at medium or high levels, the difference in time between the two scenarios is clearly distinguished in the last two executions of the test. This is due to the small quantity of operations performed by the test in the initial iterations that can be affected by a lot of different factors, like background processes or system checks.

The overhead caused by RanAware varies among the different executions of the test, although it is only significant when the system is under heavy workload. In this case, the overhead differs significantly between each test case, the reason behind this variation is the background processes that at some time can influence the execution of the benchmarking program.

In addition, it must be considered that RanAware is constantly writing the logs retrieved from the driver into a file, that also can have impact on the tests when RanAware is running. However, during the executions of the test, we observed that the bottleneck is the CPU in most cases, which in the last scenario was operating at 100% of capacity, and not the disk that was operating between 0% and 4% of its capacity. After careful observation we could not identify the culprit of this behaviour, since Windows is a highly concurrent system the cause of this high CPU usage may be caused due to many factors. The analysis of this behaviour is left as an important part of our future work.

Table 5.3 presents a summary of the data introduced in this section along with the overhead caused by RanAware in each test case.

Written Files	Low Load			Medium Load			Heavy Load		
	No Driver	Driver	Overhead	No Driver	Driver	Overhead	No Driver	Driver	Overhead
10	0,000523	0,000509	-2,63 %	0,000770	0,000901	16,92 %	0,000763	0,000971	27,23 %
100	0,005190	0,005223	0,65 %	0,012330	0,011029	-10,55 %	0,011184	0,015727	40,61 %
1000	0,049238	0,051827	5,2 %	0,096632	0,095091	-1,59 %	0,125153	0,179759	43,63 %
10000	0,519502	0,532631	2,5 %	0,991071	0,961036	-3,03 %	1,535279	1,773325	15,50 %

Table 5.3: I/O overhead summary

All in all, from the data presented in this section, it can be observed that RanAware has some impact over the system, especially when the load of the system is high, as already mentioned we plan to further analyse the reason of this usage in our future work, preliminary analysis shows that the culprit may be poor performance of the web browser during the experiments.

5.2.2. Resources used by RanAware

To evaluate the resources used by RanAware I used different Windows API calls to obtain the virtual and physical memory, the percentage of CPU usage used by RanAware and the number of I/O operations that RanAware is performing.

Based on the results obtained from the previous section, RanAware activity affects more the system when it is under a heavy workload; for this reason, the metrics presented in this section are collected during the execution of the tests when the system is under heavy workload.

Resources	Total System Resources	Used system Resources	RanAware resource usage	% RanAware resource usage
Virtual Memory	9,25 GB	3,82 GB	5,8 MB	0,15 %
Physical Memory	8 GB	3,91 GB	12,5 MB	0,31%
CPU (%)	100	92,06 %	1,66 %	1,81%

Table 5.4: System resource usage

As can be observed in *Table 5.4* RanAware resource usage over the total resource usage in the system is certainly small. The amount of memory used mainly maintains all the data structures that RanAware uses and the CPU usage is very low since the actions that RanAware performs are not complex.

Table 5.5 shows the disk usage of RanAware after the execution for the test data and the recollection of the system resource usage data.

RanAware Disk Usage	
Read Operations	5
Write Operations	85767
Other I/O Operations	593979
Read bytes	0,76 MB
Written bytes	0,33 GB

Table 5.5: RanAware disk usage

RanAware disk usage is mainly determined by the operations that RanAware performs to write the record logs retrieved from the driver into a file. Since RanAware is constantly writing into a file, the written bytes are high; nevertheless, this is a necessary task in order to obtain detailed logs of the events that are happening at a file system level.

The amount of data written can be reduced by using RanAware application without writing into a file. Although this configuration will reduce the amount of disk usage by RanAware, the action logs of a terminated process will be less detailed since RanAware will not store all the actions that occurred in the system; hence, having only access to the most recent actions.

5.2.3. Discussion

RanAware adds an overhead to write operations, however, this is barely noticeable besides when the system is under heavy workload.

On the other hand, the resource usage of RanAware is low, the most consuming operations would be the writing of the logs, that I believe it is useful to determine the actions of the terminated processes.

Overall, I believe that the resource usage of RanAware is low and the impact it has in a system is assumable taking into account the protection that it offers.

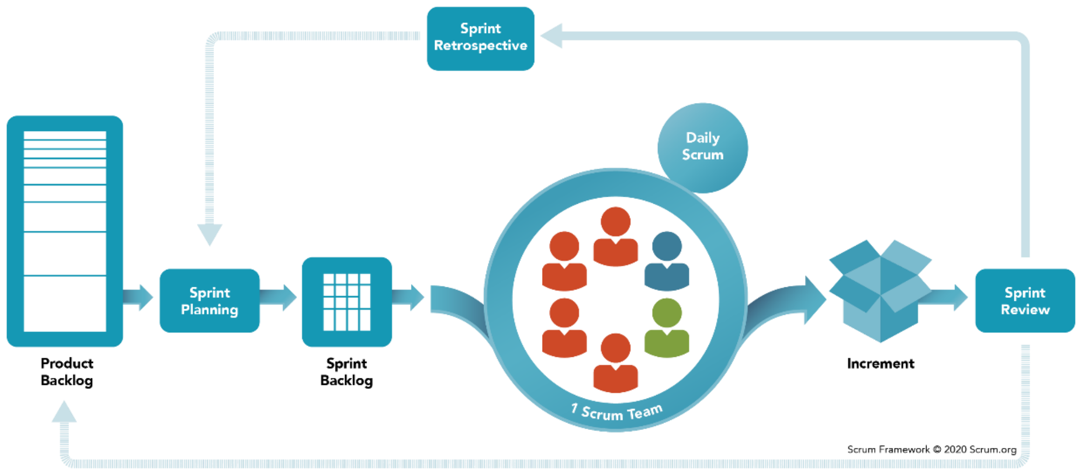
6. Project Methodology & Planification

6.1. Project Methodology

This thesis has been developed following a pseudo-*SCRUM* methodology. *SCRUM* is an agile framework that is based on iterative steps in order to develop a product as a team. *SCRUM* is characterized by splitting the work into *sprints*, that have an average duration of 2-4 weeks.

The work that is assigned to a sprint is divided into more granular tasks and these are distributed among the members of the team. During a sprint, there may be brief daily meetings or every two days in order to track the progress of the team and possible issues that may occur.

When a sprint is finished, a retrospective is performed in order to evaluate the weaknesses and strengths of the team during the sprint and take actions for the following sprints. These actions are then taken into account in the next sprint planification.



Source: https://www.scrum.org/resources/what-is-scrum

Figure 6.1: Scrum methodology

SCRUM is a very common methodology used by software development teams since it makes easy to organize, distribute the tasks and keep the team synchronized. However, as the team of this project is composed by two members, the developer and the director, there is not much point to perform a full *SCRUM* methodology.

The work methodology followed in this project is similar to *SCRUM* in the sense that the tasks were all in a backlog and each week the team members met in order to discuss the work of the previous week and decide the next steps. Nevertheless, there were no daily meetings nor retrospectives. Anytime the developer had some doubts contacted with the director and a meeting was scheduled or the discussion was performed offline, i.e., mail or telegram.

6.1.1. Tools

The tools used to work with this methodology are the following:

- **Google Meet:** Used to perform some of the weekly meetings if it was not possible to perform them face to face.

Figure 6.2: Google Meet logo

- **Telegram:** Used to re-schedule meetings, talk about small problems, etc. Replaces the e-mails as a more interactive and fast way to talk between the team members.

Figure 6.3: Telegram logo

6.2. Project Planning

Planning a project is essential in order to organize and decide which tasks are going to be performed, estimate the time is going to be spent on each task and to have an overall vision of the project progress and development. This allows to define deadlines and tweak the project in order to meet the desired requirements.

In this section I'm going to present the initial planification defined at the start of the project, followed by the final planification and finishing with expanding the deviations and changes that the initial plan suffered.

I started with the thesis work on January 14th and I expect to finish it at the middle of June. The duration of this project is about five months, considering half of January and half of June.

6.2.1. Initial Planning

Figure 6.4 shows the initial planning using a Gantt diagram. Each of the columns of a month represents one week of said month.

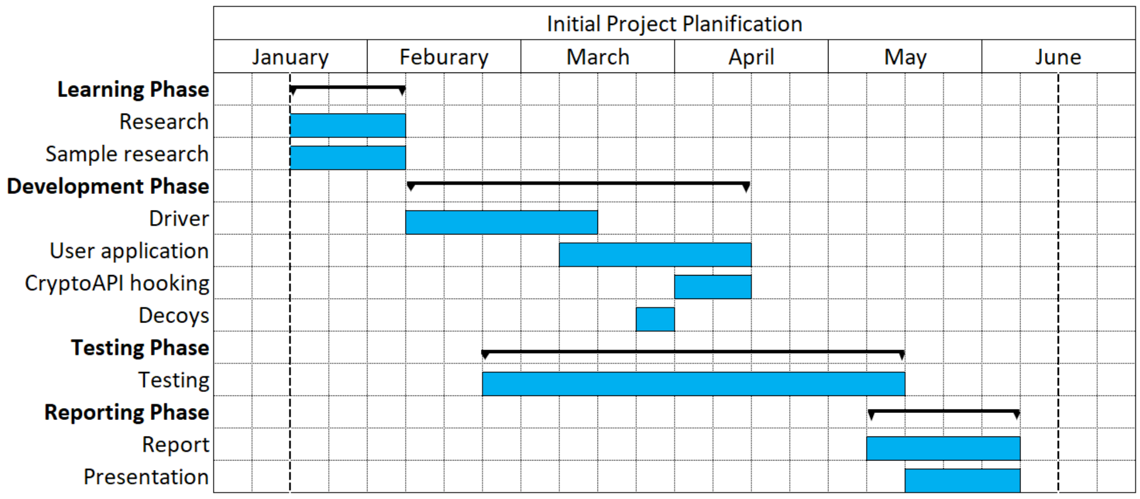


Figure 6.4: Initial Gantt planning

In this graphic, the overlapping tasks means that they were developed in parallel during that period of time, since they are complementary or are related in some way.

Detailed planning information is defined in *Table 6.1.*

Estimated time	
Task	**Dedication (hours)**
Learning Phase	**95**
Research	85
Sample Research	10
Development Phase	**200**
Driver	70
User application	80
CryptoAPI Hooking	35
Decoys	15
Testing Phase	**50**
Testing	50
Reporting Phase	**60**
Report	55
Presentation	15
Total	**405 hours**

Table 6.1: Initial estimated time

In *Table 6.1* I detailed the initial estimated number of hours dedicated to each task. This estimation takes into account that the number of hours dedicated per week to the project is not stable. Depending on other responsibilities outside the thesis I could invest more or less time into developing the project.

It is remarkable that the testing phase is the longest one in time yet the hours dedicated to it are low, this is because the testing is a continuous task performed during all the development of the project. Every implementation is tested while it is being developed, therefore the testing starts almost at the same time as the development and finishes a little bit later.

Another important thing to remark is that during the Development phase there was some investigation and learning steps in order to continue with the project, this is included in the Development hours.

6.2.2. Final Planning

The planification of the project suffered from some modifications and delays that ultimately lead to a deviation of 40 hours. In this section I am going to present the final planning considering all the deviations and changes that occurred regarding the initial planification. In the next section I will expand on the deviations and their reasons.

Figure 6.5 shows the final planning using a Gannt diagram.

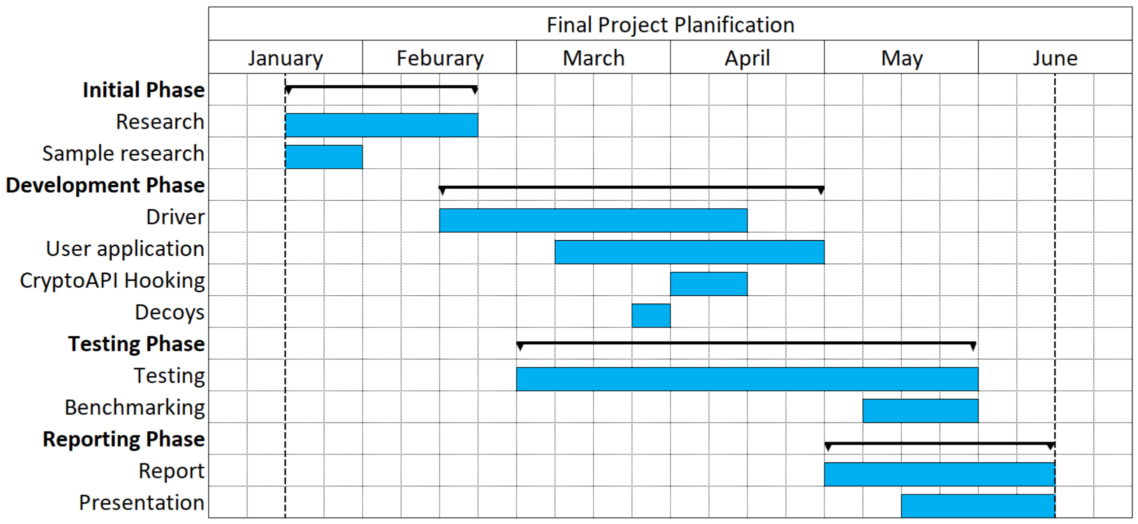


Figure 6.5: Final Gantt planning

Note that the Driver task inside the Development Phase is much longer than in the initial planification, this is because some new modifications to the driver were added and implemented during the final weeks of the task.

A more detailed comparison between the initial and final planification is shown in *Table 6.2.*

Task	Estimated Time (hours)	Real Time (hours)
Learning Phase	**95**	**95**
Research	85	85
Sample Research	10	10
Development Phase	**200**	**235**
Driver	70	80
User application	80	130
CryptoAPI Hooking	35	20
Decoys	15	5
Testing Phase	**50**	**50**
Testing	50	35
Benchmarking	0	15
Reporting Phase	**60**	**75**
Report	55	60
Presentation	15	10
Total	**405 hours**	**445 hours**

Table 6.2: Final Project dedication

At the end, the Learning Phase did not require more dedication yet the time to be completed was extended for the reasons commented in the following section. In the development phase, some tasks were discarded and the focus shifted to the user application and the process monitoring performed in it. Regarding the Testing Phase, the test of the product took less time than expected but in contrast the benchmarking task was added. Finally, the Reporting Phase took almost the same effort as expected at the start of the project although I needed more time to complete it.

6.2.3. Modification & Deviations from Initial Planification

The project planning is needed to determine which tasks are going to be developed and set deadlines in order to meet the goals. However, it is possible that due to unforeseen events or other factors, the development of the project is affected; therefore, the task planification has to be restructured.

In this section I am going to expand on the modifications and deviations to the initial planification and its primary reasons.

6.2.3.1. Initial Phase

Due to COVID-19 and some unforeseen events that forced me to step away from the project some time, the research took more time than expected.

This is the reason why in *Figure 6.5* the Research and the Initial Phase lasts longer than in the initial planification presented in *Figure 6.4*. Even when the number of hours is similar to the originally planned, it took more time to finish this task.

6.2.3.2. Development Phase

Due to the reasons mentioned in the previous paragraph, the Development Phase started later than expected. In addition, this phase was extended since some of the tasks took more time than planned originally, mainly the driver development.

Furthermore, some of the tasks that were initially going to be developed were discarded, specifically the CryptoAPI Hooking; for more details refer to section 9. In contrast some new functionalities were added, which caused a restructuration of the tasks and more time was invested in order to complete the new desired features.

6.2.3.3. Testing Phase

The Testing Phase was expanded with a new task, the Benchmarking of the product.

This new task was not planned during the initial planification since it was not considered at the start of the project. The Benchmarking task was mentioned during a meeting with the project director and we both considered this very important, hence it was added to the project scope.

6.2.3.4. Reporting Phase

The fact that some of the previous phases suffered from modifications and that the workload of the master was misestimated during the final weeks of the course, lead to increase the priority of the thesis report over the development in order to meet the deadline of the project.

This encouraged that some of the development tasks were left as a future work to have more time to produce a quality thesis report and avoid the rush of this task.

7. Budget

All projects have a cost, as well as this thesis. In this section I will estimate the economic cost this project would have in a real business environment. I will consider direct costs, indirect costs and a contingency budget.

7.1. Project Cost Estimation

7.1.1. Direct Costs

The direct costs are the expenses that are directly tied to a department or project. Direct costs comprehend the costs related to labour, materials, licenses, equipment, etc. In the thesis scope I am going to distinguish between human costs and material costs.

7.1.1.1. Human Costs

Human costs are the ones related to labour. This project has been developed by one student with the guidance of a professor. So, the human resources are a developer and a director or project manager.

As there is a single developer for this project, there is no point in performing a role separation among different tasks. The average salary of a developer is estimated around 18€/h and the average salary of the project director is estimated to be 35€/h.

Role	Estimated Hours	Estimated Salary per hour (€/h)	Estimated Cost (€)
Developer	445	18	8.010
Director	50	35	1.750
	Total:		**9.760**

Table 7.1: Human costs

7.1.1.2. Material Costs

In the material costs I included the Hardware and Software costs of the project. These costs include the laptop used to develop the project, and the software licenses for the non-free software, in this case VMWare Pro. The rest of software either has a free community version or has no cost.

Product	Price (€)	Quantity	Amortization period	Estimated cost (€)
MSI GF75 Thin	1199	1	3 years	166,52
Visual Studio 2019 Community	0	1	-	0
VMWare Pro	189,48	1	-	189,48
Windows Driver Kit (WDK)	0	1	-	0
		Total:		**356**

Table 7.2: Material costs

The total direct costs of the project result as the combination of the material costs and the human costs, obtaining the following estimation:

Type of cost	Estimated cost (€)
Human costs	9.760
Material costs	356
Total	**10.116**

Table 7.3: Direct costs

7.1.2. Indirect Costs

The indirect costs correspond to the costs that are not directly related with a single product but to the process of developing them. In other words, indirect costs are the costs needed to operate the business as a whole.

Some indirect costs are: office expenses, office rent, telephone expenses, commute costs and so on.

With the Work from Home policy most of this cost now can be omitted. For this project I will consider only the cost of electricity and internet as indirect cost.

Product	Price	Estimated Cost (€)
Electricity	0.28027 €/kWh	124,72
Internet	32 €/month	192,00
	Total	**316,72**

Table 7.4: Indirect costs

7.1.3. Contingency costs

As any project I must take into account possible deviations and risks that can affect the normal development of the project; therefore, I established a 10% margin over the global cost of the project as a contingency measure in order to deal with unforeseen events.

7.1.4. Final Budget

The total cost of the project is results as shown in the following table.

Type of cost	Estimated cost (€)
Direct costs	10.116
Indirect costs	316,72
Subtotal	10.432,72
Contingency measures (10%)	1.043,27
Total	**11.475,99**

Table 7.5: Total costs

8. Future Work / Improvements

During the lifetime of the project, the director and the author proposed multiple ideas to detect ransomware and enhance RanAware, additionally some issues appeared during development which require further investigation. Unfortunately, I did not have enough time to implement all of them.

Accordingly, this section presents several improvements that can be done to RanAware and some other ideas or extra modules that can be incorporated into the project.

8.1. Further testing on configuration parameters

To configure RanAware ransomware detection I used several different configurations of file renames per second, number of decoy files placed in the system, number of operations to log, etc.

Despite the current values being tested and producing good results, more testing could be done in order to slightly improve the effectiveness of RanAware.

8.2. Deep Analysis on RanAware Overhead

As mentioned in section 5.2.1, RanAware has some impact in the system's performance, particularly when it is under heavy workload. The results obtained for this test scenario show that RanAware affects the system significantly in some particular cases while in the rest of the cases the effects of RanAware are minor.

To further analyse the reason behind this behaviour of RanAware is an important task that we decided to leave as a future work due to not having enough time to investigate it.

8.3. Use the driver to stop operations of malicious processes

This functionality would involve the communication of the malicious processes detected by the application to the driver, this could be done with a new command sent through the communication channel already stablished.

When the driver receives information about a new offending process, stores it and starts blocking all the operations that are issued by this process with the objective of further reducing the effects of ransomware in the system.

This functionality was started during the development phase, unfortunately due to the complexity of the driver it was decided to leave this task for future developments of the project.

8.4. Analyse entropy of memory buffers

This improvement is aimed to attack the encryption property of ransomware. Its objective is to detect the encryption data on the memory buffer that contains the bytes that are going to be written into a file.

The driver would collect and send to the user application the address of the memory buffer containing the data to be written into a file. This way, RanAware user application would analyse the memory buffers computing the *Shannon* entropy of the bytes in the buffer. If RanAware obtains a high entropy it could determine that the bytes in the buffer probably represent encrypted data since the entropy of the bytes representing text in files is very low.

With this strategy, RanAware aims to detect encrypted data that is being written into a file. The objective is identifying processes that are doing an intensive use of cryptography and mark them as possible ransomware. What's more, this method would allow to detect custom encryption solutions since it is only analysing the data after the encryption has been performed and not the encryption method.

The downside of this strategy is that there are some types of data that possess also a high entropy, like compressed files, or benign encryptions; for this reason, RanAware should be careful with the type of files and data that is analysing in every moment.

8.5. Analyse binaries to find encryption traces in machine instructions

One of the ideas proposed by the director of the project is to detect the usage of encryption by analysing the machine instructions of the binaries.

By knowing which are the machine instructions that are commonly used to perform the cryptographic operations, it would be possible to compare this data with the instructions that a binary executes in order to detect the presence of encryption operations.

This could be used by RanAware to detect whether a process is performing encryption operations or not. However, due to the difficulty of this method, not only to obtain the machine instructions used by a process but to analyse encryption standard algorithms and learn which are the instructions used by them, this task was left as a feature for the future work.

All in all, this method would also allow to detect custom cryptography solutions that are common in recent ransomware.

8.6. Detect deletion of the volume shadow copies

A common operation of ransomware is to delete the volume shadow copies if these exist in the system. The shadow copies are created by Windows and are used to recover critical information in case that it is lost.

A very straightforward strategy is to monitor these volume shadow copies to detect any malicious process that attempts to delete them. An additional measure is the replication of these copies to avoid losing them if a malicious process manages to delete them. Both of these actions could be incorporated to RanAware.

8.7. Implement a ranking system of processes

This is an alternative system to classify processes that was proposed during RanAware development.

With this system, RanAware maintains a list of dangerous processes and assigns a score to each one of them. This score is incremented each time a process performs a suspicious operation, the more suspicious operation, the more points it increments.

Then with a defined threshold, each process score that meets that threshold is considered a malicious process, in this case a ransomware, and is terminated by RanAware.

Additionally, this system would make it even easier to incorporate more modules into RanAware.

8.8. Create an Installer along with a certificate for the driver

This is a Quality of Live improvement (QoL) that I believe is necessary before introducing RanAware to a production environment.

The first thing to do is to create an installer that configures the environment and installs the RanAware components into the system.

In order to have a complete product and to be able to install the driver without problems it is also necessary to obtain a certificate from Microsoft to be able to digitally sign the driver so it is trusted by the system. Unfortunately, there is a monetary price that it will only be considered when the product is ready for production deployment.

8.9. Use IRP data to feed machine learning algorithms

The data obtained with the driver opens the door for a lot of possibilities.

As found in the research, this data could be used to profile the behaviour of legitimate applications and ransomware at IRP level. With the operations logged, a machine learning algorithm can be trained to distinguish between a benign process or a ransomware based on the IRP data that is collected by the driver.

8.10. Detect Ransomware notes

Another approach to detect ransomware is to monitor the creation of ransom notes.

These files represent a very common trait of ransomware. They are usually created in every directory of the system after the legitimate files are encrypted. In the ransom note, the victim is informed that the data in the computer has been encrypted and a ransom is asked, usually in crypto-currencies, accompanied by an address to pay. Often there is also a limited period of time to pay the ransom.

Since the content of these files is very characteristic, some rules or even a machine learning algorithm could be defined to detect the creation of these files.

9. Considered Alternatives

This section describes some ideas that were considered to be developed but after research and investigation they were discarded.

9.1. CryptoAPI Hooking

This was the first thing that came to my mind when I was thinking of methods to detect ransomware.

The CryptoAPI is the API offered by Windows to perform cryptographic operations. By hooking the function calls of the CryptoAPI, it would be possible to detect which process is calling them and in addition, to backup the keys used so the encrypted data could be recovered.

However, this solution had several issues that ultimately lead to discard it as a module of RanAware, even when the functionality was almost implemented.

This first problem is that in order to hook the DLL in user mode, it was necessary either to implement the hook for all of the processes in the system one by one or to hook it at start-up to all processes that loaded the "User32.dll". Both of these solutions would cause RanAware to log cryptographic operations of benign processes.

On the other hand, the hooking could be performed at kernel level, nevertheless, this approach resembled more the behaviour of a malware than a benign application and it was discarded.

The second problem of this solution is that after logging the keys used to encrypt data, these keys need to be related to the writing operations recorded by the driver in order to know which key has been used to encrypt each file. This task could be simple if the ransomware uses only one key to encrypt the whole system or tremendously challenging if the ransomware uses different keys to encrypt the files, that is the most common approach.

Ultimately, most of the recent ransomware do not use the APIs to perform the cryptographic operations. The most common approach is that each ransomware implements or uses its own version of the cryptographic algorithms. This renders useless the hooking of the cryptographic API of Windows considering that the only processes that use it are benign processes.

10. Conclusion

Upon considering all the facts, I can confidently say that I achieved the objective of the project, RanAware is able to perform an early detection of ransomware and terminate the attack. Additionally, I was able to measure the impact that RanAware has in the system.

Considering the impact that RanAware has on a system, I believe that it is a fair price to pay for effectiveness of the protection mechanism.

Although this project has a lot of room for improvement, I believe that the results obtained with RanAware are satisfactory and very promising. On top of that, I was able to learn a lot about ransomware, Windows system and driver development.

I also want to remark that RanAware is not a standalone solution that can operate on its own; RanAware is meant to be a complementary specialized module to detect ransomware. Therefore, it should never be used as the only security measure of a computer since it only focuses on one type of malware among the myriad of malware types there are nowadays.

References

[1] A. Alqahtani and F. T. Sheldon, "A Survey of Crypto Ransomware Attack Detection Methodologies: An Evolving Outlook," *Sensors 2022, Vol. 22, Page 1837*, vol. 22, no. 5, p. 1837, Feb. 2022, doi: 10.3390/S22051837.

[2] N. Hampton, Z. Baig, and S. Zeadally, "Ransomware behavioural analysis on windows platforms," *Journal of Information Security and Applications*, vol. 40, pp. 44–51, Jun. 2018, doi: 10.1016/J.JISA.2018.02.008.

[3] A. Palisse, H. le Bouder, J. L. Lanet, C. le Guernic, and A. Legay, "Ransomware and the legacy crypto API," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10158 LNCS, pp. 11–28, 2017, doi: 10.1007/978-3-319-54876-0_2.

[4] G. McDonald, P. Papadopoulos, N. Pitropakis, J. Ahmad, and W. J. Buchanan, "Ransomware: Analysing the Impact on Windows Active Directory Domain Services," *Sensors*, vol. 22, no. 3, Feb. 2022, doi: 10.3390/s22030953.

[5] A. Arabo, R. Dijoux, T. Poulain, and G. Chevalier, "Detecting Ransomware Using Process Behavior Analysis," *Procedia Computer Science*, vol. 168, pp. 289–296, Jan. 2020, doi: 10.1016/J.PROCS.2020.02.249.

[6] D. Nieuwenhuizen, "A Behavioural-based Approach to Ransomware Detection." https://labs.f-secure.com/archive/a-behavioural-based-approach-to-ransomware-detection/ (accessed May 04, 2022).

[7] Monika, P. Zavarsky, and D. Lindskog, "Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and Characterization," *Procedia Computer Science*, vol. 94, pp. 465–472, Jan. 2016, doi: 10.1016/J.PROCS.2016.08.072.

[8] A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, "Extinguishing ransomware - a hybrid approach to android ransomware detection," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10723 LNCS, pp. 242–258. doi: 10.1007/978-3-319-75650-9_16.

[9] D. Gonzalez and T. Hayajneh, "Detection and prevention of crypto-ransomware," *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2017*, vol. 2018-January, pp. 472–478, Jul. 2017, doi: 10.1109/UEMCON.2017.8249052.

[10] S. H. Kok, A. Abdullah, N. Z. Jhanjhi, and M. Supramaniam, "Prevention of crypto-ransomware using a pre-encryption detection algorithm," *Computers*, vol. 8, no. 4, Dec. 2019, doi: 10.3390/computers8040079.

[11] R. Moussaileb, N. Cuppens, J. L. Lanet, and H. le Bouder, "A Survey on Windows-based Ransomware Taxonomy and Detection Mechanisms: Case Closed?," *ACM Computing Surveys*, vol. 54, no. 6. Association for Computing Machinery, Jul. 01, 2021. doi: 10.1145/3453153.

[12] A. Continella *et al.*, "ShieldFS: A self-healing, ransomware-aware file system," in *ACM International Conference Proceeding Series*, Dec. 2016, vol. 5-9-December-2016, pp. 336–347. doi: 10.1145/2991079.2991110.

[13] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A real-time detection system against cryptographic ransomware," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11050 LNCS, pp. 114–136. doi: 10.1007/978-3-030-00470-5_6.

[14] "Volume Shadow Copy Service | Microsoft Docs." https://docs.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service (accessed May 05, 2022).

[15] "Windows Kernel-Mode I/O Manager - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-i-o-manager (accessed May 05, 2022).

[16] "Overview of the Windows I/O Model - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/overview-of-the-windows-i-o-model (accessed May 05, 2022).

[17] "IRP Major Function Codes - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-major-function-codes (accessed May 05, 2022).

[18] "IRP_MJ_CREATE - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-create (accessed May 05, 2022).

[19] "IRP_MJ_CLOSE - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-close (accessed May 05, 2022).

[20] "IRP_MJ_READ - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-read (accessed May 05, 2022).

[21] "IRP_MJ_WRITE - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-write (accessed May 05, 2022).

[22] "IRP_MJ_QUERY_INFORMATION - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-query-information (accessed May 05, 2022).

[23] "IRP_MJ_SHUTDOWN - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-mj-shutdown (accessed May 05, 2022).

[24] "File systems and filter driver design guide - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/ (accessed May 05, 2022).

[25] "Filter Manager Concepts - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts (accessed May 05, 2022).

[26] "Load order groups and altitudes for minifilter drivers - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers (accessed May 05, 2022).

[27] "Communication Between User Mode and Kernel Mode - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/communication-between-user-mode-and-kernel-mode (accessed May 05, 2022).

[28] "Git." https://git-scm.com/ (accessed May 20, 2022).

[29] "GitHub." https://github.com/ (accessed May 20, 2022).

[30] "Visual Studio: IDE y Editor de código para desarrolladores de software y Teams." https://visualstudio.microsoft.com/es/ (accessed May 20, 2022).

[31] "Download the Windows Driver Kit (WDK) - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/download-the-wdk (accessed May 20, 2022).

[32] "Download VMware Workstation Pro." https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html (accessed May 20, 2022).

[33] "Debugging Using WinDbg Preview - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-using-windbg-preview (accessed May 20, 2022).

[34] "4d61726b/VirtualKD-Redux: VirtualKD-Redux - A revival and modernization of VirtualKD." https://github.com/4d61726b/VirtualKD-Redux (accessed May 20, 2022).

[35] "DebugView - Windows Sysinternals | Microsoft Docs." https://docs.microsoft.com/en-us/sysinternals/downloads/debugview (accessed May 20, 2022).

[36] "Process Monitor - Windows Sysinternals | Microsoft Docs." https://docs.microsoft.com/en-us/sysinternals/downloads/procmon (accessed May 20, 2022).

[37] "Cracked5pider/conti_locker: Conti Locker source code." https://github.com/Cracked5pider/conti_locker (accessed May 30, 2022).

[38] "conti leaks (@ContiLeaks) / Twitter." https://mobile.twitter.com/contileaks (accessed May 30, 2022).

[39] "Load order groups and altitudes for minifilter drivers - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers (accessed May 21, 2022).

[40] "IRP_MJ_SET_INFORMATION (IFS) - Windows drivers | Microsoft Docs." https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/irp-mj-set-information (accessed May 21, 2022).

[41] "Process Security and Access Rights - Win32 apps | Microsoft Docs." https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights (accessed May 30, 2022).

[42] "goliate/hidden-tear: ransomware open-sources." https://github.com/goliate/hidden-tear (accessed May 29, 2022).

[43] "ANY.RUN - Interactive Online Malware Sandbox." https://any.run/ (accessed May 29, 2022).

[44] "Malwarebytes Cybersecurity for Home and Business | Anti-Malware & Antivirus." https://www.malwarebytes.com/ (accessed Jun. 08, 2022).

[45] "OSR Developer Community." https://community.osr.com/ (accessed May 28, 2022).

[46] "Downloads:Driver Loader." https://www.osronline.com/article.cfm%5Earticle=157.htm (accessed May 28, 2022).

Appendix A: Installing the driver manually

Manually installing a driver in Windows is not a trivial task. In this appendix I will describe the process to generate and install the RanAware driver in a computer.

The first step is to have all the necessary tools installed in order to compile the project. These are the Windows Driver Kit (WDK) and Visual Studio 2019 Community Edition.

The compilation can be done with Visual Studio, after opening the project just click the "Build" tab and then "Build Solution". The project can be built for the x86 and x64 platforms, even though the majority of the testing has been performed in a x64 environment. If there is any dependency that is not installed in the computer, the compilation will fail. The most important thing is to install the WDK.

After compiling the project, all the executables and necessary files will be generated. In particular for the driver, a certificate (*RanAware_FS_Filter.cer)*, an *inf* file (*RanAware_FS_Filter.inf*) and a system file (*RanAware_FS_Filter.sys)*. These files are the ones needed to install the RanAware driver.

As the driver is not digitally signed by Microsoft, we need to boot the system in *Test* mode so we are able to install the driver. To boot Windows in *test* mode we can execute the following command in an administrator command line and then reboot the system.

```
> bcdedit /set testsigning on
```

If everything is correct, the bottom left corner of the screen should indicate that we are in Testing mode. To disable the testing mode, execute the following command.

```
> bcdedit /set testsigning off
```

Once the system is in Test mode, we need to install the *inf* file. To do this right-click on the file and click "install". After the installation we need to execute a command in a privileged command line to start the driver. The command to start the driver is:

```
> net start RanAware_FS
```

If everything goes well, we should get a message informing that RanAware is running.

That is all to load and run the driver from scratch.

To stop the driver from running we can use the following command:

```
> net stop RanAware_FS
```

There is an alternative to load and execute drivers that is a tool developed by a driver developer community called *OSR Developer Community* [45]. The tool to load drivers is called *Driver Loader* [46] and it is a really intuitive and easy to use.

Even though it should not happen, Windows may still complain about the driver signature, if this happens the driver signature enforcement must be disabled. To do this there are two alternatives:

Execute the following command:

```
> bcdedit.exe - set loadoptions DISABLE_INTEGRITY_CHECKS
```

If the previous command does not work properly, perform this follow the steps:

1. Restart the computer and press **F8** key until the **Advanced Options** menu pops up.
2. Choose **Troubleshoot > Advanced Options > Startup Settings** and click **Restart**.
3. When the computer restarts select **Disable driver signature enforcement**.

To enable the driver signature just reboot the system or execute the following command if the option remains disabled:

```
> bcdedit.exe - set loadoptions ENABLE_INTEGRITY_CHECKS
```

Appendix B: Benchmarking data summary

In here I present a summary of the data used to create the boxplots for the benchmarking.

The data represents the number of seconds that it took to complete writing the contents of a determined number of files.

Configuration	Low Load							
	No Driver				Driver			
Nº of Files	**10F**	**100F**	**1000F**	**10000F**	**10F**	**10F**	**100F**	**1000F**
Minimum	0.000474	0.004788	0.046516	0.479982	0.000447	0.004562	0.049075	0.504155
First Quartile	0.000484	0.004956	0.048062	0.501878	0.000486	0.004843	0.050550	0.521153
Median	0.000516	0.005051	0.048806	0.517065	0.000503	0.005254	0.051695	0.533063
Third Quartile	0.000558	0.005445	0.050494	0.535912	0.000544	0.005598	0.053088	0.544105
Maximum	0.001435	0.010503	0.092646	0.841232	0.001452	0.010029	0.094135	0.833065
Mean	0.000541	0.005450	0.049885	0.528135	0.000539	0.005319	0.052631	0.536369
Mean (No Outliers)	0.000523	0.005190	0.049238	0.519502	0.000509	0.005223	0.051827	0.532631
IQR	0.000074	0.000489	0.002433	0.034034	0.000058	0.000755	0.002537	0.022952
Lower Limit	0.000374	0.004222	0.044413	0.450828	0.000400	0.003710	0.046744	0.486725
Upper Limit	0.000668	0.006178	0.054143	0.586963	0.000631	0.006731	0.056893	0.578532

Table B.1: Low load benchmark data

Configuration	Medium Load							
	No Driver				Driver			
Nº of Files	**10F**	**100F**	**1000F**	**10000F**	**10F**	**100F**	**1000F**	**10000F**
Minimum	0.000493	0.006537	0.075327	0.890066	0.000544	0.006677	0.077751	0.843604
First Quartile	0.000559	0.009194	0.089507	0.948992	0.000652	0.009042	0.087720	0.919688
Median	0.000638	0.011487	0.094511	0.985603	0.000808	0.010645	0.091640	0.950433
Third Quartile	0.001066	0.015131	0.105239	1.030136	0.001197	0.013096	0.103526	1.014221
Maximum	0.005158	0.040136	0.212306	1.283693	0.005123	0.027313	0.198929	1.650289
Mean	0.000914	0.013529	0.099564	1.002490	0.001060	0.011674	0.098794	0.979104
Mean (No Outliers)	0.000770	0.012330	0.096632	0.991071	0.000901	0.011029	0.095091	0.961036
IQR	0.000507	0.005937	0.015732	0.081144	0.000546	0.004054	0.015806	0.094532
Lower Limit	-0.000202	0.000288	0.065910	0.827276	- 0.000167	0.002961	0.064012	0.777889
Upper Limit	0.001827	0.024037	0.128837	1.151853	0.002015	0.019177	0.127235	1.156019

Table B.2: Medium load benchmark data

Configuration	Heavy Load							
	No Driver				Driver			
Nº of Files	10F	100F	1000F	10000F	10F	100F	1000F	10000F
Minimum	0.000452	0.006379	0.078969	1.176225	0.000566	0.006726	0.097987	1.137752
First Quartile	0.000593	0.008013	0.109869	1.363849	0.000675	0.008820	0.142358	1.565544
Median	0.000685	0.009990	0.124373	1.530483	0.000848	0.011306	0.177100	1.801511
Third Quartile	0.000964	0.014313	0.143671	1.716171	0.001219	0.024414	0.211381	1.948201
Maximum	0.015328	0.073986	0.248035	2.117679	0.036724	0.067437	0.369730	2.522436
Mean	0.001083	0.013919	0.127550	1.545477	0.001690	0.019045	0.183963	1.777846
Mean (No Outliers)	0.000763	0.011184	0.125153	1.535279	0.000971	0.015727	0.179759	1.773325
IQR	0.000371	0.006300	0.033802	0.352322	0.000544	0.015594	0.069023	0.382657
Lower Limit	0.000037	-0.001437	0.059166	0.835367	-0.000142	-0.014571	0.038824	0.991558
Upper Limit	0.001520	0.023763	0.194374	2.244653	0.002035	0.047804	0.314915	2.522187

Table B.3: Heavy load benchmark data

Glossary

- IRP: Input/Output Request Packet.
- C2: Command and Control.
- RSA: Rivest, Shamir and Adleman.
- AES: Advanced Encryption Standard.
- API: Application Programming Interface.
- I/O: Input/Output.
- FltMgr: Filter Manager.
- IDE: Integrated Development Environment
- WDK: Windows Driver Kit