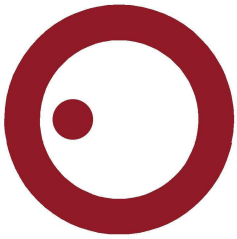


escola superior de tecnologia e gestão
instituto politécnico de leiria

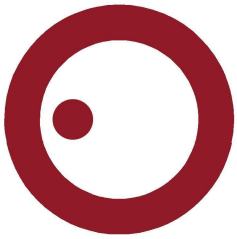
Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Master in Cybersecurity and Digital Forensics

INTERFACE GRÁFICA PARA A FERRAMENTA FORENSE DE ANÁLISE DE RAM

Volatility GUI : Simplifying Memory analysis

STUDENT ALEXANDRE DE SOUSA MONTEIRO

Leiria, November 2021



escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Master in Cybersecurity and Digital Forensics

INTERFACE GRÁFICA PARA A FERRAMENTA FORENSE DE ANÁLISE DE RAM

Volatility GUI : Simplifying Memory analysis

STUDENT ALEXANDRE DE SOUSA MONTEIRO
Number: 2190231

Project conducted under the guidance of the Professor Doutor Luís Alexandre Lopes Frazão (luis.frazao@ipleiria.pt).

Leiria, November 2021

ACKNOWLEDGEMENTS

A special thank you to professor Doctor Luís Frazão not only for his help during the entire development of this project and the many fresh and new ideas he brought to the project. Without his orientation, this project would have never come to fruition.

Another special thank you to professor Baltazar Rodrigues for being the proponent of this project and the knowledge delivered during the first year of the cybersecurity masters. Without his idea, this project would never have even been under my radar.

Thank you to my colleagues and friends for the help testing my project and the inspiring words during this challenging year.

Last but not least important a deep thank you to my partner Cláudia Ferreira for her comprehension during these challenging years and all the understanding and help that she provided. Without her, this project would take many more months to complete.

ABSTRACT

Digital forensic analysis is an area generally restricted to a small number of individuals. This happens due to the complexity of entry to the area due to a small user base, few and out-of-date software restricting the more advanced software behind paywalls and licenses. Unfortunately, due to the small user base, open-source projects take a long time to be updated and fall behind software present in other areas.

More and more crimes are being committed using digital means, which means that the ability to analyze the physical computer and its components is becoming increasingly more important. The Volatility Framework is used for the analysis of one of these components, the RAM. But the currently available software is far from perfect due to a lack of graphical interface and other functionalities that have come to be the norm for software in the year 2021. This means that the end-user needs to be even more experience to use this software.

This thesis has as its main objective the development of a graphical interface to the software Volatility. This graphical interface takes as its main objective to ease the usage of Volatility as much as possible. This is accomplished by introducing automatic commands, reports, more accessible exports, integrations, functionalities that had to be done outside of the tool, among many others. This is done to help the final user that might have very little knowledge and do a complete RAM images analysis without having to use any external tool.

The results obtained by the development of the software allow the proof of effectiveness, in levels of time and usefulness, of this tool with new users of this software and this area, and with more experienced users.

CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AES256	Advanced Encryption Standard 256.
API	Application Programming Interface.
DC3	Department of Defense Cyber Crime Center.
DLL	Dynamic Link Libraries.
DoD	Department of Defense.
GUI	Graphical User Interface.
IR	Incident Response.
IT	Information technology.
MD5	Message-Digest algorithm 5.
OS	Operating System.
QoL	Quality of Life.
RAM	Random-access memory.
SHA1	Secure Hash Algorithm 1.
SHA256	Secure Hash Algorithm 256.
USB	Universal Serial Bus.

1 INTRODUCTION

In today's world cybercrime is evolving at a much faster rhythm than before using new technology to its advantage and tricking even the more expert users. To fight this it's necessary to invest in cybersecurity solutions and to train people to be able to handle this kind of task.

Data from 2020 from one of the biggest developers and sellers of cybersecurity products tells us that cybercrime costs $2.9 million every minute and the use of malware increased by 358% in this year. Predicting an increased number of devices connected to the internet every year we can also expect this number to rise exponentially. This means that more than ever it's necessary to introduce the tools necessary to possibly resolve these problems to the maximum number of users while keeping more experienced users with updates to keep the demand for new functionalities met.[8]

A big part of cybersecurity is forensic investigation. This field of expertise normally requires very specific tools and a steep learning curve meaning that not many users have the patience to enter this field. A big challenge is the introduction of new users to this area due to the reasons previously mentioned, this translates to a small user base and fewer updates to existing tools to auxiliate in the simplification of the usage of more experienced users. This is the big challenge of this field! To have software that can keep up with today's demands while being accessible to new users and this problem it's what will be discussed about in ore detail in next point.[8]

1.1 OBJECTIVES & CONTRIBUTIONS

This software was created with the intent of introducing the memory analysis area to new users without the barrier of knowledge entry that was previously existing. This Graphical User Interface (GUI) for Volatility Framework creates the possibility for the user to have the ability to use it without know any particular knowledge of this tool or its commands to get the information it requires. This software was also created to help existing users of this area with built-in functionalities to assist

in tasks that previously would have to be done with other tools, with an interface to facilitate its usage and the ability to automatically get complete reports with an analysis of the Random-access memory (RAM) image. Despite the existence of other projects similar to this one, they all lack something that in my opinion is of importance to introduce new users to this kind of technology, that will be detailed further down this document.

The objective of this project is to be able to develop a GUI for one of the available software in the area of memory analysis that was easy to install and able to use without a lot of previous knowledge. Another objective of this project is to give more experienced users a better way to work with the chosen software and to streamline the workflow process for this type of users. The contributions on this project are:

- Easy Installation / Portable Installation
- Ability to save & load VolatilityGUI projects
- Creation of customized reports
- Easiness of loading RAM images
- Easiness of running Volatility commands
- Keep multiple command results available
- Antivirus integration
- Ability to extract select files
- Search strings in results
- Tutorial of all available commands
- Ability to run commands that require variables through selection boxes
- Ability to associate processes with user and detect the presence of malware or network connections

1.2 REPORT STRUCTURE

This report is divided into 6 chapters. Starting on the second chapter, an overview is presented about Forensic Analysis on RAM Memory, about the existing tools of this area and how they differ from one another, including the advantages of each one. After this, it will be presented how Volatility is used for this project including the reason why it was chosen.

The third chapter is where we can find everything from the process of developing the GUI. In it, the architecture of the solution and the idea of how it could work are both explained. The planning behind the solution and how it thought and came to be, are also mentioned in this chapter. Next is presented the methodology of the development, including the user stories that have been created at the beginning of the project and the functional requirements. Since this project is the development of a GUI, mockups have been drawn to get an idea of the design of the software. And finally, this chapter ends with a brief presentation of very simple data models of how the tool works.

In the fourth chapter of the report, the development of the software is presented with a description of every functionality of the software that has been implemented in detail and also mentions every choice that has been taken and the reason behind it.

After this, chapter five is reached where it's presented both Load Tests that have been performed on the tool and also tests of usage with users. The tests with users will be both with the satisfaction of the usage of the tool and also a type of test comparing the performance on the software with and without the usage of the GUI. After the presentation of the results, we will do a critical analysis of the same results and make some conclusions from these results.

And finally, the report finishes with the sixth chapter with a conclusion and description of the future work where it is mentioned all of the improvements that could be applied and also new ideas for the developed software.

2

FORENSIC ANALYSIS OF RAM MEMORY

This chapter will discuss the concept of Forensic Analysis of RAM Memory with a mention of the tools that are currently used to do this type of analysis. While also doing a brief presentation of some of the available software and its capabilities for the task of analyzing RAM images. This chapter will explain the concept of memory analysis, volatile memory, volatile data, and memory dumps.

It will also be possible to see a comparison of some of the most used memory analyzing software and a discussion in detail of these software pros and cons, with a deep dive into the software used for this project and mention what makes Volatility one of the best software in the memory analysis area. Then, the chapter will continue with a deep dive into the contributions that have been made with the developed work.

2.1 WHAT IS IT

Forensic Analysis of digital devices is a very recent area of study that is becoming more relevant each day. This is because of the necessity of keeping up with the technology used by the general population, meaning that more and more criminal investigations pass by the examination of digital artifacts. RAM Memory forensics consists of capturing volatile data from a device's RAM and using this method's information to auxiliary the investigation of incident response, malware analysis, or simply for its digital forensic capabilities. Vital information can be taken from a RAM Image. To obtain any RAM Image it is necessary to capture the memory data of either a live system or a dead system. Despite not going into much detail about the part of capturing this data, it's necessary to know that this is probably the most crucial step in the entire investigation since, a poorly done capture done can ruin the whole investigation. The contents of the image depend on the previous explained step. In a typical investigation, this image is taken from a live system. It contains information about processes, Dynamic Link Libraries (DLL), general files, process memory, image identification, kernel memory and objects, networking, registry, and malware.[2] [4]

Before moving forward, a simple explanation about what is volatile memory, in contrast to Non-volatile memory, like a hard drive or a flash memory, also known as Universal Serial Bus (USB) Pen Drive, a volatile type of memory, is computer memory that requires an electrical current to retain data. So when the power is shut down for any reason, the data contained in this kind of memory is erased. This is known as volatile data, and the most common volatile memory is RAM.

Volatile data is the data that the volatile memory stores, which is constantly changing. Because of this, it's impossible to work with RAM data directly. Therefore, data stored in RAM can only be accessed as an image through a memory dump.

Memory dumps are files that contain a copy of a computer's volatile memory at a specific time of the capture. A memory dump can be used to find information about running programs and the operating system itself. This can be useful for the following purposes:

- Gather diagnostic information;
- Find the source of a system crash;
- Find the reasons for memory leaks, bugs, and errors;
- Digital forensics on suspicious activates (for example malware);

Next, it will be explained in detail each type of file that a memory dump can contain and what it represents:

- Processes

All currently active processes are stored in RAM, and they can be recovered from the data structure in which they are located. Malware can also create hidden processes that cannot be traced by standard methods or by simply reviewing active processes. These hidden processes can be complicated to find but not impossible. These kinds of processes can change information about themselves and even disguise as other processes. This type of malware requires a profound knowledge of the operating system's functioning to understand how it works. Within processes, it's also possible to find processes that have been stopped if the device was not powered off from the moment when a certain process was closed. If the environment has not yet assigned that space to another process, and if this is the case, it's possible to see some information about closed processes.[18]

- DLLs

Information about DLLs used by the operating system can be of great importance. In case the process is part of a malware code this can point to outsource processes

of where the code itself exists and if it contains malware. It can also show if data is possibly being sent to any external network or if files were changed. These types of files are exclusive to the Windows environment, meaning that they don't exist in a RAM image of another operating system. [3]

- General files

All files that have been either opened, read, or modified have to pass through the RAM, meaning that more information can be taken for the investigation. The normal behavior of the RAM with a file is to create a copy of the file in memory while it's being edited. Meaning that it is possible to recover the full file or fragments of it from the RAM. This all depends on the size of the RAM and the size of the files. Sometimes these files can have great importance to the investigation. Furthermore, copied and pasted files can stay in the RAM when stored in the clipboard, which can be recovered to help the investigation.[5]

- Networking Info

It's also possible to analyze the RAM to get information about connections, open ports, local sessions, and remote sessions. This can be useful for the investigator because a lot of malware code depends on these connections to send information or to do code injections. This information can also be beneficial because it's almost impossible to hide open connections on rogue processes. This kind of information can be beneficial to identify networking details about the malware. [1]

- Internet Data

The "Internet Data" that can be retrieved from the RAM is anything from browser downloads, messaging apps conversations, emails, and others. This information can be extremely beneficial due to the nature of the information that passes through this kind of channel.[1]

- Passwords and Keys

One big advantage of analyzing the RAM is the possibility to recover passwords and cryptographic keys, which can be useful to recover information relevant to the investigation. Normally, the passwords and cryptographic keys are never stored in the hard drive of the computer without any additional protection, but when it comes to the RAM when these kinds of passwords and keys are used they, are stored in the RAM until other data overwrite them. And by this logic, the investigator can find passwords to many types of accounts that can be relevant to the investigation.[18]

- Decrypted Files

When using passwords and cryptographic keys these are normally used to assess accounts but they can also be used to decrypt files. As explained before, when a file is being used, the RAM keeps a copy of this file, but when a previously encrypted file is accessed using its encryption key or password, a copy of this file is stored in the RAM unencrypted. And by analyzing the RAM is possible to recover vital information to the investigation of files or fragments of files that were previously encrypted, like encrypted files. [5]

- Other data

In Figure 1, it's possible to see how this data is stored inside the RAM.

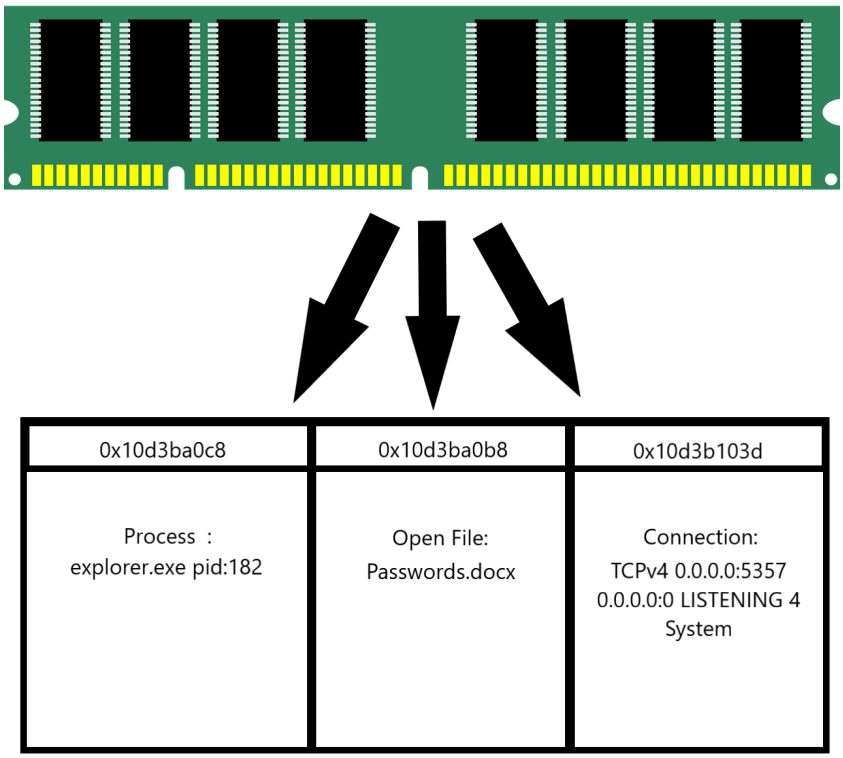


Figure 1: RAM Diagram

There are many other types of data that can be retrieved from the RAM. Other relevant data can be operating system event logs. In the case of Microsoft Windows, this event log can help establish a timeline of events in the Operating System (OS).

This is especially useful in a system with multiple users during an investigation. Event logs can also include anything from system related events to the connection of new external devices and many others. These event logs exist in multiple other operating systems like Linux and OSX. Normally, these event logs are very specific to each OS, but other data like processes and passwords work similarly in all operating systems. Furthermore, data can also be retrieved from virtualized environments. In this case, it is straightforward to retrieve the RAM for analysis because these files exist on the host machine that is being used to virtualize the virtual machine. Due to this, a file already exists in the form of a virtual RAM that can be easily extracted. [16]

This information can be important to an investigation cause it represents real-time data about the operating system environment, which can tell us a lot of useful information about the investigation in question. After the acquisition of the RAM image, it's necessary to use specific software to analyze this data. This software is going to be analyzed in the next sub-chapter.[11] [16]

2.2 REVISION OF EXISTING TOOLS & INNOVATIONS

As explained previously, a significant problem in this area is the user base of users is very limited in terms of capacity. However, despite this, there still exists some software capable of handling the task of analyzing RAM Images, but when compared to other areas, it's noticeable the lack of tools and options.

Starting with probably the most well known tool, Autopsy is a free GUI-based open source digital forensic program capable of analyzing everything from hard drives to mobile devices and much more. Despite the capabilities to analyze RAM Images being dependent on an external tool (Volatility Framework), this tool still offers a user-friendly way to import RAM Images and automatically run the most important commands. This feature is still very recent to Autopsy, and it's still in development. Despite this, it takes advantage of one of the most interesting functionalities of autopsy is the automatic flags in the "Interesting Files" category. With this, Autopsy can automatically find files that trigger alerts simplifying the work that needs to be done by the investigation. Despite this tool not being made for just the purpose of RAM analyses, it provides useful information to the user.[10][16]

The following tool to be presented is called MANDIANT Memoryze. This tool's available for free, and it's capable of not only analyzing RAM Image files but also capable of analyzing live systems RAM. These tool main functionalities are the

ability to image the full range of system memory, image a process entire address space to disk, including a process loaded DLLs, EXEs, heaps, stacks, image a specified driver, or all drivers loaded in memory to disk, enumerate all running processes and report all open handles in a process.[13]

One of the more advanced tools is wxHexEditor. This software is an open-source, cross-platform hex editor not explicitly used for RAM Image files but can serve this purpose. This software has the advantage of being very fast, with the functionality to open massive files in a fast way. This tool is very different from the other presented until now since, for any RAM analyses, we would have to interpret the files Hex meaning a profound knowledge in this theme. Because of this, we will not dive much further than this on this tool and acknowledge its existence.[22]

Rekall Forensics is an open-source network forensic analysis tool with a console interface. This tool started as a memory forensic framework, but it's currently an advanced forensic and incident response framework that presents similar capabilities to previously presented tools. The main advantage of this tool is the capability to detect profiles for different operating systems automatically. Another particular point with Rekall is Rekall Agent, which is an endpoint incident response and forensic tool based on modern cloud technologies that can be deployed from small to large scale enterprises to provide unprecedented visibility of endpoint security and collection and preservation of volatile endpoint evidence.[17]

MemGator, an interface-based tool, is a conglomerate of other tools, including Volatility Framework, Scalpel File Carver, and AESKeyFinder. For the theme of this project, this tool works very similarly to the tool previously presented Autopsy, in which it uses the Volatility Framework to present results from the RAM Image files. Despite this, the results provided by Volatility can be analyzed automatically by one of the other tools present in MemGator to search for usernames and passwords for Gmail, Hotmail, Yahoo, Facebook, and auto fill form entries for the Chrome web browser.[12]

Other software was not considered for this comparison due to being very similar to the ones presented in terms of functionalities. And due to being less recognizable than the ones shown or due to being very complete suites of tools where the RAM analysis is a very small part of the complete tool, where RAM analysis is not the focus of the tool's functionalities. Tools like 'Paladin Forensic Suite' is one of these cases, where its a very complete suite of tools where the focus is not on RAM analyses functionality. Also, because in terms of RAM analyses works very similar to another tool presented before, Autopsy. Because of this, it was decided only to show

one tool of this type to avoid any kind of repetition on the available software. The demonstration of other tools of this type wouldn't contribute to anything relevant to this work due to being very similar, in terms of RAM analyses, to the one already presented, Autopsy.

Finally, we have Volatility Framework that is the software that this work was based on, and because of this, we will do an extensive description of this tool in the next sub-chapter.

2.3 VOLATILITY FRAMEWORK

Volatility is an open-source framework used for memory forensics and digital investigations. The Volatility Framework is currently one of the most popular tools for volatile memory analysis. This cross-platform framework allows you to work with images of volatile memory, analyze them and obtain data from memory dumps. The framework inspects and extracts the memory artifacts of both 32-bit and 64-bit systems. The framework has support for all distributions of Linux, Windows OS, macOS, and Android. It can analyze raw memory dumps, crash dumps, virtual machine snapshots, VMware dumps, Microsoft crash dumps, hibernation files, virtual box dumps, and many others.[21] [6]

In terms of being open-source, the volatility framework is under the general license of Open Source GPLv2, which means that it can be read, learned, and extended. This is the main reason why many other tools use the volatility framework as a tool for memory analysis and also the reason why this work was possible without the imposition of many licensing rules. This was done so that it was easier to *' help yourself become a smarter analyst'* and to gather the collective knowledge of the community to publish new plugins and to fix any issues that may appear. Because of the small user base, this is a much needed resource, and many other products similar to Volatility stop receiving support because of this. [21] [6]

Volatility is developed in the programming language Python because, according to the volatility foundation, it is an established forensic and reverse engineering language with loads of libraries that can be easily integrated, and it's a language that most analysts are familiar with.[21] [6]

Volatility runs on almost all "normal" operating systems, including Windows OS, Linux, and macOS. The only requirement is that the operating system can run Python. This is a big advantage due to not being locked behind a specific operating

system like other software, making it accessible to almost all current operating systems.[21]

An extensible and scriptable Application Programming Interface (API) is a big part of what keeps this tool alive to this day. Because of the API available in Volatility, it's easy to develop customized interfaces and develop new features, including malware sandbox, perform virtual machine introspection or just explore kernel memory.[21]

Volatility is also used by many other tools, like Autopsy, because it was an Unparalleled feature set based on reverse engineering and specialized research. Volatility can get command histories, console input/output buffers, USER objects (GUI memory), and network related data structures.[21]

The Volatility Framework can analyze almost all types of files related to volatile memory, and this includes dumps, crash dumps, hibernation files, VMware .vmem, VMware saved state and suspended files (.vmss/.vmsn), VirtualBox core dumps, LiME (Linux Memory Extractor), expert witness (EWF), and direct physical memory over Firewire.[21]

Another great advantage of Volatility is its velocity and efficient algorithms that can analyze large files in just a few seconds. Of course, this depends a lot on the available resources on the host machine and on the command that was requested, but usually, it is much faster than other frameworks.[21]

A big part is the community of practitioners and researchers who work in the forensics, Incident Response (IR), and malware analysis fields behind this project. Like mentioned before, because this tool is backed by big companies including Google, USA Department of Defense (DoD) Laboratories, Department of Defense Cyber Crime Center (DC3), and many Antivirus and security shops, this make sure that this tool doesn't stop receiving support and becomes outdated as it happens to many others. This is important to keep up with the ever changing technology used in malware or attacks. If the tool doesn't keep up with the technology, a memory analysis may be unable to retrieve the necessary information for the investigation. [21]

Another important point is the focus of this tool on forensics, incident response, and malware. It was developed by experts in this area, so the focus is kept on the types of tasks that these analysts typically do.[21]

And as the last advantage is that Volatility is completely free, and there is nothing another memory analysis framework can do that Volatility can't or that it can't be quickly programmed to do.[21]

The framework is intended to investigate the system's state independently of the host machine and consists of over 35 plugins for analyzing the memory. Plugins, also known as commands, are inputs in the command line to run code that gives the results we want. These plugins can be either developed by individual users or by the Volatility Foundation. These pieces of code are written in Python, like Volatility itself, and come in a large variety of functions. It's hard to explain what a plugin is exactly because each plugin is a block of code that runs in the memory dump and gives the expected result. The volatility foundation also offers a comprehensive guide for users to learn how to build their plugins in this way, enforcing the idea of an open-source platform. [21] [6]

The major problem most users have with Volatility is the lack of an interface or a GUI. This not only makes it less attractive, but it also makes that some actions are harder to perform. With the lack of a GUI, it's harder to see results, to scroll to previous results, and it's harder to copy and save these same results. Because of all the previous reasons, it's easy to see why the volatility framework is an amazing tool. Its major problem currently exists is the lack of a GUI. This is why Volatility, among other tools, was chosen as the tool to develop a GUI for this project.[6]

2.4 COMPARISON OF SOFTWARE

In Table 1, a comparison between all of the previously presented software can be seen. Each column contains the presented software, and each row represents basic functionalities found in most of the software as it's possible to know each software has its advantages over the other with Volatility being the one lacking more futures. However, the Volatility Framework is the base for the functionality of the RAM analysis for most of the other software. While some of the questions that each software is compared can be subjective is possible to get the overall idea for each software. By analyzing the table, we can see that software like Autopsy, despite being almost all positive in the questions asked, the lack of obtaining advanced results can be important for an investigation. Other software like Memgator checks all the boxes, but despite this, the software may still lack important functionalities for an investigation due to a generalization of the asked questions on the table. Overall by

analyzing the table, it's possible to see that each software brings something unique to this area and that each one of them serves a purpose.

Table 1: Table of comparison of memory analysis software

	Autopsy	Memoryze	wxHexEditor	Rekall	Memgator	Volatility
Easy to use	Yes	No	No	No	No	No
Easy to install	Yes	No	No	No	Yes	Yes*
Able to get advance results	No	Yes	Yes	Yes	Yes	Yes
Dependent on other tools	Yes, Volatility	No	No	Yes, Volatility	Yes, Volatility	No
Unique functionabilities	Yes	No	Yes	Yes	Yes	No
GUI	Yes	No	Yes	Yes, Web	Yes	No

*It can be easy to install but it depends on the version.

2.5 RELATED WORK OF GUI DEVELOPMENTS

Despite the developments of new software being low, there still exist some projects similar to this one that I will present and talk about the differences and the improvements that have been made. Starting with the oldest one, made by Steffen Logen, Hans Höfken, and Marko Schuba in 2012. This GUI presents the ability to run Volatility Framework plugins and to extract a file from the RAM Image. Despite being a very good project, I found some points lacking. These lacking points include the presentation of the results, which was still done by the Volatility Framework and presented on the command line that was not very user-friendly. Also, the ability to save results in this project was done by the result of the commands being extrapolated to a text file. This project also comes with a very important function that is the ability to extract selected files from the RAM Image. This is very important since the analysis of files is one of the key points in an investigation. Despite not being perfect, this project was very ambitious and well made for its time, with great ideas to inspire the development of the new project. In Figure 2, it's possible to see an overview of the described software. [19]

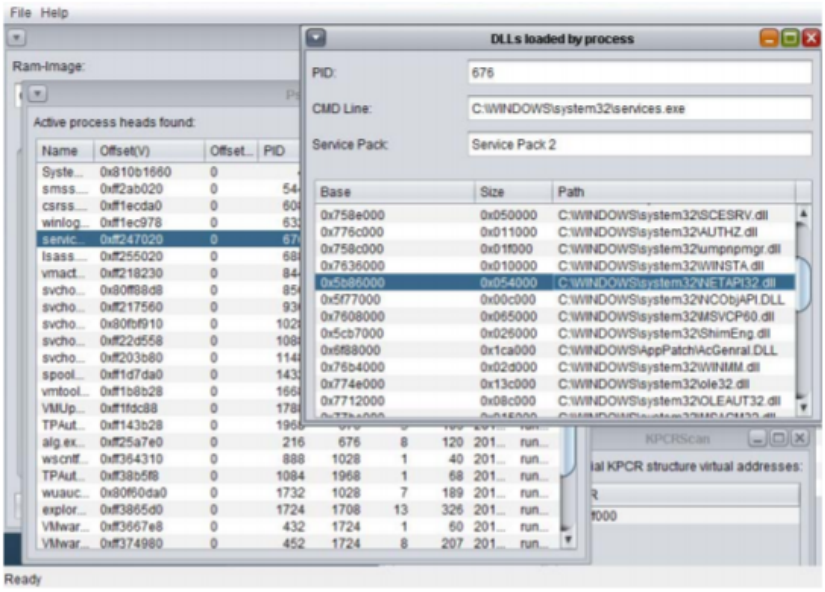


Figure 2: Volatility GUI 2012

Another project in this area was done in 2018 by Mr. Vivek Ravindra Sali and Mrs. H.K.Khanuja. This project is less of a "one does all" GUI and more of a dump of the RAM Image processes and search of strings. This project was based on the YARA plugin available in the Volatility Framework, with the ability to search for malware in the resulting dump of processes through a search in the resulting files with certain rules. The GUI on this project is not one of the most intuitive, but it gets the job done for its purpose. The biggest takeaway from this project was the necessity of the user to use the software to search for a keyword on a dump or a result. This is relevant since it gave ideas of implementation to the newly developed software. In Figure 3, it's possible to see the GUI of the described software.[14]

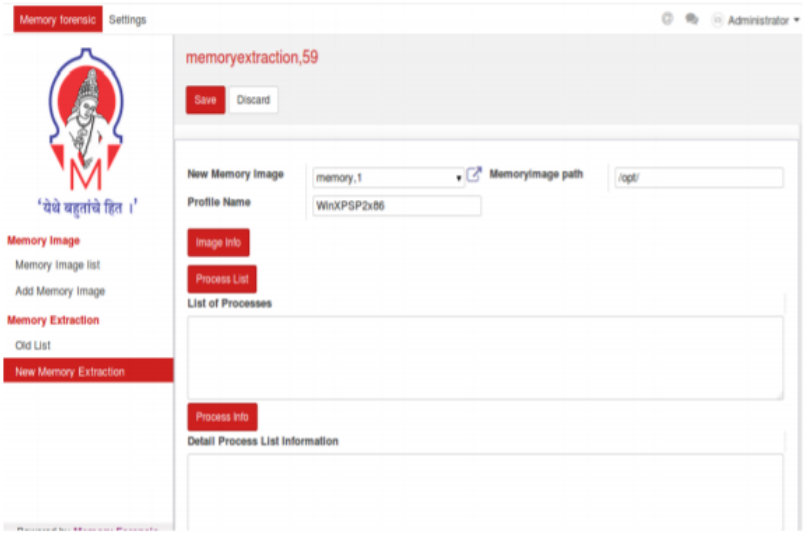


Figure 3: Volatility GUI 2018

The last project we will talk about was done by kevthehermit, and it was released in 2016. The installation of the project made by kevthehermit, while not being extremely complicated, can be more challenging to perform for a less experienced user. A more complex installation can divert many users from ever trying the software due to the problems with the installation. This project comes with some very impressive features, including the ability to run every Volatility plugin, a web interface meaning that it's possible to run in any system that can run a browser, and an immaculate presentation of the results inside the tool. In my view, some lack of features that are common in other areas of study like the creation of automatic reports, and others, is something that still lacked in this tool, but despite all, this tool is probably the best one that has been discussed in this chapter.

This project is the most similar to the one developed, but in my study on a similar project, I considered that this software was not very user-friendly. Starting with the installation, the focus of my project was that any user could pick up the installer associated with this project and be able to run the GUI without very much trouble.

In Figure 4, it's possible to see the GUI of VolUtility working on a webpage.

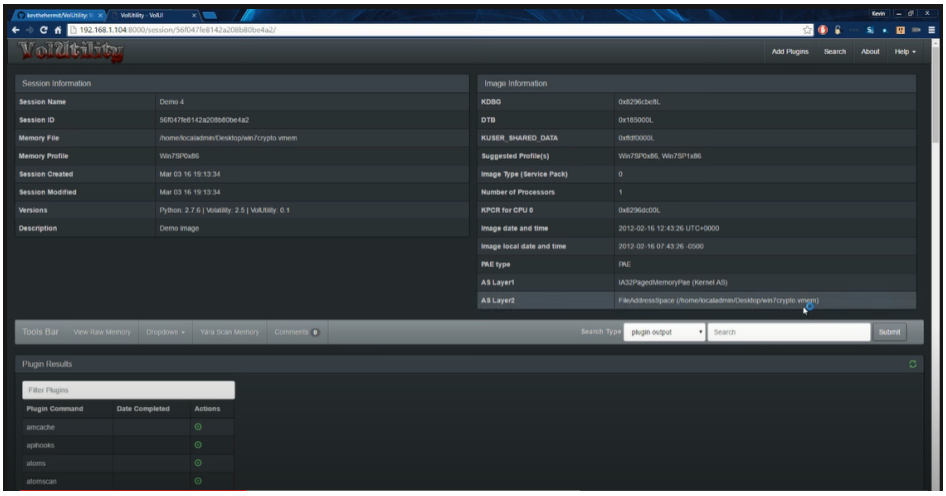


Figure 4: VolUtility

In table 2 it can be seen a brief comparison of the present related works and the project developed for this thesis, where we can see each software's strengths. All of these software were based or made to work on top of the Volatility Framework.

Several questions were asked to be able to compare all of the previously presented software and the developed software. Each of the questions tries to represent functionality that is present in any of the other software. As it's possible to see by analyzing table 2, each software brings some unique functionality where ultimately the developed software, VolatilityGUI, comes out on top being the only one that contains all of the functionalities that the other software have, including some uniques ones that can't be found on any other. Also, by analyzing the table, a conclusion can be reached that with each development and analyzed software, each iteration of these GUIs comes with more and more functionalities to help the user in an investigation. Meaning that the developed VolatilityGUI comes as a natural evolution of these iterations, and maybe in the future, there will be more software that will come with even more functionalities.

Table 2: Table of comparison of memory analysis GUIs

	VolatilityGUI 12	VolatilityGUI 18	VolUtility	VolatilityGUI (this work)
Easy to install	Unknown	Unknown	No	Yes
Easy to use GUI	No	No	Yes	Yes
Run Volatility Plugins	Yes	Yes	Yes	Yes
See results inside GUI	No	No	Yes	Yes
Multiple results at the same time	No	No	Yes	Yes
Save work	Yes	No	No	Yes
Search in dumps for keywords	No	No	No	Yes
Extraction of files	Yes	Yes	Yes	Yes
Creation of reports	No	No	No	Yes
Assoiate users with processes	No	No	No	Yes
Antivirus Integration	No	No	Yes	Yes
Save file encryption	No	No	No	Yes
History of perfomed actions	No	No	No	Yes
Easy to acess tutorial	No	No	No	Yes

2.6 SYNTHESIS

This chapter explained what the area of Forensic Analysis is and what it does with a focus on it impacts the investigation of RAM images. It also explained the differences between volatile and non-volatile data and what information can be retrieved from the non-volatile devices in detail.

After this, a revision of the existing tools for the had the capability of analyzing RAM was done. This was done by analyzing each tool and detailing what each tool was capable of doing.

Next, a deep analysis of the Volatility Framework was presented. The analysis contained everything from how it worked, how it was developed, where it was used, what its capabilities were, and its main problems.

After all of the software was presented, a comparison was made between all of them. The questions made were not only opinions but also technical questions like the presence of a GUI. From the comparison, it was possible to see that many of the existing tools were already depending on Volatility, each one with advantages over the others.

Finally, this chapter ends an analysis of the GUIs that have already been developed. This analysis saw what each GUI bought in terms of developments to the area and the improvements that each had to Volatility. Ending with a comparison between all of these software, including VolatilityGUI. In this comparison, it was possible to see that not only VolatilityGUI, the developed software, included not only all of the other software's unique functionalities but also some new ones that further improve the usage of the tool Volatility.

In the following chapter, we will talk about the development of this project.

3

VOLATILITYGUI

In this chapter, a discussion will take place about how the software VolatilityGUI was developed and all its requisites. Including the decision behind things like the choice of programming language, the methodology used, the user stories that have been made, and also the mockups of the design of the GUI and the data models of how the software works.

3.1 ARCHITECTURE OF THE SOLUTION

While choosing the tool that would be the base for this project, it was important that the implementation of the GUI and the process of the installation could be simple. Volatility was chosen because of all the previous stated reasons but also because of this.

There are various ways to implement a GUI, but a simple way is to send commands through the command line and receive the results. Because Volatility works through the command line, this was an easy choice for the first problem of getting the results from Volatility into a new GUI. In terms of installation, Volatility was also a good choice because of licensing. Volatility also comes in a standalone file that can be attached to the GUI for a simple installation process.

In Figure 5, it's possible to see a design of the communication architecture of how the GUI would work along with Volatility.

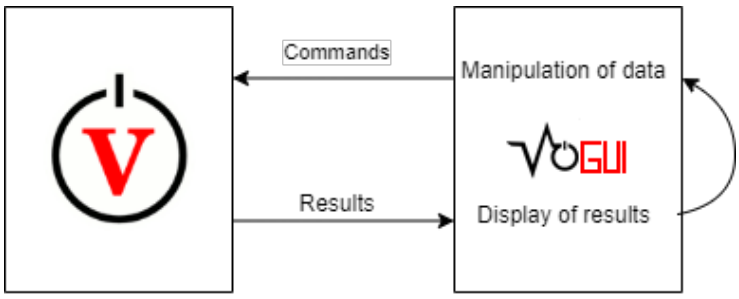


Figure 5: Diagram VolatilityGUI

3.2 PLANNING OF THE DEVELOPMENT

The first step in the development of this project was the creation of the project proposal. This document, which can be seen in Appendix A written in Portuguese, describes similar work previously done of both projects identical to this GUI and project done in the same programming language that this project and the main ideas behind this project were identified. The document also describes the purpose behind the GUI and the main functionalities that it should have present in the final delivery of this project. In the initial stages of the development, this project was thought of as a project developed by two persons, and this proposal was written with this idea. Despite this, the document gives a general idea of the project's methodology, like the programming languages that could be used for the development of this tool. The chronological plan of the project is also presented, in Figure 6.

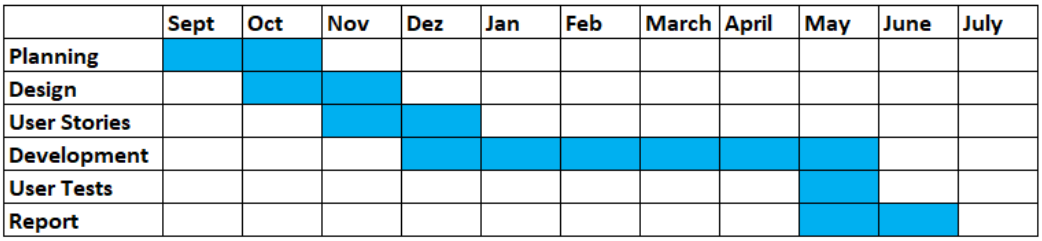


Figure 6: Chronological Plan

After this, the object to deliver is presented, where it is told it should be delivered in the final delivery of the project. The final part of this proposal describes how the evaluation of the project will be made and what kind of tests the tool is going to be subjected to.

Of course, the ideas and functionalities of the project changed during the development of the tool, and this tool wasn't any different. Despite that, this was the initial planning of the tool. In one of the following sub-chapters, the initial mockups of the tool have been drawn.

3.3 METHODOLOGY OF THE DEVELOPMENT

The methodology behind the project followed an agile-inspired methodology with the user stories to be developed. Meetings were also scheduled every two weeks to discuss the progress made on these user stories, with monthly main functional objectives that should be met. Because the development of this project was inspired

by an Agile Methodology development, it's necessary to discuss the definition of this concept. An Agile Methodology is a group of software development methods that are based on iterative and incremental development. It's composed of four major characteristics that are fundamental to all agile methodologies that are: adaptive planning, iterative and evolutionary development, rapid and flexible response to change, and promoting communication. During the development of this project, many of these characteristics were applied, like adaptive and iterative, and promotion in communication planning. After each weekly meeting, there was a discussion on the next steps to take. Not only this but there was also a demonstration of the development made and a discussion on what could be improved, meaning a flexible response to change. This project, in particular, was not taken into any specific because of the size of the team and because of the not defined deadlines for delivery of the project, meaning that it would not be worth applying any specific methodology. [9]

The user stories were done with the functionalities described and the acceptance criteria, and a rating of priority that considered the more used functionalities inside of the Volatility Framework and the amount of work that it would take for the functionality being developed. These user stories were thought of as the main functionalities that the GUI should have before being delivered and some other ideas that would be "nice to have". The initial developed user's stories are present in the following subsection.

3.3.1 *User Stories*

In the following Tables 3, 4, and 5, it's possible to see the initial user stories that have been developed for this project.

These user stories contain the following columns, Theme, User Story, priority, Acceptance Criteria. The Theme separates each user story by the category it belongs to, this can be Interface, Backend, or a combination of both. The new column is the User Story that contains the description of the User Story, where it determines what needs to be developed on the software. The priority can have options between 1-5, this determines the priority that the User Story should be implemented on the software. The bigger the priority, the more critical it is to implement. The last column is the Acceptance Criteria that gives the activity that determines if the User Story has been completed, meaning that for the User Story to be closed, the user must be able to complete the activity described in the Acceptance Criteria.

These User Stories were developed before the beginning of the development of VolatilityGUI, some functionalities were created with a User Story assigned to them. Most of the User Stories present in Tables 3, 4, and 5 were successfully completed.

Table 3: Table of User Stories - Part 1

Theme	**User Story**	**Priority**	**Acceptance Criteria**	
Interface	As a user I want to create a new project	5	The user should be able to choose to create a new project	
Interface	As a user I want to open the Easy Report window	5	The user should be able to open the Easy Report window	
Backend	As a user I want to be able to load the RAM image into both Easy Report and the general GUI	5	The user should be able to load the desired RAM image	
Backend	As a user I want to be able to choose between several options in Easy Report	5	The user should be able to choose several options in Easy Report	
Interface	As a user I want to be able to see information about the options chosen in Easy Report	4	The user should be able to see information about the chosen option in Easy Report	
Backend	As a user I want to be able to create a report in PDF format through the chosen option	5	The user should be able to create a report with the specified options	
Interface	As a user I want to be able to instantly see crucial information when loading the image in the GUI before loading the information in the GUI	4	The user should be able to see information quickly when loading the image.	
Interface	As a user I want to choose between the option of commands available in Volatility	5	The user should see all the commands available in the GUI	
Backend	As the user, when choosing a command I want Volatility to preform that action on the selected image	5	The user should be able to see the effect of his commands in Volatility	

Table 4: Table of User Stories - Part 2

Theme	**User Story**	**Priority**	**Acceptance Criteria**	
Interface	As a user I want to see the results of the commands in the "Output" section	5	The user should be able to see the results of the commands in the GUI	
Interface	As a user I want the possibility to write my own command and see the result in the Output	4	The user should be able to do his own command in the GUI and see the result in the GUI	
Interface	As a user I want to see the Easy Report interface by selecting the button with the same name	4	The user should be able to reach the Easy Report menu through the button in the main GUI interface	
Interface	As a user I want to see custom information about the command I will run	3	The user should be able to see information about the command he will run depending on his image	
BackEnd	As a user I want to use the Find section to find the keyword chosen in the Output section	3	The user should be able to search for the keyword chosen in the output section	
Backend	As a user I want to be able to change the image without having to close the GUI	4	The user should be able to change the RAM image without having to open and close the GUI again	
Backend	As a user I want to be able to save the current state of the GUI like Output, chosen image, history, among others	3	The user should be able to save the state of his "project"	
Interface	As a user I want to be able to choose to load a previously worked project	3	The user should be able to load the previously saved project	

Table 5: Table of User Stories - Part 3

Theme	**User Story**	**priority**	**Acceptance Criteria**	
Interface/Backend	As a user I want to be able to see a graphical interface of the process tree of the chosen RAM image where it is possible to see all the information about the processes without the need to make commands	2	The user should have a new interface of the process tree and see all the necessary information without any commands	
Backend	As user I want to do an antivirus analysis of the RAM image using both plugins and external websites for both output and report	3	The user should be able to choose the "Antivirus" option in the File Menu and run an antivirus analysis of the RAM image	
Backend	As a user I want to see the history of the commands done and the loaded image through the "File" menu	2	The user should be able to see the history of the commands done in the GUI	
Backend	As a user I want to be able to extract all the DLLs present in the RAM image	3	The user should be able to extract all the DLLs from the RAM image	
Backend	As a user I want to be able to extract all the EXEs present in the RAM image	3	The user should be able to extract all the EXEs from the RAM image	
Backend	As user I want to be able to extract only the DLLs chosen from a list	2	The user should be able to extract the DLLs chosen from the list from the RAM image	
Backend	As a user I want to be able to extract only the EXEs chosen from a list	2	The user should be able to extract the chosen EXEs from the list in the RAM image	
Backend	As user I want to see options for Easy Report with various levels of complexity for more or less complex reports	4	The user should be able to choose options for various types of reports	

3.3.2 *Functional Requirements*

In the initial stages of development, the GUI was supposed to be cross-platform, meaning it could work in any of the main operating systems, including Windows OS, Linux, or macOS. But due to some constraints with the technologies used and the technology of the Volatility Framework, this was not possible. The GUI could be cross-platform but be different from other developed GUIs and develop some more advanced functionalities. This would not be possible, so it was decided that the GUI should mainly work in the most used operating system, Windows OS.

In terms of system requirements, the GUI should be able to run in any system that is able to run the Volatility Framework. Because some functionalities, like calculations and comparisons in the code, of the GUI take full advantage of the system's computational power, this means that the GUI can "run" faster in a more powerful system.

Because the GUI runs only in the Windows operating system, this means that the Volatility Framework works through the standalone version indicating that the GUI can run without any extra component. It's automatically installed through the setup, like the installation of python that it's needed for some versions of the Volatility Framework.

In the following sub-chapter, we will see the mockups that have been designed in the initial stages of the development of this project.

3.4 MOCKUPS

The following Figures 7, 8, 9, 10, 11, 12, 13, 14, and 15 are a compilation of the mockups that were designed at the beginning of the development of this tool to get an idea of how the GUI should work and look.

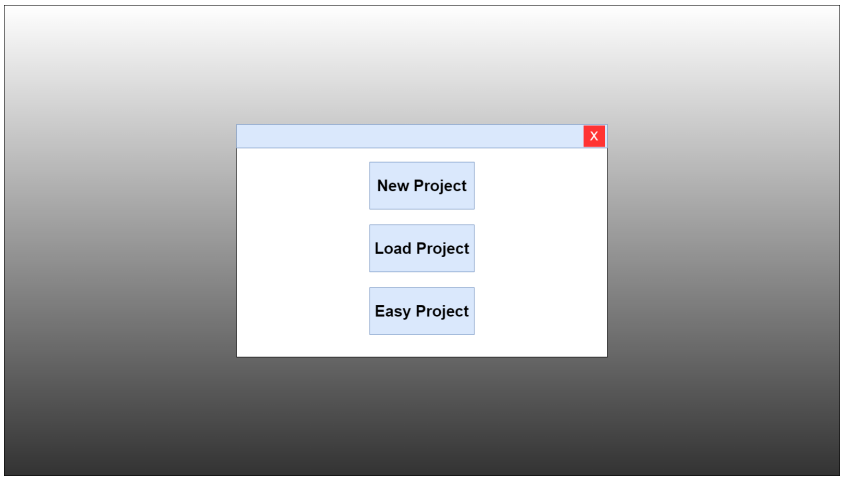


Figure 7: Mockup Entry Screen

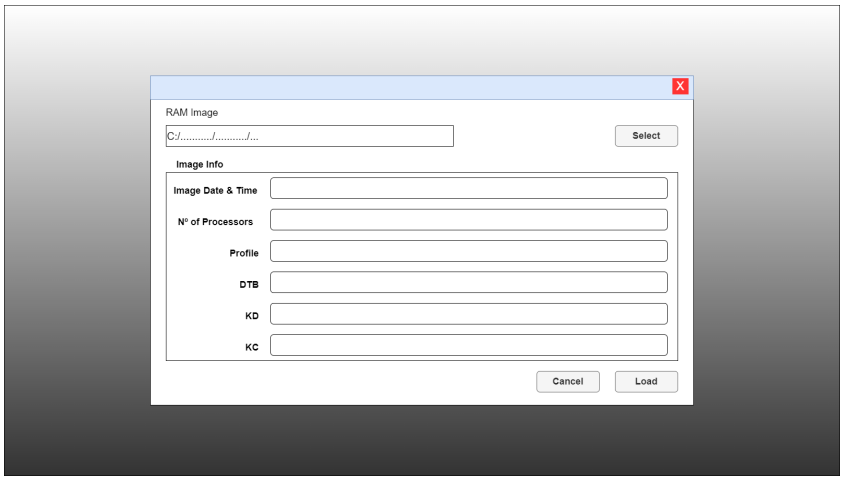


Figure 8: Mockup Load RAM

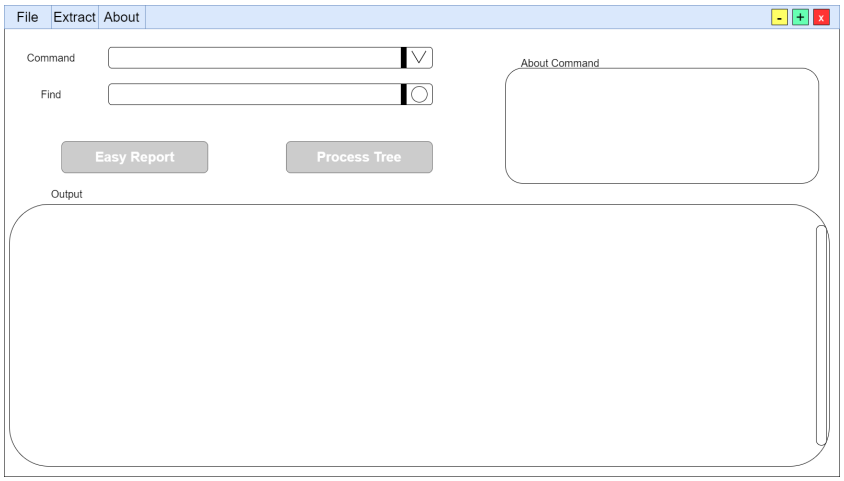


Figure 9: Mockup Main Screen

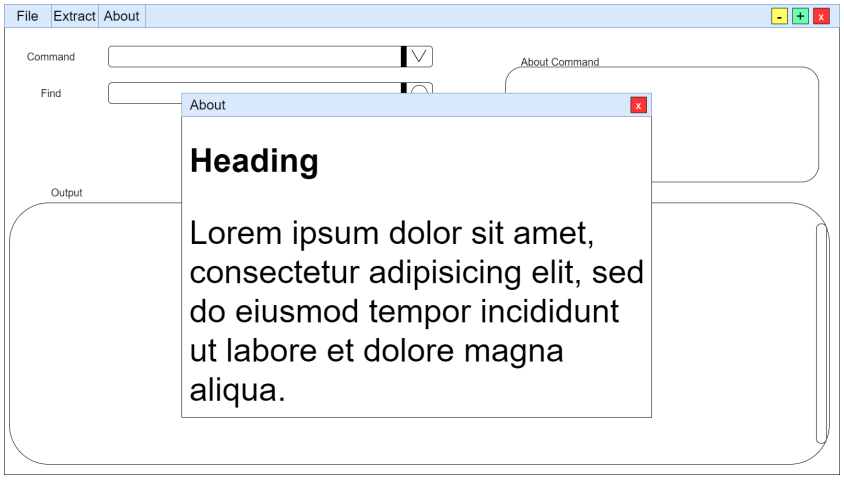


Figure 10: Mockup About Screen

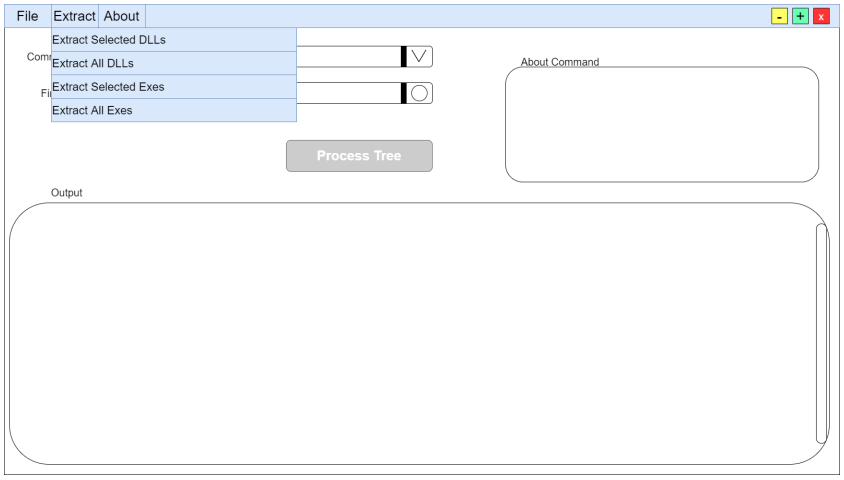


Figure 11: Mockup Toolbar

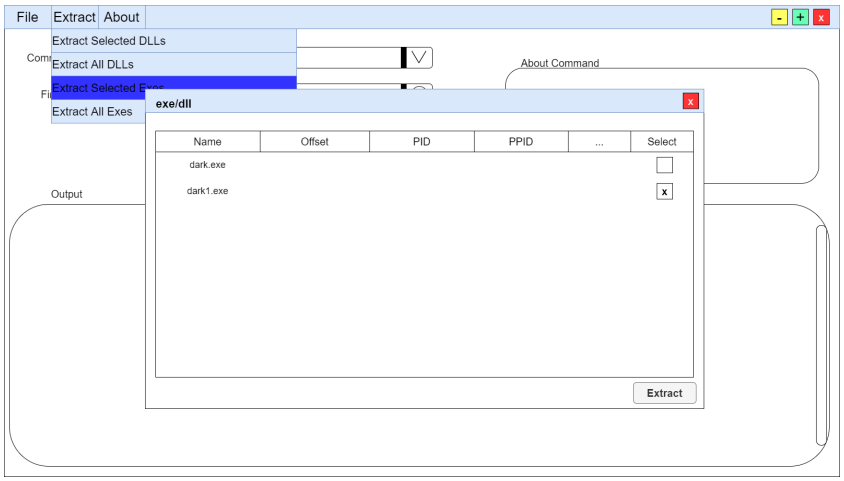


Figure 12: Mockup Extraction

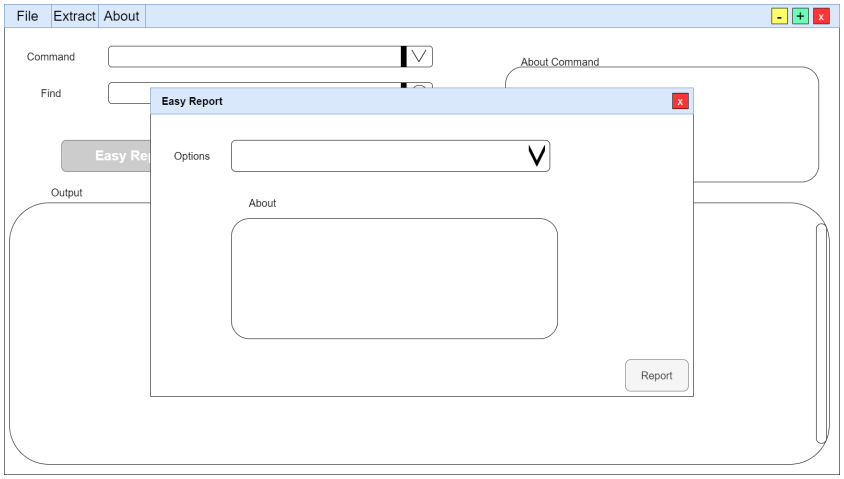


Figure 13: Mockup Easy Report

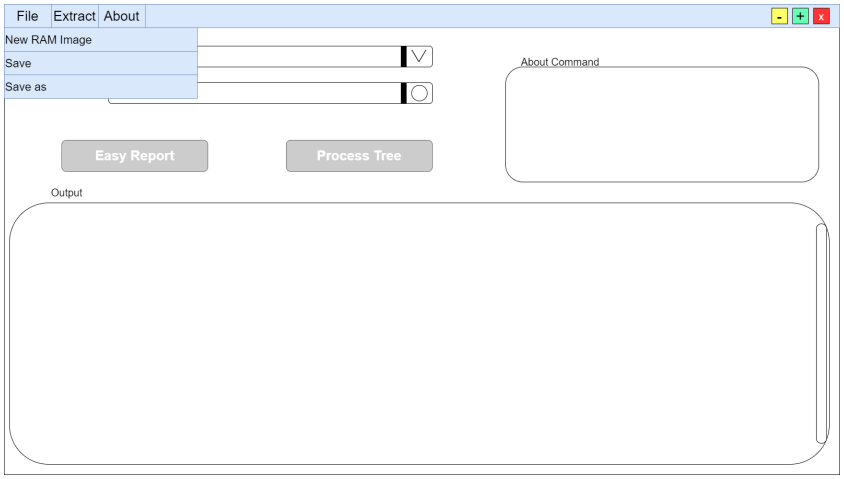


Figure 14: Mockup Toolbar Options

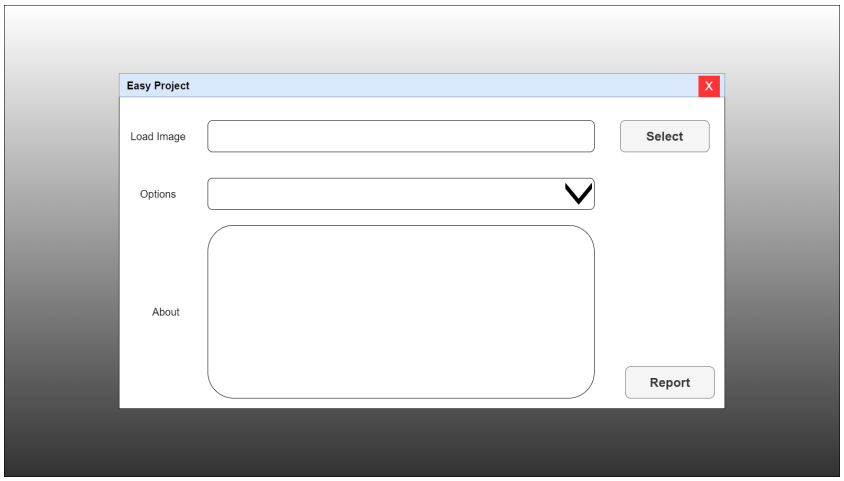


Figure 15: Mockup Easy Project

Some functionalities were only developed after the mockups had been designed, meaning that mockups were not developed for every functionality that had been implemented.

The next chapter will describe how the GUI and the Volatility Framework data model works and how both this software can communicate with each other and share data.

3.5 DATA MODELS

Despite Volatility Framework being a great software, it has not been made to be integrated with other tools meaning that for any GUI, some techniques had to be applied to make this work. This will be specified in this section because of the complexity of how the GUI handles data and gets data.

While the GUI can transform data and show it to the user, to get any kind of data from a RAM image, the GUI sends the input of various commands to the Volatility Framework, so this one can run in it and get the expected results. After any commands that have been sent by GUI have finished running, the data is automatically sent to the GUI, where it will be transformed and displayed to the user.

The following Figure 16 shows a diagram of how this process works.

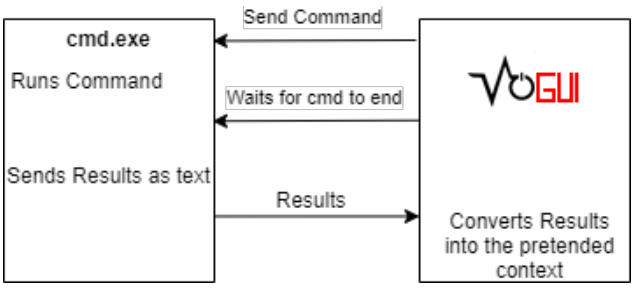


Figure 16: Data Model Diagram

As shown in the diagram in Figure 16, it's possible to see that the data that the developed software requested, it's only able to display data and show it to the user because of the original Volatility Framework software. VolatilityGUI is only responsible for managing and handling the data while the back work it's done in the Windows OS Command line. Following the diagram, we can see that after the command has been input by the user and it's been run by the command line, the result it's returned as text to VolatilityGUI. The data is always handled as

text and manipulated to display the pretended results to the user. Following some security methods, this data is only kept as long as the user desires, and saving this data creates encryption of these results, but this is possible to be seen in the next chapter.

3.6 SYNTHESIS

In this chapter, a study was done on how the VolatilityGUI would be developed and several ideas on functionalities to implement. This study was intended to know how VolatilityGUI would be developed.

The first part explains how VolatilityGUI communicates with the system where Volatility is running. Followed by a detailed explanation of the planification of development, where each development step is explained with timelines. In this part, the project proposal is also presented, where the programming language to be used is detailed and several more small points of the development like the object to be delivered.

Finally, the methodology of development is presented and explained. Also, in this part, all of the mockups of VolatilityGUI are shown. And lastly, the diagram of the VolatilityGUI is presented and explained. From all of this study came the technical developments of VolatilityGUI that will be described in the next chapter.

4

TECHNICAL DEVELOPMENT

This section is going to discuss the development of the software. The discussion is going to be more centered on the decisions taken in more technical terms, like the code that has been made and the ideas behind each implementation, like the first implementation and the final implementation, after some discussion in the meetings. It will also be possible to see the problems encounter and the workarounds behind them. This section is divided into each area of implementation, like the choice of the working environment and the load of the RAM Image. This section also follows an order of subsections in the order in which each part was implemented into the software.

4.1 CHOICE OF ENVIRONMENT

Because of the previously mentioned solution with how the VolatilityGUI and the Volatility Framework communicate and trade information was necessary to facilitate the installation. This was done was by limiting VolatilityGUI to a single environment/operating system in which the GUI should run. This was done to facilitate is the introduction of users to this area of RAM analysis. With a single environment to work with is was much simpler to tackle a problem like a simplified VolatilityGUI installation. For a simple installation, it was necessary to restrict this installation to one operating system, because according to many websites and reports Windows OS it's still the operating system mostly used. It was chosen as the environment to develop the software in. [7]

The Windows OS was also chosen as the environment to develop this software because of the version of the Volatility Framework that the GUI installs and runs. The GUI used the Volatility Framework Standalone version to do the back-end processes, and this is automatically installed by the VolatilityGUI setup. Because the Windows OS environment was chosen, this influenced the language of programming used for the development. During previous research for this project, there were some indications that the programming language C# would be the best one to tackle all of the problems using the editor Visual Studio. C# felt like the obvious

choice for this development because it's the recommended language for developing applications in Windows OS and because of the many libraries and integrations with software that work like Volatility.

The setup is as easy and simple as many of the other software of this kind, with the possibility to choose the installation location. Like it was said before, the setup installs everything necessary for the VolatilityGUI to run with the purpose of facilitating the process of installing the software to the user. The setup also creates a desktop shortcut for easy access. Opening the software requires administrative rights because Windows OS requires some permissions for the software to run correctly. The setup was created using the 'Microsoft Visual Studio Installer Projects' and following the official documentation provided by Microsoft. The only customizations were the icon that was created and the inclusion of the Volatility Standalone program in the installation. Everything else was kept fairly with the default setting, only changing some settings like the description and the product name.

In conclusion, the chosen environment was Windows OS, and the software was developed in C# using Visual Studio. Running the VolatilityGUI setup, which was kept very simple in terms of configuration, is the only thing necessary to start the VolatilityGUI and start working. In Figure 17, it's possible to see an image of the setup of VolatilityGUI.

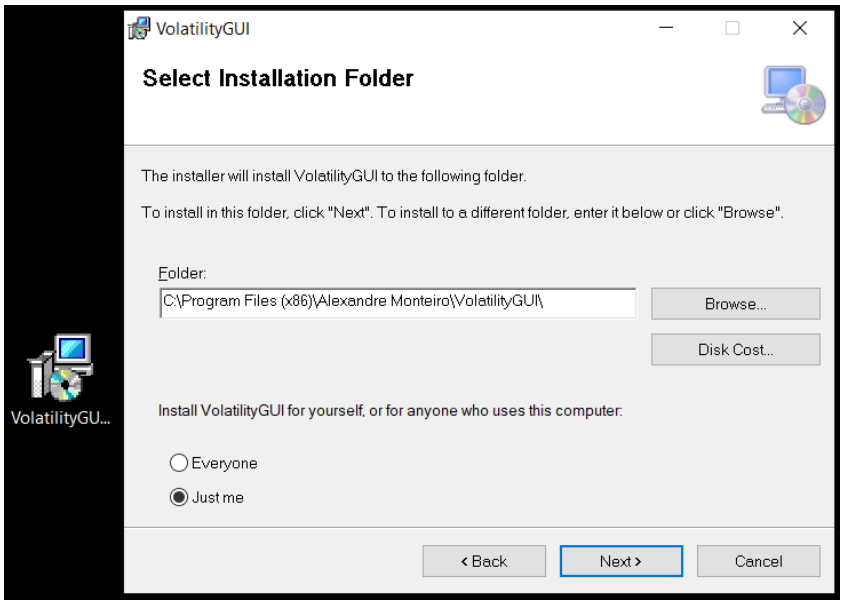


Figure 17: VolatilityGUI Setup

4.2 CONNECTION TO VOLATILITY FRAMEWORK

This subsection will explain how the software communicates with the Volatility Framework and vice versa and how this integration was developed. Due to previous experience, it was possible to know how the Volatility Frameworks works, and like it's been explained before, this software run in the command line. Because of this knowledge, it was necessary to find a way to run the commands of Volatility directly in the command line and then retrieve the results after this has finished.

On a first approach, the solution went through creating a new process for each command that the VolatilityGUI needed to send. Since Volatility runs in the command line, it was necessary to define the running process as a "CMD.exe". The next step was to run the Volatility command. A standard command line for Volatility is, for example, "volatility.exe -f ram.raw imageinfo". As it's possible to see, the command doesn't exist in the Windows system while using Volatility Standalone to run. It was necessary to include the Volatility Standalone in the installation of the software and make use of some specific commands to have a result. After the command is sent to the command line, the created process starts and waits until it's finished to read the output of the command line into a variable that will receive this output. The software handles the manipulation of the data as it's possible to see by the previously described behavior. Other configurations were done in this process to give a more pleasant experience to the user, such as the creation of the command line window.

On a second approach, it was necessary to include some more functionalities, such as running this process in the background. With the first implementation, the behavior of the VolatilityGUI was to "freeze" the software screen until the process was over. The solution is to run this process as an asynchronous process. Asynchronous programming is the ability to deal with multiple requests simultaneously, and the application in this type of software is so that the software won't "freeze". For this, it was created a BackgroundWorker that would take the various steps of the previously created process and run it on the background. This is also very important to a much better experience for the user since it was possible to show a progress bar that indicated that the process was running, as it's possible to see in Figure 18.

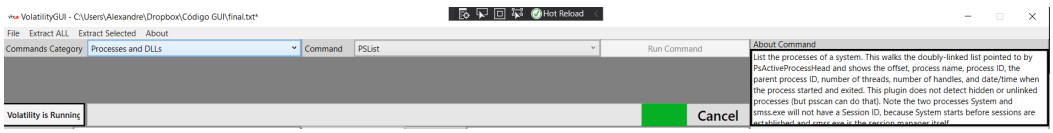


Figure 18: Progress Bar

The connection between Volatility and the developed GUI is one of the most important functionality in the entire development of the software because most of the following steps make use of this process. The development of this process had to be as good as possible so that it wouldn't affect the other actions. Depending on the functionalities, the described process of how Volatility and VolatilityGUI can suffer alterations, and in those cases, the alterations made to this process will be explained in detail. Inside of the process, there is a configuration with the name arguments that can be changed to suit the command that has to be sent to fit the functionality in question.

In the following sub-chapter, we will see how the VolatilityGUI loads RAM images and how it uses this defined process to get some basic info about the loaded image.

4.3 LOADING OF RAM IMAGES

In this sub-chapter, we will see how the software loads RAM images. Unlike the previous sub-chapter, this implementation has stayed pretty much the same since the beginning. For the user to be able to load a RAM Image, it is necessary to access the specific window for this functionality, which is accessed through the shortcut located in the top navbar of the main windows of the software.

Opening the "Load RAM" Window, the only available option is to click the button "Select". Clicking this button will open "Windows OS Explorer" where it is possible to choose a file of the RAM image. After clicking the "Explorer" button to select the pretended file, the VolatilityGUI will verify if the extension of the selected file is any of the many types of files supported by the Volatility Framework. If the software's response is positive, the VolatilityGUI will continue the process. Otherwise, the software will give a warning message, and it will not continue the process. Continuing the process, the VolatilityGUI will send some arguments to the command line and get the basic information related to this RAM Image. The command used is "imageinfo". After the command has finished running, the more important information is taken from the temporary variable and displayed to the user.

This information that is shown to the user is selected based on importance, meaning that the more specific information that won't be relevant to most users will not be displayed. But this information can still be accessed by other methods. One very important field that is displayed to the user is shown as the profile field. This field is crucial for the entire rest of the usage of the software. Because all

other commands other than "imageinfo" depend on this command when inputting a required command in the command's arguments to send to Volatility. The field profile will be displayed as a list of choices because the output of "imageinfo" normally displays multiple possible values for this field. Because of this and to make an easier experience for a less experienced user, this profile is automatically chosen as the best one to use, but if any problems occur with a command, which is a possibility because of the profile, it's always possible to change this profile before and after loading the RAM Image into the software.

One of the last additions to this window suggested by one of the professors that assisted with this project was the inclusion of the calculation of several cryptographic algorithms of the loaded image. This was added due to the necessity to prove that the file being investigated is in the fact that one. For this, the software automatically displays the values of the file for Secure Hash Algorithm 256 (SHA256), Secure Hash Algorithm 1 (SHA1), and Message-Digest algorithm 5 (MD5).

This window can be used to load a new RAM Image, but it can also be used to load a new RAM Image for the same project. The way the software was designed revolves around the idea of the project and not the investigation of a single RAM Image, meaning that it is possible to start working on a new image without having to create a new project. Loading a new RAM Image will erase almost all of the work done in the previous Image. This information will also be displayed in the attempt of loading a new RAM Image.

In Figure 19, it's possible to see the Load New RAM Image window.

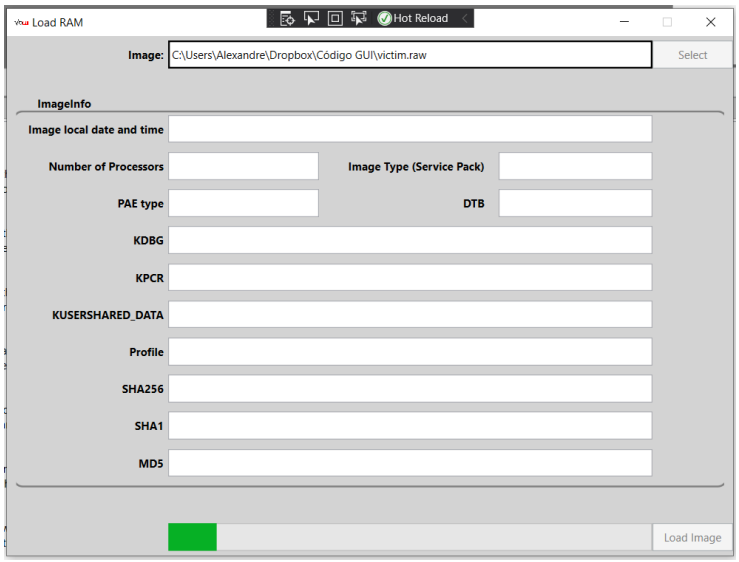


Figure 19: Load RAM Window

The following sub-chapter will show how the commands can be run and what commands were chosen to be included in the software.

4.4 CHOICE OF COMMANDS

This subchapter will show which commands were included in the software and the motive behind the ones that were excluded. It will also be possible to see how the final user will run commands. Figure 20 shows how the main window of the VolatilityGUI looks and where the user can run commands.

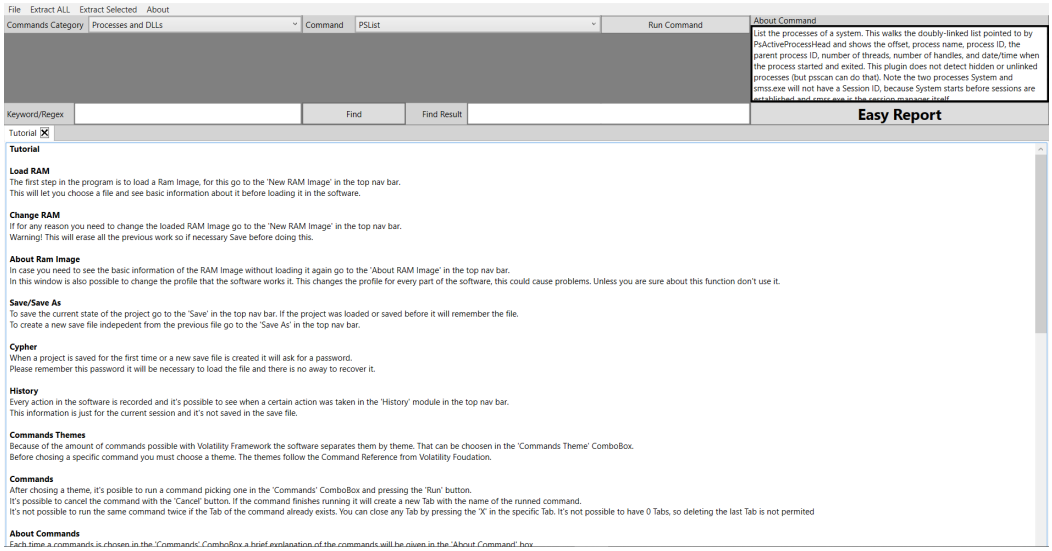


Figure 20: VolatilityGUI Main Window

Due to the sheer number of different commands that the Volatility Framework can run, it was necessary to find a way to display all of the commands other than with a single list because of the size of this list. For this, it was created the list "Commands Category". This list contains several values that influence the second list with the name "Commands". These lists were created following the official documentation of the Volatility Foundation. The list "Commands Category" box is the part of the software that each type of command affects. For example, the commands inside of the category Networking contain all of the commands that will get information from all of the networking points inside of the RAM image. The VolatilityGUI will not run any command without going through the process of choosing both the Command Category and the Command itself.

In the "Commands Category" list, there is an extra option that is not present on the Volatility Foundation website. This option is called "Advance Commands".

This option was created to give the user the ability to run commands that can use variables such as specific processes to run quicker and to provide a more specific output. For example, choosing the "Advance Commands" category will make the command "MemMap" appear on this list. While it's possible to run this command in its entirety in the Category "Process Memory", this command will take a significant amount of time to process. The amount of time to run can be reduced by choosing the optional parameter for other commands that are affected by this problem. This parameter can be an offset, a type, or any other. This can be done by using the options present in the "Advance Commands" category. The only exception in this list is the presence of the command "HiveDump" that requires parameters and for this is not present in any other category other than this one.

To facilitate the usage of the user when choosing a command from the "Advance Commands" category a third box will appear that will automatically be filled with the option of the parameters that a specific command can receive. This works by running other commands inside the VolatilityGUI that will fill this list with the required values. As it's possible to see in Figure 21, a third box is filled with the possible values for the user to run commands.

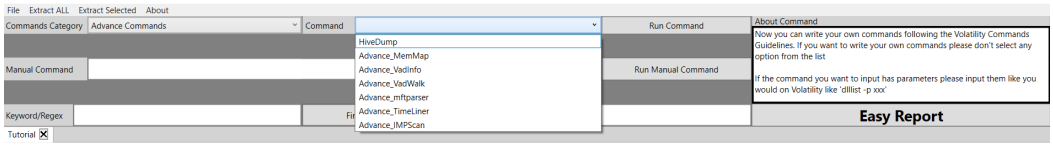


Figure 21: Advance Commands

Choosing the "Advance Commands" category will also open the ability to run manual commands. These manual commands can be anything from the available official commands from the Volatility Framework but also from any additional installed plugin. In addition, to give the ability to run even more specific commands or commands that exist outside of the Volatility sphere. In Figure 22, it's possible to see the box that can run the manual commands.

Figure 22: Manual Commands

An extensive study was done on every command that Volatility offers, despite that not every command will work for every image. This is because many commands were explicitly created for RAM images of a specific operating system, like VirtualBox, or even if the RAM images have any information related to crashes. Despite this,

the software is prepared for this kind of situation. As previously mentioned, the VolatilityGUI was created to work on the Windows OS platform and on Windows OS images, and due to a lack of time and resources, any command related to RAM images from the operating system Linux and macOS was not included. Another problem encountered was with the command "screenshot". This command was not included in the VolatilityGUI due to the fact that this command uses a specific library to run. This library appears to no longer exist/work, despite multiple attempts, and so, this command was not able to run, even outside of the created software, and for this reason, it was not included. [15]

The command "hashdump" it's used to extract and decrypt cached domain credentials stored in the registry. For this, the command requires several parameters from the hive, usually the user is required to get these values by hand, but VolatilityGUI automatically receives these values and runs the entire process of this command for the user. This is the only command that requires some more complexity, and because of this, it was decided that it needed a separated explanation.

Lastly, every command that is chosen from the "Command" box will be given a brief description on the right side of the software. This description provides information on what the command does and the output of the command. In the case of the advance commands, this description will also say the extra parameters that each command requires.

While the commands are running, it's possible to see a progress bar that will give the information to the user that the VolatilityGUI is running the process of the chosen command. After this process has finished, this bar will disappear, and an output will be given, which is the information of the following sub-chapter.

4.5 CHOICE OF OUTPUT

In this subchapter, it will be shown how most information is displayed to the final user. The output went through many iterations, but it will only be described the first and the final implementation of the output. In the first implementation, the output of any run command was inserted in a unique text box. This text would be cleaned every time a new command was selected and run. Because normally, an investigation required the analysis of multiples commands outputs at the same time, this mechanism didn't make sense for this kind of project. For this reason, some advancements were made in this implementation by adding the outputs of multiple commands to the same textbox. In this phase of this implementation, the output

textbox would be empty, and for every command that runs, it would automatically be added to this textbox, and the multiple commands would be separated by a black bar. This was not very user-friendly. For these reasons, a new implementation of the outputs mechanism was necessary.

The final implementation involved a system of tabs similar to a web browser. Each tab requires a title for identification, and the system automatically creates a new tab for each run command and gives a title according to the command given. In the case of normal commands, the tab's title will be the name of the command like it appears on the commands box. In the case of advance commands, the title of the tab will be the command and the parameters given, and in the case of manual commands, the title will be the word "advanced" plus the command given by the user. Because the result of a specific command will always be the same, the VolatilityGUI will detect if the command that the user is trying to run already exists in the form of a tab and will prevent the same command from being run again. This is to prevent repeated results making it more confusing for a less experienced user. The tab can be deleted, but the software is always required to have at least one tab existing. Any deleted tab enables the user to run the previously deleted tab command once again in VolatilityGUI. In Figure 23, it's possible to see the final implementation of the tab mechanic in the developed software.

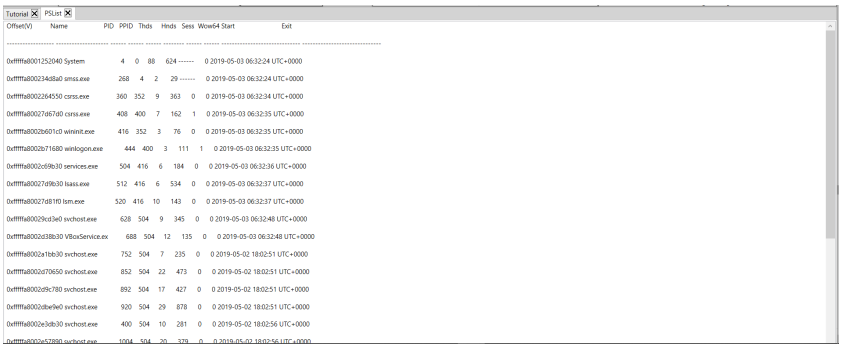


Figure 23: Tab Output

Because of this, a tab will always be required to exist, so in any new project the first tab will always be the tutorial.

The tutorial inside of the first tab exists with the purpose of giving the user the ability to learn everything necessary to use the VolatilityGUI without having to leave the software. This tutorial presents a short description of every functionality inside of the software, including how to use it, limitations, and general information. This tutorial despite being a good tool to accompany a new user into the software it only gives a brief explanation of the functionalities and only the usage and the

experience can give a full picture of how to use the VolatilityGUI. The tab of the tutorial can be closed like any other tab. In Figure 24, the Tutorial of VolatilityGUI can be seen.

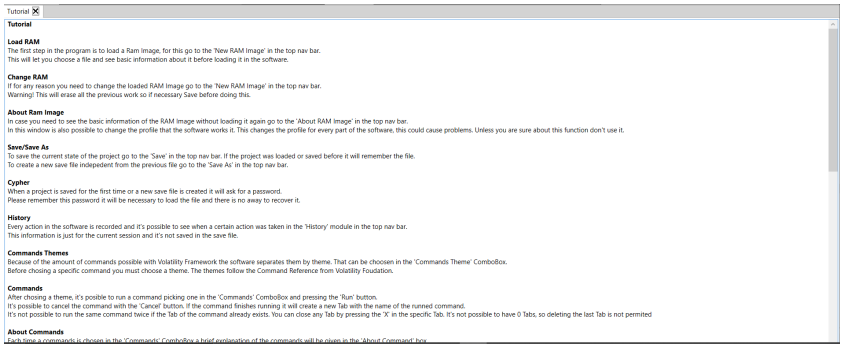


Figure 24: Tutorial Tab

In the following sub-chapter, it will be possible to see a description of the developed tool on the software for the purpose of easing the usage of the software, also known as "Quality of Life" functionalities.

4.6 QUALITY OF LIFE

It was previously mentioned that the focus would be on tools that have been developed to assist the user on the usage of the VolatilityGUI or even on the investigation. The first Quality of Life (QoL) mechanic that was already mentioned was the "About Command" box that gives information about the command that the user is running.

Another tool developed in this thematic is the "Find Function". With this tool, the objective was to eliminate the necessity to use external tools to find simple strings in specific files. With this, the user can write a word or a sentence, and pressing the "Find" button will initiate the process. The VolatilityGUI will try to find the input text in the tab that is currently selected. VolatilityGUI will automatically scroll to the first found match. If more than one match is found by the "Find" function, the button "Find" can be repeatedly clicked. This will cause the VolatilityGUI to scroll to the next match. Suppose the software is able to find any positive results it will automatically highlight in yellow the text that matches the input text. In that case, it will count the total number of matches, and it will display this information in the text box by the right of the "Find" button. It's possible to see this behavior in Figure 25.

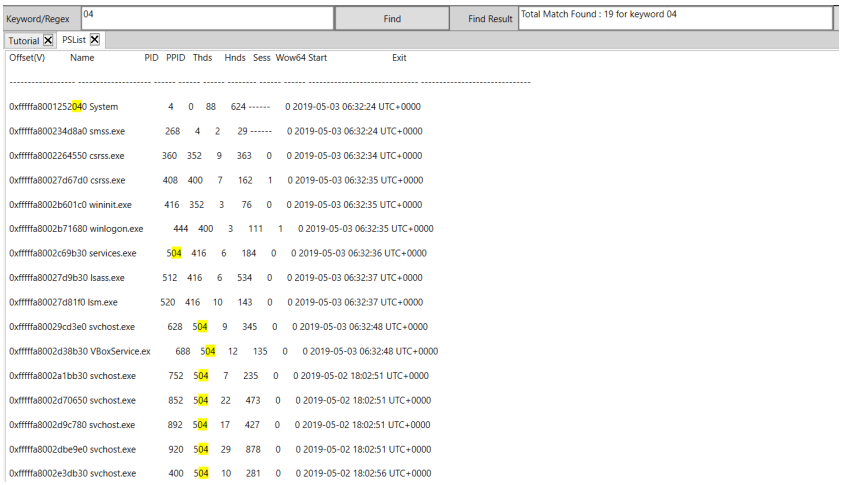


Figure 25: Find Function

If a new Find process is triggered, the previously highlighted text will be converted to normal, and the total number of matches will reset for the next input text.

Other than text, the "Find" tool was also developed to work with regex. Regex is the ability to design patterns to find specific information in the text. For example, the pattern '[0-9]' will try to find every number on the text of the selected tab. With this, the user can use any pattern to start the "Find" process that will behave exactly like the previous string search, highlighting the text and counting the total number of matches. To use the regex functionality, the user must start the text of the Find textbox with "$Regex:". This is described in the previously mentioned tutorial. This way of searching was implemented in a way to not restrict the user to find, for example, the word "regex" or to find text similar to any pattern like '[0-9]'. In Figure 26, it's possible to see the Find process using regex rules.

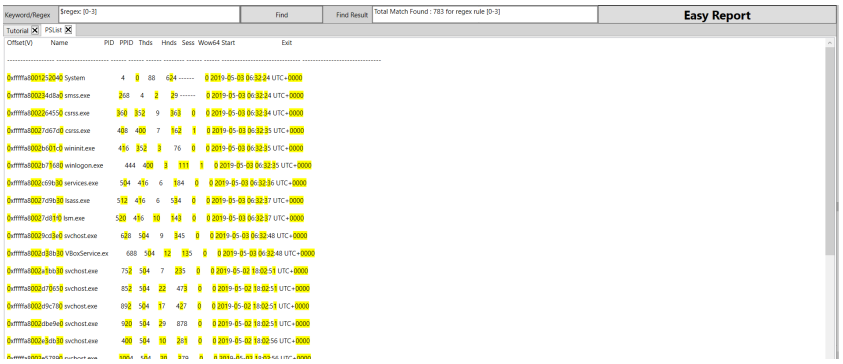


Figure 26: Find Regex Function

Another QoL tool developed was the Forensic Investigation. Its primary purpose was to automatically match processes and the respective users responsible for these

processes. This development came from the necessity of many investigators to join both of these data together. This specific window can be accessed by pressing the button "Forensic Window" in the tab. It's possible to see the behavior of this tool in an existing RAM image in Figure 27.

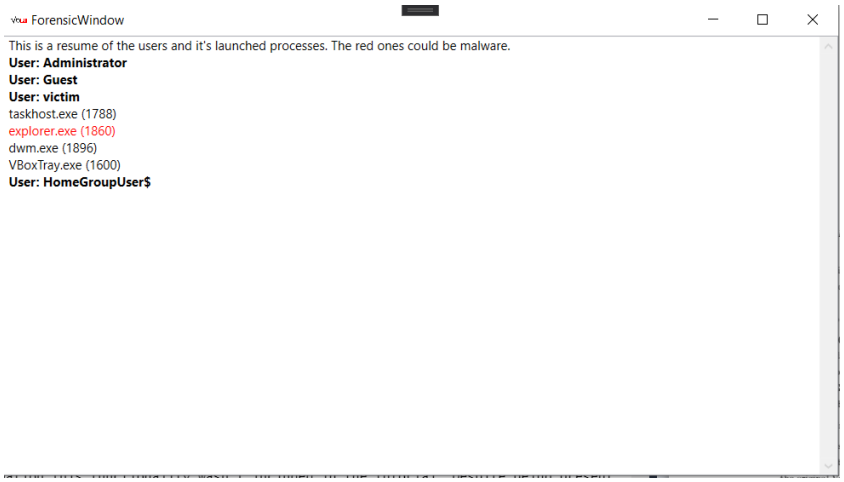


Figure 27: Forensic Window

This process works by finding every user present in the current image and tries to find the processes that those users are responsible for. This process is also capable of discovering if these processes are suspected of having or being part of malware and if they have any type of inbound or outbound connection. Because this process verifies all of these points it can take some time to display results.

Due to all of the previously described points and due to this being a big part of the RAM images investigation, this functionality wasn't included in the tutorial. Despite being present with any installation of the VolatilityGUI, it was decided not to mention this functionality as much as the others. This is because it gives a shortcut to the conclusion of many investigations that can lead to flawed results. The purpose of this tool was always to help the user and not do the work for him. This shortcut is available more to confirm results and not to give a shortcut to the final conclusion of the investigation. So, after finishing running, this process also gives a warning message that this is just a feature of VolatilityGUI and that any value should be confirmed.

Besides all of the developed tools other small QoL improvements were made like:

- Tooltips on every textbox and button.
- A responsive GUI that automatically hides unused functionalities.
- Several warning messages and confirmation messages.

After describing every QoL development, the next sub-chapter is about everything related to the Saving and Loading of the project.

4.7 SAVE PROJECT

This subchapter will discuss how the VolatilityGUI handles the saving/loading of the project and all of the details related to this point. A normal RAM investigation can take several days. Due to this, and to give the ability to pass the investigation to other users, it was necessary to create the ability to save the project and to load it.

The implementation of this feature was the creation of a text file that saved the location of the imported RAM image and every currently opened tab with every piece of information inside of each tab. Other data to save inside of the file was considered but ultimately not implemented due to the size of the save file and the amount of processing power to save and load this file. The save function works by writing the data inside of the file and, if the project has been saved before, the VolatilityGUI automatically saves on the previously used file. Like other similar software, the VolatilityGUI also counts with the "Save As" function to create a new save file. The loading of the save file works by reading the file's contents and allocating this information in the correct places inside the developed software.

Because the information taken from the RAM image usually contains personal or sensitive information, it was necessary to protect this information, especially in the text file created to save and load the projects. The idea was to implement cryptography to the save file. On a first approach, the implementation was to use the native and easy-to-use cryptographic function from Windows OS, which uses the cryptographic service provider installed on the computer and the file encryption keys of the process being run in the code. This method was able to encrypt the file and decrypt it without the necessity to input any type of key or password. This method work by associating the encrypted key to the Windows OS user. Despite the good results with this method, it created some problems, including the inability to transfer the file to another user and the risk of losing access to the Windows OS account that created a specific project meaning the loss of access to the same project.

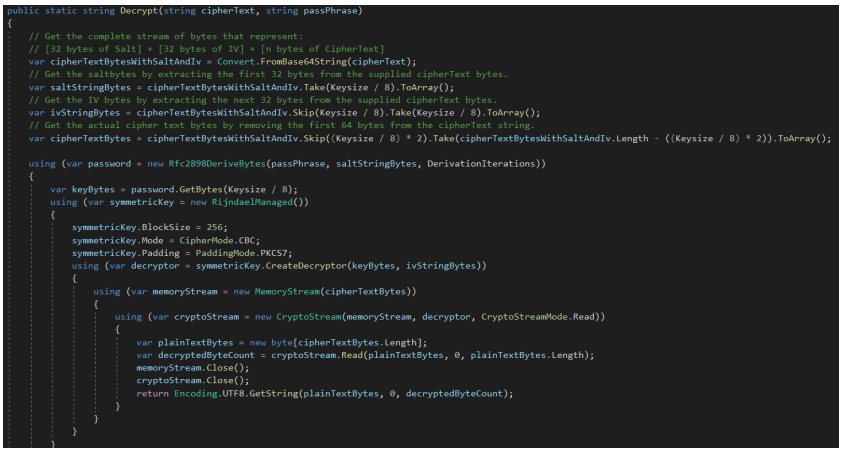
Because of this, a second and final implementation was made to solve the previously mentioned problems. This implementation uses the algorithm Advanced Encryption Standard 256 (AES256) to encrypt the text of the save file with the usage of a

provided passphrase by the final user. Advanced Encryption Standard or AES is a block cipher available in multiple sizes of key lengths and block sizes of 128bits. This algorithm is currently accepted as secure by most companies and governments, and due to its security, it was chosen as the algorithm to implement in the VolatilityGUI. The passphrase or password is asked every time before saving or loading the file. A verification is made to ensure that this key is not empty when saving the project. In the case of the project's load, verification is also made to check that the key input is the correct one. In Figure 28, it's possible to see the code responsible for encrypting the file.

```
public static string Encrypt(string plainText, string passPhrase)
{
    // Salt and IV is randomly generated each time, but is preprended to encrypted cipher text
    // so that the same Salt and IV values can be used when decrypting.
    var saltStringBytes = Generate256BitsOfRandomEntropy();
    var ivStringBytes = Generate256BitsOfRandomEntropy();
    var plainTextBytes = Encoding.UTF8.GetBytes(plainText);
    using (var password = new Rfc2898DeriveBytes(passPhrase, saltStringBytes, DerivationIterations))
    {
        var keyBytes = password.GetBytes(Keysize / 8);
        using (var symmetricKey = new RijndaelManaged())
        {
            symmetricKey.BlockSize = 256;
            symmetricKey.Mode = CipherMode.CBC;
            symmetricKey.Padding = PaddingMode.PKCS7;
            using (var encryptor = symmetricKey.CreateEncryptor(keyBytes, ivStringBytes))
            {
                using (var memoryStream = new MemoryStream())
                {
                    using (var cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))
                    {
                        cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);
                        cryptoStream.FlushFinalBlock();
                        // Create the final bytes as a concatenation of the random salt bytes, the random iv bytes and the cipher bytes.
                        var cipherTextBytes = saltStringBytes;
                        cipherTextBytes = cipherTextBytes.Concat(ivStringBytes).ToArray();
                        cipherTextBytes = cipherTextBytes.Concat(memoryStream.ToArray()).ToArray();
                        memoryStream.Close();
                        cryptoStream.Close();
                        return Convert.ToBase64String(cipherTextBytes);
                    }
                }
            }
        }
```

Figure 28: Cypher Code Block

As it's possible to see, the code uses a lot of already created functions from several Windows OS libraries to create the ciphered text. The code is commented in each step on what it does. A brief explanation is that this function takes as inputs the text, the user inputted key, the text to be encrypted, several random values, and uses the memory to create the encrypted text. The decrypt function works in a very similar way but in reverse to be able to decrypt the files. In Figure 29, it's possible to see the code responsible for the description of the file.



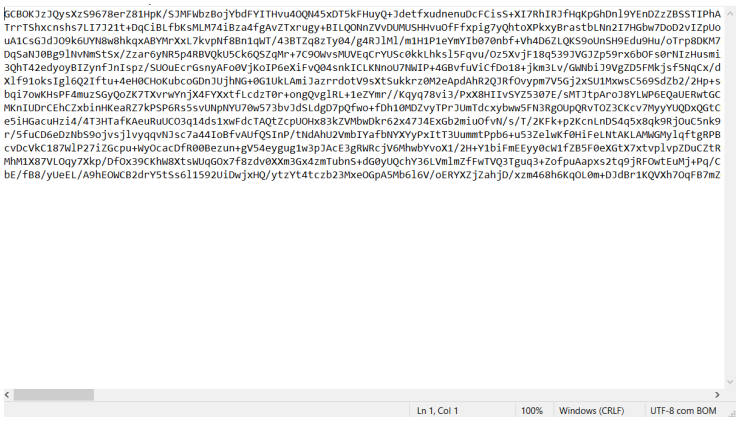
```
public static string Decrypt(string cipherText, string passPhrase)
{
    // Get the complete stream of bytes that represent:
    // [32 bytes of Salt] + [32 bytes of IV] + [n bytes of CipherText]
    var cipherTextBytesWithSaltAndIv = Convert.FromBase64String(cipherText);
    // Get the saltbytes by extracting the first 32 bytes from the supplied cipherText bytes.
    var saltStringBytes = cipherTextBytesWithSaltAndIv.Take(Keysize / 8).ToArray();
    // Get the IV bytes by extracting the next 32 bytes from the supplied cipherText bytes.
    var ivStringBytes = cipherTextBytesWithSaltAndIv.Skip(Keysize / 8).Take(Keysize / 8).ToArray();
    // Get the actual cipher text bytes by removing the first 64 bytes from the cipherText string.
    var cipherTextBytes = cipherTextBytesWithSaltAndIv.Skip((Keysize / 8) * 2).Take(cipherTextBytesWithSaltAndIv.Length - ((Keysize / 8) * 2)).ToArray();

    using (var password = new Rfc2898DeriveBytes(passPhrase, saltStringBytes, DerivationIterations))
    {
        var keyBytes = password.GetBytes(Keysize / 8);
        using (var symmetricKey = new RijndaelManaged())
        {
            symmetricKey.BlockSize = 256;
            symmetricKey.Mode = CipherMode.CBC;
            symmetricKey.Padding = PaddingMode.PKCS7;
            using (var decryptor = symmetricKey.CreateDecryptor(keyBytes, ivStringBytes))
            {
                using (var memoryStream = new MemoryStream(cipherTextBytes))
                {
                    using (var cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))
                    {
                        var plainTextBytes = new byte[cipherTextBytes.Length];
                        var decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.Length);
                        memoryStream.Close();
                        cryptoStream.Close();
                        return Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount);
                    }
                }
            }
        }
    }
```

Figure 29: DeCypher Code Block

With this, the user is able to transfer the file to another computer or another user if they have access to the used key without the risk of losing access to the file.

Another important point is that the user is not able to load any type of file in the VolatilityGUI. This is done with a kind of identification put in place inside of the save file that verifies the authenticity of the file as a VolatilityGUI save file. In the following Figure 30, it's possible to see the final result of the encrypted save file.



```
GCBOKJzJQysXzS9678erZ81HpK/SJMFWbzBojYbdFYITHvu4OQN45xDT5kFHuyQ+Jdetfxudnenu DcFCisS+XI7RhIRJfHqKpGhDnl9YEnDZzZBSSTIPhA
TrrTShxcnshs7LI7J21t+DqCiBLfbKsMLM74iBza4fgAvZTxrugY+BILQONnZVvDUMUSHHvuOfFfxpig7yQhtoXPkxyBrastbLNn2I7HGbw7DoD2vIZpUo
uA1CsGJdJO9k6UYN8w8hkqxABYMrXxL7kvpNf8Bn1qWT/43BTZq8zTy04/g4RJlMl/m1H1P1eYmYIb070nbf+Vh4D6ZLQKS9oUnSH9Edu9Hu/oTrp8DKM7
DqSaNJ0Bg9lNvNmStSx/Zzar6yNR5p4RBVQkU5Ck6QSZqMr+7C9OWvSMUVEqCrYUSc0kkLhksl5Fqvu/Oz5XvjF18q539JVGJZp59rx6bOFs0rNIzHusmi
3QhT42edyoyBIZynfJnIspz/SUOuEcrGsnyAFo0VjkoIP6eXiFvQ04snkICLKNnoU7NWIP+4GBvfuViCfDo18+jkm3Lv/GWNbiJ9VgZD5FMkjsf5NqCx/d
Xlf91oksIgl6Q2Iftu+4eH0CHoKubcoGDnJUjhNG+0G1UkLAmiJazrrdotV9sXtSukkrz0M2eApdAhR2QJRfOvypm7V5Gj2xSU1MxwsC569SdZb2/2Hp+s
bqi7owKHsPF4muzSGyQoZK7TXvrwYnjX4FYXxtfLcdzT0r+ongQvglRL+1eZYmr//Kqyq78vi3/PxX8HIIvSYZ5307E/sMTJtpAroJ8YLWP6EQaUERwtGC
MKnIUDrCEhCZxbinHKeaRZ7kPSP6Rs5svUNpNYU70w573bvJdSLdgD7pQfwo+fDh10MDZvyTPrJUmTdcxybww5FN3RgOUpQRvTOZ3CKcv7MyyYUQDxQGtC
e5iHGacuHzi4/4T3HTafKAeuRuUCO3q14ds1xwFdcTAQtZcpUOHx83kZVMbwDkr62x47J4ExGb2miuOfvN/s/T/2KFk+p2KcnLnDS4q5x8qk9RjOuC5nk9
r/5fuCD6eDzNbS9ojvsjlvyqqvNJsc7a44IoBfvAUfQSInP/tNdAhU2VmbIYafbNYXYyPxItT3UummmtPpb6+u53ZelwKf0HiFeLNtAKLAMWGMylqftgRPB
cvDcVkC187WlP27iZGcpu+WyOcacDfR00Bezun+gV54eygug1w3pJAcE3gRWRcjV6MhwbYvoX1/2H+Y1biFmEEyy0cW1fZB5F0eXGtX7xtvplvpZDuCZtR
MhM1X87VLOqy7Xkp/DfOx39CKhW8XtsWUqGOx7f8zdv0XXm3Gx4zmTubnS+dG0yUQchY36LVmlmZfFwTVQ3Tguq3+ZofpuAapxs2tq9jRFOwtEuMj+Pq/C
bE/fB8/yUeEL/A9hEOWCB2drY5tSs6l1592UiDwjxHQ/ytzYt4tczb23MxeOGpA5Mb6l6V/oERYXZjZahjD/xzm468h6KqOL0m+DJdBr1KQVXh7OqFB7mZ
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8 com BOM

Figure 30: Save File

The following sub-chapter, will describe how the software handles any type of available extraction.

4.8 EXTRACTIONS

As previously mentioned in the sub-chapter, it will be enumerated all the possible extraction of a file from the VolatilityGUI and how the software handles some of

the more complex extractions. The Volatility Framework is able to extract any kind of file present in the RAM image. To simplify the extraction of some type of files, the Volatility Framework has commands for this purpose. With these commands, the Volatility is able to extract:

- DLLs Files
- Memory Pages
- EXEs Files
- VAD Pages
- Event logs
- Drivers
- Registry Files

All of these items can be extracted from the loaded RAM image by accessing the two extract menus on the tab of the VolatilityGUI.

The two menus available related to extraction in the VolatilityGUI give the final user the ability to extract every file from the selected type present on the RAM image using the function "Extract All". The other option gives the user the ability to extract the selected type of file from a chosen process. This process is chosen by first selecting the type of file to extract in the function "Extract Selected" where a new window will open where the user can choose the process to extract the selected file from. Not every type of file is present in the option "Extract Selected " due to the limitations of the Volatility Framework commands. Some commands don't give the option to extract from a selected process inside of the RAM image and only give the option to extract the files from the entire RAM image in bulk. In both options, after selecting the type of file to extract, a Windows OS window will appear to record the save location of the extracted file/s. In Figure 31 and 32, it's possible to see all the available options in both extraction options.

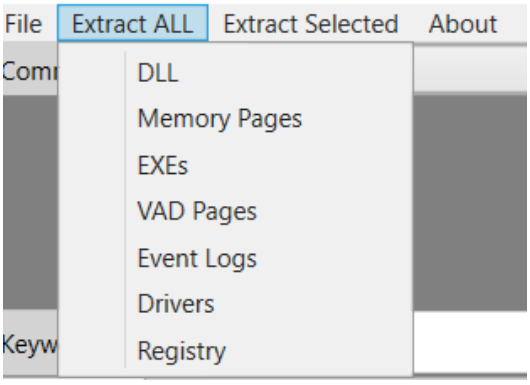


Figure 31: Extraction Options

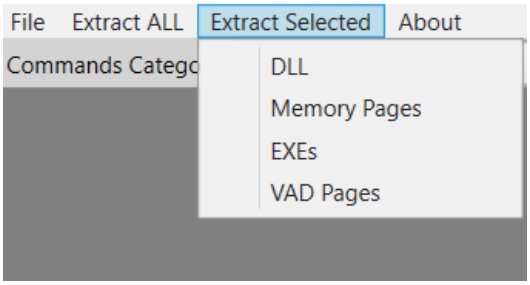


Figure 32: Extraction Selected Options

In specific and due to the options that the Volatility commands give, the extraction of selected files from the type of file "DLL" works differently from the others. When selecting this option a new window will open. This window unlike any others, gives the user many options of extraction. When opening the window, the VolatilityGUI will start to run some background processes that will fill the list of available processes in the RAM Image. The user can select a process from the list that gives two options: download every DLL file from the selected process by pressing the button "Download all selected PID DLLs", or fill the result list with every DLL from the selected process by pressing the button "Find PID DLL". This last option automatically fills the result list with a check box to give the user the ability to choose each DLL individually to extract. In Figure 33, it's possible to see an example of this function working.

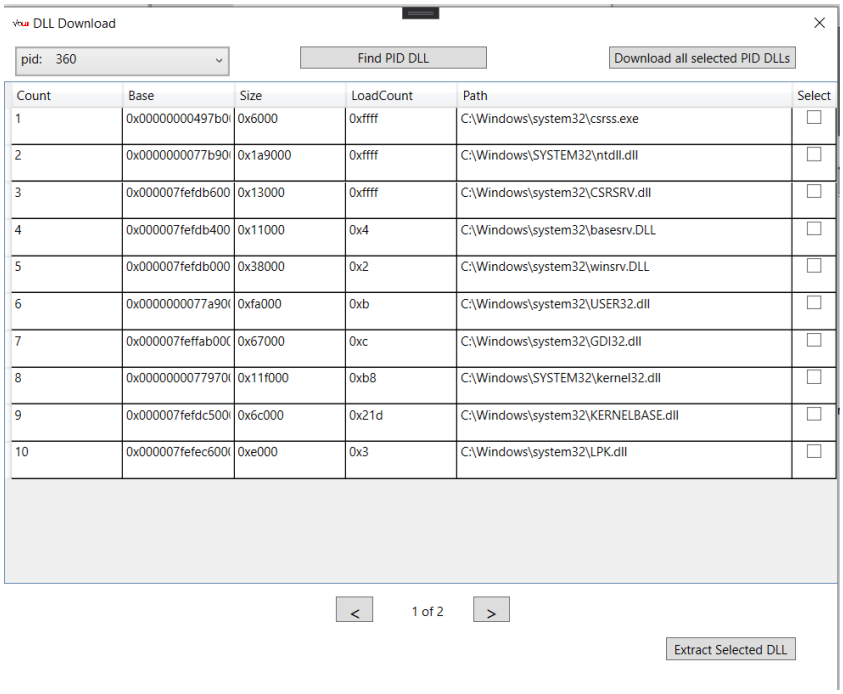


Figure 33: DLL Extraction Window

With this, the user can select all the intended DLLs from each page and press the button "Extract Selected DLL" to extract the intended file. After pressing this button, the function will follow the same process as the ones before, where it will open the window to select a save location. The only limitation is the VolatilityGUI inability to save the selected DLLs when changing pages. This means that when selecting a DLL from any page and then changing to another page, the previously selected DLL from the first page won't be saved or even extracted. In summary, the user can only extract the intended DLLs from a single page at a time. This limitation only exists due to the possible workaround being very heavy for the normal usage of the VolatilityGUI. This limitation is also explained in the Tutorial.

This functionality was created this way due to the sheer amount of DLL files per RAM image, giving the user a lot of possible options for extraction. And so, to simplify the process to the user, this specific window was created to maximize the level of detail that the user can choose when it comes to the DLLs and to facilitate the extraction of intended DLL files.

The following sub-chapter will demonstrate how the software automatically creates reports with the minimum possible input from the user.

4.9 EASY REPORT

The "Easy Report" functionality gives the user the option to create a full scoped report with very few steps. This function can automatically create a report that gives an overview of the selected topics, from the RAM image, by the user. This means that the user can get information from the RAM image without having to run a single command using VolatilityGUI.

This function can be accessed in two distinct ways. The first is the button "Easy Report" on the main window of the VolatilityGUI, while the second one is on the first window when opening the software and clicking the button "Easy Project". Starting with the first option, opening the window by clicking the "Easy Report" button will open a new window like the one in Figure 34.

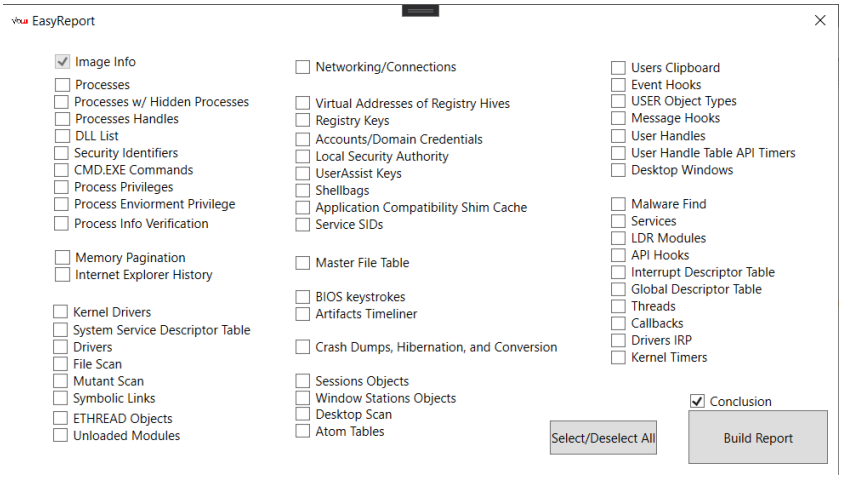


Figure 34: Easy Report Window

As it's possible to see, this window displays many options that will be discussed later. Clicking the "Easy Project" button will open a different window like it's possible to see in Figure 35. By analyzing both Figures 34 and 35, it's possible to see that the difference is that in the "Easy Project" Window, the user can select a RAM image. This difference will be detailed later in this sub-chapter.

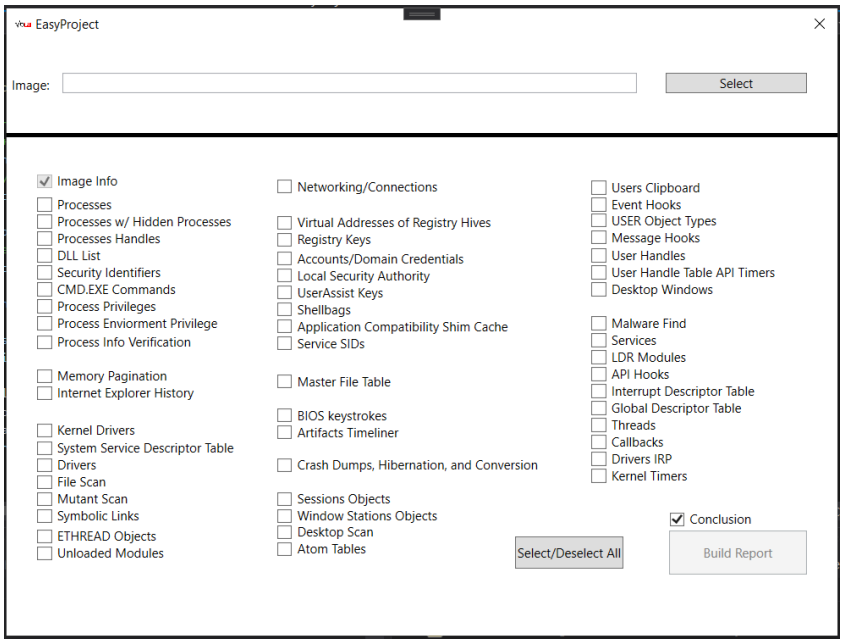


Figure 35: Easy Project Window

This was done to give the final user the ability to create a report without having to enter the main developed software and to create a report with all the necessary information without any complexity. This can be used to save time for an investigator that just requires a specific set of information from the RAM image. In the Easy Project is necessary to load a RAM image onto the software without going through the previously explained process of loading a RAM image. This simplifies the process and gives the ability to create a complete report with just one window.

The options present in both windows are the same, and the user can create a new report with any of them. The option of "Image Info" is mandatory due to giving basic information about the image. Each presented options on this window correspond to a different command. This decision was made to include the options in a more common language to help less experienced users using the reports function. For example, the option "Processes" corresponds to the command "pslist" from the Volatility Framework that gives basic information about the processes presented in the loaded RAM image. In contrast, other options like "Networking/Connections" are a mashup of several commands related to networking. This was done due to the similarity in the results of the commands and due to some commands only working on a specific version of the Windows OS. What results from this option is a mashup of all the available networking information.

Another report option that is not a single command is the "Conclusion" option. This option is a mashup of several commands that creates a formatted list of

the processes that are suspected of having/being malware. The list contains basic information about the suspected processes, like the process ID, the start time, the name, and several others, but also network information, privilege information, the DLLs present in the process, and the code that each process ran. Like the name of the option, this was created as a form of conclusion to the investigation and in a way to summarize the information that a normal investigation entails. Because of this, the "Conclusion" option is selected by default but can be unselected by the user at any time.

If the user wants to create a report with all the possible options, the button "Select/Deselect All" will select or deselect all the options depending on the previously performed action. When the user is satisfied with its choice of selected options, the next step is to press "Build Report" button. This will start the report process. This process can take a while, especially when the "Conclusion" option is selected due to being very processing heavy. To help with this point, while the process is running, a progress bar will appear that displays the total number of options that will be created in the report and the current number of options that are running. Because some commands take more time to process than others, the report process being at the halfway mark doesn't imply that it's halfway done, but it can give an estimate of the progress of the process.

What results after the process has finished running is a message box with the saved location of the report file. Clicking okay on this message box will automatically open the report. On the first approach, the report was created as a PDF-type file, but due to restraints with the engine that created this file, it was not possible to create the expected outcome, and later down the life of the project, there could be some licensing problems. On the second and final approach, it was decided to create a report as a webpage, which means that when the report process has been finalized, a webpage will open in the default browser application. Despite not being the typical PDF file, the final user can use the native capabilities of the browser to convert the webpage into a PDF filetype. Figure 36 displays an example of a created report.

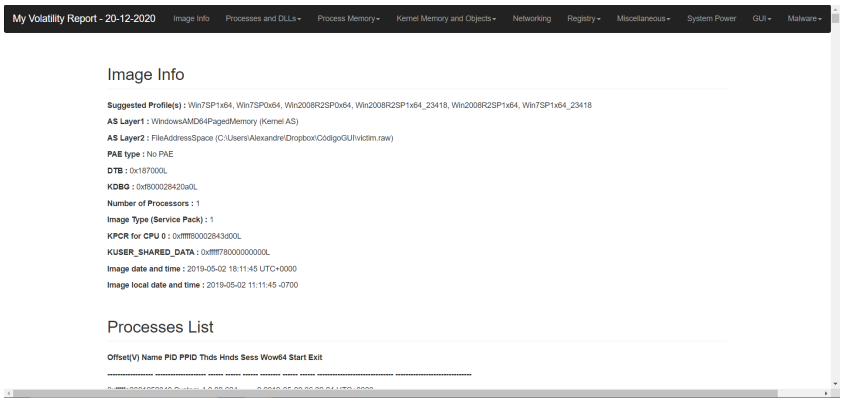


Figure 36: VolatilityGUI Report

As it's possible to see, by analyzing Figure 36, the report is formatted for each selected report option to facilitate the user's reading experience. But it also included a navigation menu that only consists of the chosen options that can automatically navigate to the selected option. The webpage is responsive, using bootstrap, meaning that it can be adapted to any screen size, including mobile devices. Some options, due to their size, are created in a completely separated webpage that can be accessed through the top menu. This was done to save space on the created main page. The top menu it's also configured for any additional created web page.

The next sub-chapter demonstrates another QoL function called "Historic" that records each action performed by the user.

4.10 HISTORIC

This sub-chapter demonstrates another QoL function that has been developed. This function goes by the mane of "Historic", and like many other similar software to VolatilityGUI, this function was created to record user actions.

This function can be accessed by the button "History" on the tab of the main window of the VolatilityGUI. This button will open a new window that will present a timestamp, according to the host computer and the action performed. This registered action depends on the actions performed by the user on the VolatilityGUI and can be anything from "Opened DLL Window" to "Run pslist command". In Figure 37, it is possible to see an example of this "Historic" Window.

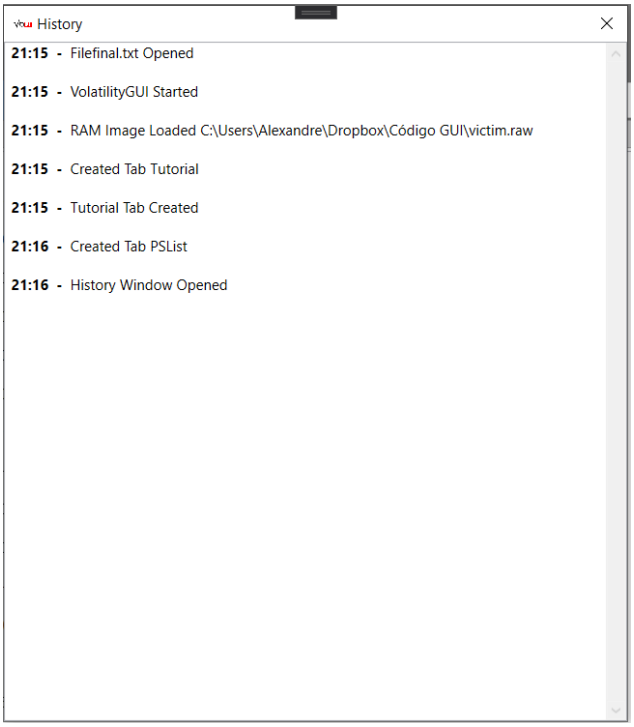


Figure 37: Historic Window

This function was developed to help the user to keep track of the performed action on the software. By choice, it was decided only to include a history of the performed action from the current session. Meaning that closing and opening the VolatilityGUI will always reset the history window, even when loading a previously saved file. This was also done to save processing resources when opening the saved project file.

The last sub-chapter of the development section it will explain how the integration with the VirusTotal website was made and how it benefits the final user.

4.11 VIRUSTOTAL API

During a normal investigation of RAM images, an antivirus is often used to search for any kind of malicious content in any present file. This task normally involves the extraction of a certain file and the usage of specific software to perform this task, and normally this process is very time-consuming. So in the sub-chapter, it will be explained how a solution was developed to try and circumvent this action without having to leave the VolatilityGUI and possibly save some time and clicks.

The solution involved doing an integration between VolatilityGUI and VirusTotal. VirusTotal is a website that aggregates many antivirus products and online scan engines to check for viruses that can take any file or website. Due to the number of antivirus products, it is effortless to find any virus that other software may have missed or verify against any false positives.[20] VirusTotal was also chosen due to being the tool normally used by many investigators and to the availability of an API for easy integration.

To access this integration, with VolatilityGUI, the user must first, like many other functions, load a RAM image into the software. After this, the button "VirusTotal" can be found on the top nav bar, and clicking on it will open a new window. The window that will open can be seen in Figure 38.

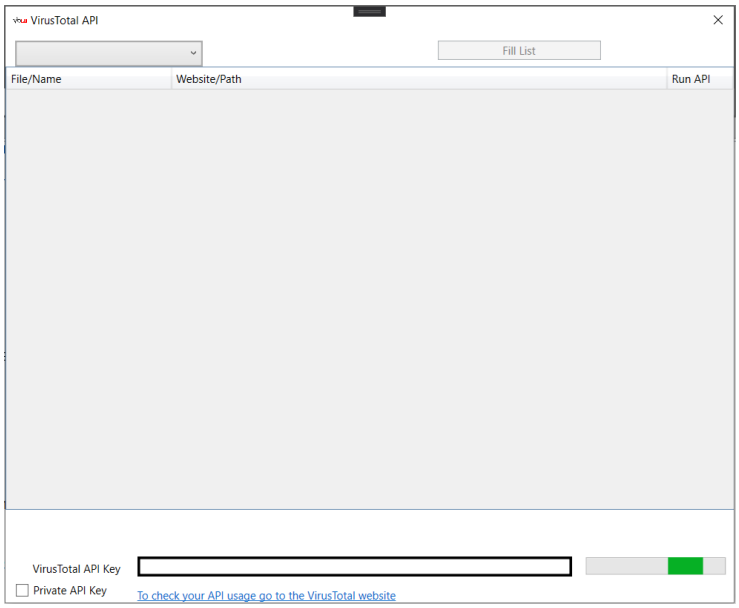


Figure 38: VirusTotal Window

This window works by selecting the type of file to be analyzed by VirusTotal. It can either be a file type EXE or DLL. By selecting the option of file type DLL the user can also choose the process related to the DLLs files. This was done due to the multitude of DLLs present in each process of the RAM image. This doesn't occur in the EXE file type due to being just one file per process, and by choosing this option, the list will display each EXE file from the RAM image. After either option is selected, the user can press the button "Fill List" to fill the list with the selected file type. The final result should be something similar to what is presented in Figure 39.

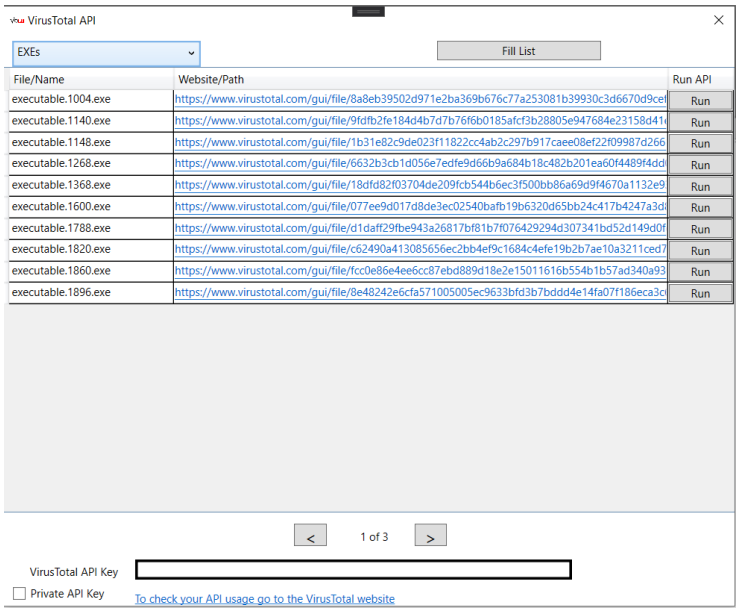


Figure 39: VirusTotal Window List filled

The list will be filled with the intended file where it will display the name of the file, the VirusTotal website of the file, and the button "Run" in each file in a pagination system if the number of total files is bigger than 10. This was done due to restrictions on the C# library that was used for this function. Meaning that the code limited the interaction of files if these were not visible to the user and that any interaction with a file that was not visible to the user would throw an error. The workaround used for this problem was limiting the number of records displayed each time to the user to a very small number, in this case, 10. The VirusTotal analysis website works by analyzing the file through several antiviruses and saving the results in the form of an URL. Each file is identified in the URL by their SHA256 hash, for example, if a file had the hash '123' the URL would be 'https://www.virustotal.com/gui/file-analysis/123'. The software automatically calculates the hash of each file and displays the URL of each file by merging the URL with the hash.

The way this window/integration is supposed to work is as follows. First, the user will click the URL of the intended file to analyze, and this will open the default web browser application. If anyone has previously scanned the file and within time limits, the results will appear. If this is not the case, the webpage displays a default error message. Then, to solve this, the user can return to the VolatilityGUI and click the "Run" button on the line of the intended file. This will lock the line of the file and start the API request. The process involves sending the file through the API to be analyzed. This process normally takes about 30 seconds, and after this,

the line will be unlocked, and the user can once again click the URL to open the webpage with the results. An example of these results can be seen in Figure 40.

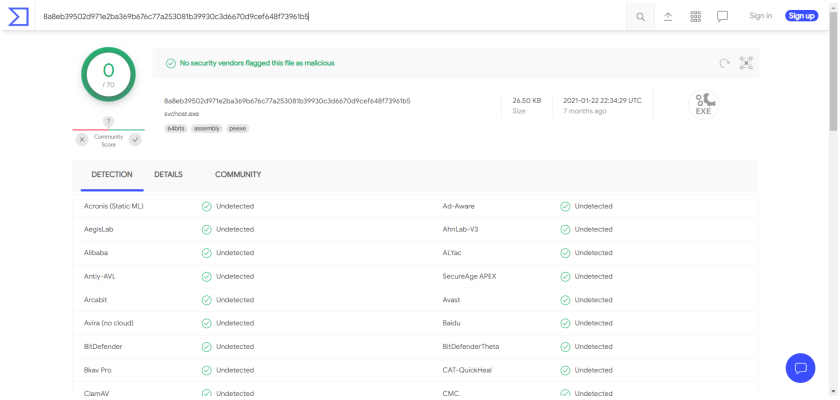


Figure 40: VirusTotal Website

To use the VirusTotal API, the user must first get an API key that can be requested by creating a new account on the VirusTotal website. This key can be inserted on the VirusTotal window, on the text box "VirusTotal API Key" that will be used to run the API. If the user has accessed a private API key instead of a public one, the "Private API Key" check box should be checked to remove any restrictions.

The VirusTotal API key restricts the user to 500 requests per day and 4 requests per minute. Each request is a click of the "Run" button. The user can run multiple requests at the same time on the VirusTotal window. The VolatilityGUI limits the user with 4 requests by minute, with a counter of 60 seconds from the start of the first requests, and displays the information of how many requests are currently running and displays error messages if this number is exceeded or even if the API key is wrong. The VolatilityGUI also controls the number of requests per session at a maximum of 500 to simulate the maximum number of requests per day. Meaning that closing and opening the VolatilityGUI will reset this counter. The user can, at any time, check the number of remaining requests on the official VirusTotal website.

There were some other approaches to this problem, but this solution was the best overall to minimize the number of requests made to the API. The first approach had the software automatically check if the file was analyzed before, but this meant that this verification per file would take most of the allowance of the public API key since it would be many files to be analyzed. Therefore, not only would the public API allowance be a problem, but also it would take a very long time to analyze everything due to the time restrictions. Another approach also involved the display of the antivirus analyses on the VolatilityGUI, but once again, this involved more

API usage and a much inferior displaying of the results than the VirusTotal website. The final solution was the one that was better in terms of API usage and time. The objective of this integration was to give any user the ability to use the VirusTotal functionalities with ease of usage and to streamline the entire antivirus analysis process.

Figure 41 presents a high-level diagram of the previously explained process's architectural communication between VolatilityGUI and VirusTotal. The order of the steps that the process follows can be found above the corresponding steps in the diagram.

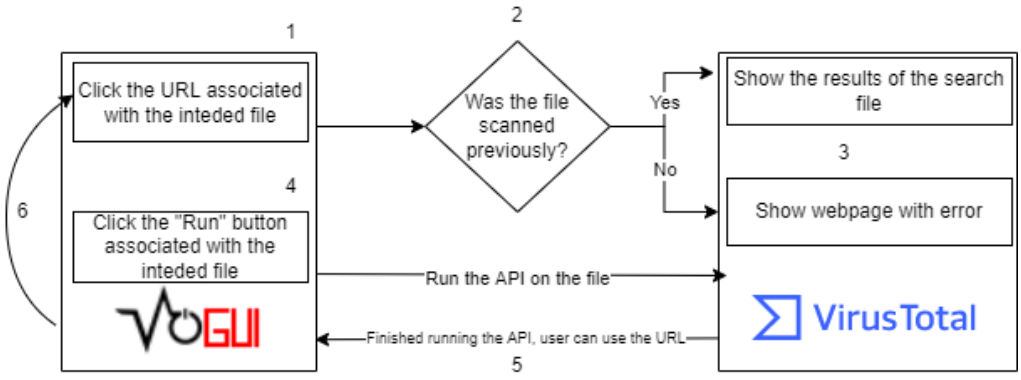


Figure 41: VolatilitGUI VirusTotal Communication Diagram

This concludes the VirusTotal integration explanation and the development section. In this section, every developed function was presented in more technical detail, but it was decided not to present blocks of code to decrease the technical level necessary to understand this document in its entirety. The code will be available in a private project on GitHub and consulted at any time with the following link VolatilityGUI. The following section will contain the tests performed on this project and the results.

4.12 SYNTHESIS

In this chapter, it was explained every decision taken on the technical development of VolatilityGUI.

This chapter is separated by functionality, where it's explained all of the developments. This includes every implementation before the final one. Every functionality section includes every implementation done, the problems faced, the reason behind said functionality, and what each functionality helps the user achieve.

Every functionality is detailed in technical terms, including screenshots of how they look inside of VolatilityGUI. This includes not only functionalities of VolatiltyGUI but also decisions on the choice of environment and how the connection to Volatility works. In summary, this chapter explains how VolatilityGUI works and the choices behind each functionality. This results in a GUI capable of doing everything that the Volatility Framework is capable of and more. VolatilityGUI is a C# developed GUI capable of analyzing and loading RAM images. It's capable of running every Volatility command and helps the user find specific keywords in the results of these commands. It gives the user a simple way to analyze the results on multiple commands, and it's capable of saving these results so the user can load them later. VolatilityGUI is also capable of extracting different types of files with an easy-to-use interface. Moreover, the user can check the history of taken actions in a simple list where the action is described and the time that the action was taken. The user can also build reports that can exclude the user from running any kind of command. These reports are automatically built and shown to the user in a dynamic way. VolatilityGUI is also capable of joining processes with users, something that is very important to any investigation. And finally, the user can also analyze files for malware without having to leave the software with integration with Volatility GUI.

The evaluation of each functionality is going to be analyzed by tests in the next chapter.

5

TESTS & RESULTS

This section contains all the tests performed on the VolatilityGUI. Because this project is a development of an already existing tool, tests were done comparing the two tools to prove the improvement made over the already existing tool. This section is divided into subsections, each one talks about the performed tests and finishes with a critical analysis of the results.

5.1 USER TESTS

In this subsection, it will be discussed the user tests done related to the VolatilityGUI. These tests were done by providing several users with a guide that follows every functionality inside of the VolatilityGUI. At the end of this guide, a questionnaire was provided with questions that compared the Volatility command line with the VolatilityGUI to determine what software the user would rather use. The questionnaire also asks what the user thinks about the VolatilityGUI functionalities and if these are hard to find or to work with. Because a lot of the questions from the questionnaire compare both Volatility and VolatilityGUI, a prerequisite for all of the users that answer the questionnaire was that they should have experience with both tools. This guide was provided in a shared folder containing the guide in both document format and website format, the setup of the VolatilityGUI, and a RAM image that could be used to test the software. This guide was written in Portuguese and contained pictures in each step of the expected result.

The analyses of this questionnaire and the results of each question can be found in the last subsection of this chapter and also a critical analysis of the obtained results. The link to the questionnaire can be found at the end of the provided guide. This guide can also be found in Appendix B of this document.

5.2 REAL USER TESTS

This subsection will explain how the real user tests were done. This test involved the usage of technical users with Information technology (IT) knowledge and with non or little experience in the digital forensics area. These users followed a small set of tests using both the Volatility Framework and VolatilityGUI with the objective of comparing the time that each task would take in both tools.

These tests involved every main point from both software, Volatility, and VolatilityGUI, from the first usage of each tool to simple commands, extractions, and many other functionalities. Any functionality that was exclusive to either tool was not included due to a lack of comparing the times that each user would take to complete each task. Each task had the maximum time to be finished of 5 minutes due to time restrictions of doing the maximum amount of tests possible. The guide that contains the tasks performed by the users can be found in Appendix C of this document.

This was done to understand the amount of time a user could save using VolatilityGUI instead of Volatility. This was also done to try and find if the users could do tasks that would be impossible for them using only Volatility. The users who performed this test were also asked to do the same questionnaire as those who followed the tutorial. This was done to get an opinion on both of the tools and especially on VolatilityGUI. Any extra functionality not used during the timed tests was demonstrated to the user before the questionnaire.

The methodology behind this test was to give the user a first question with the objective of "starting up" both software. After the first question, a quick interval is given so the user could familiarize with both software and ask any necessary questions. This was done to simulate a real scenario where the user could explore the software before starting to work with the software as intended. Each question asks to perform a simple task that is recurrent in any RAM investigation. And while the last question is a bit more complex, more time is given to finish the given task. The user performing the test will always have access to the internet to get any kind of information. It will also have the Volatility Framework Foundation website opened from the start. This website contains information about every command available in Volatility. The developer of VolatilityGUI will also be available to answer any question or doubt during the test, but it will not give away the answers. Even if a task is not completed in the provided time frame, a solution was given just to help

out the user for the following steps and so that any user that fails the first tasks can learn and try to solve any remaining tasks.

The results of this test can be found in the following sub-chapter, with the average time that each task took to be completed, following a critical analysis of these results.

5.3 CRITICAL ANALYSES OF THE RESULTS

In this subsection, it will be demonstrated a critical analysis of the results obtained from both tests. Starting with the results from the timed test and following up with the results from the questionnaire since both tests fulfilled this step. Starting with the results and analysis of the timed tests.

5.3.1 *Results of the Timed Test*

In the first question of the timed test, the user objective was to retrieve basic information from the RAM image. This was to test how long the user would take to open each software and run a basic command. It was also done to prove the difficulty that a user can have in using the command line. So in the results, it was possible to see that on average, using Volatility, the users took 5min, and using VolatilityGUI on average took 3min12s, which means that an improvement of 56,84% was obtained by using VolatilityGUI. This means that all of the tested users were unable to complete the first task in the available time using Volatility. At the same time, with VolatilityGUI, they were all able to complete it, meaning it was much easier to launch VolatilityGUI. All of the users had problems understanding how to run basic commands using Volatility. After this question, it was given some time so the users could learn each software and ask the necessary questions.

The second question focus on the user running one of the more common commands in Volatility. This was done to exemplify how normal commands run in both software and the ability to find the required command for the task at hand. This task also serves as a way to exemplify how they must backtrack to a webpage to find any command. So in this test, it was possible to see an improvement of around 665,35% with the task on Volatility taking 3min43s and on VolatilityGUI 31s. While the users were able to complete the task in both tools, the improvement while using

the VolatilityGUI was massive, meaning that the users had much fewer problems while running simple commands.

The third task involves finding another command but this time while using a different profile. This highlights the backtrack necessary to change into a new profile and once again find the necessary command. In this task, the average time to solve it in Volatility was 1min7s, and in VolatilityGUI was 1min35s with a change of -31,26%. During this task, most of the users were able to complete it quicker using Volatility, this is due to the way VolatilityGUI works. Because this task involved the change of the working profile, something that is not recurrent in an investigation, it was decided that the results were enough to sustain and keep this functionality working like this.

The fourth question is more about general knowledge of how to work with other tools. This task is asked to find a specific string from the results of the last task. This highlights the necessity of the user to know or search for the necessary command to find the required string. On Volatility, this task took on average 3min29s, and on VolatilityGUI 39s, where an improvement of 989,84% can be seen. This task tested the ability of the user to use the Windows OS command line. Because this is a functionality available in VolatilityGUI and much easier to the user it's possible to see a massive improvement. Unlike the Windows OS commands line, the available functionality doesn't require any previous knowledge meaning that a new user will have much fewer problems in using it to find a specific string.

The next question involves an extraction. The required task is simple and something common for any investigation: the extraction of a simple DLL file. This task was created to show how hard sometimes it can be to extract specific files from Volatility and how much easier it can be with VolatilityGUI. This task took 4min5s on Volatility and 2min2s on VolatilityGUI, with a change of 136,76%. All of the users were able to complete this task in both tools but found it much less troublesome to use VolatilityGUI. The users had problems with things like nonexisting directories, finding the specific file to extract, and other things. All of these problems were eliminated in VolatilityGUI with a graphical interface.

Task number six involves the extraction of another file and the analysis of this one. A typical investigation usually also consists of the analysis of multiple files using an antivirus. This task was created to show the user the necessity of using an external website to do this process and making it lose more time while VolatilityGUI contains the integration built-in. Due to the necessity of using an API key, this key was provided beforehand on the side of VolatilityGUI. So this task took on average

4min49s on Volatility and 1min3s on VolatilityGUI, with a change of 391,99%. Almost all the users were unable to complete this task. Many had problems with extracting the file to be analyzed and having enough time to run it by the VirusTotal website. A problem that was solved in VolatilityGUI due to the integration that was made to solve this problem.

The next question is about using Yara Rules. This task was created to show how hard this process can be using Volatility. This task took on Volatility 3min55s and VolatilityGUI 30s, showing a change of 760,72%. Once again, most users had problems with finding the correct command and how to use it. Despite all of the users being able to complete this task with both tools, there exists a massive improvement while using VolatilityGUI.

The eighth question asks the user to run the command 'hashdump'. This command is used to find hashes of passwords from local accounts. This question was created due to the complexity of building the command in Volatility, requiring running previous commands, while in VolatilityGUI, the entire process is included on running the single command. It took on average 4min58s to finish this task on Volatility while it took 1min2s on VolatilityGUI, with an improvement of 399,57%. Most of the tested users couldn't complete this task, due to the complexity of the parameters required to run the asked command. Because VolatilityGUI is able to get the required parameters automatically, the time taken to do this task is much lower, meaning that a task most users couldn't do while using Volatility was easily done using VolatilityGUI.

Much like the previous question, question number nine focuses again on finding a specific parameter for the required command. Something that can be hard to do due to the output of Volatility. In VolatilityGUI, this becomes a more straightforward task because the possible required parameters are converted into a choice list to help the user in its choice. The construction of this command took 1min14s on Volatility and 30s on VolatilityGUI, with an improvement of 146,63%. While this task was able to be completed by all users in both tools, an improvement was seen while using VolatilityGUI. This was due to the ability to choose the required parameter while using the developed software. Volatility required the manual retrieval of some information to run the asked command.

The last standard question, number 10, is the simplest because it doesn't involve writing any command but finding the necessary command to perform the required task. The purpose of this question is to show how easier it is to find the information on the VolatilityGUI instead of the Volatility. Since the objective of this task is to

find a command, the replication of this task could impact the second tool of the performed test. To solve this, the user taking the test must show the description that proves that the specifically chosen command is responsible for doing what the task asks. So in this task, on average, the user using Volatility took 2min34s, and using VolatilityGUI took 27s, which shows a change of 503,33%. While this task was the simplest, some users had problems with finding the specific command asked in this task. While the improvement is massive, the value should not really be considered due to the advantage that VolatilityGUI could take on this task while being the second tool to be tested. While this is true most of the users had much fewer problems finding the information about a command while using VolatilityGUI.

On this extra question, a more complex task was asked to be performed. This task involves the output of several commands to find which users are responsible for each process. This task was created to highlight a developed functionality created for VolatilityGUI, but that is something normally done by investigators, so it was relevant to be included. Because of the complexity of this task and because this task normally takes several minutes and more advanced knowledge to accomplish, this task won't be considered for any of the final results and analyses. And due to time restraints, it was not possible to get the required amount of data to take into consideration. Despite this, the question will be kept in this document and in the timed test for any future reference or future work.

In general, it was possible to see an improvement of 401,99%. Meaning that it is much faster to use VolatilityGUI for an investigation and that results can be obtained faster and easier.

This concludes the timed tests that evaluate the improvement in terms of time to use VolatilityGUI. The following section will be discussing the questionnaire, which means evaluating what tool the users would rather use and how they feel about all of the functionalities of VolatilityGUI.

5.3.2 *Results of the Questionnaire*

The questionnaire was divided into 20 multiple-choice questions to ask several opinions about the VolatilityGUI. 2 Control questions are asked just to verify if the questionnaire was completed and 3 open answer questions just to give some final opinions. In each paragraph, it will be analyzed the question that was asked and the results.

The control questions asked if the tutorial or tests have been completed and how much experience the user that if fulfilling the questionnaire has with Volatility. 100% of the users answered that they had completed the tutorial. The average experience of the users was 1,5, which means that from all of the users who answered the questionnaire, the experience between 1 and 2 years was the average. In Figure 42 is possible to see a distribution of the given answers.

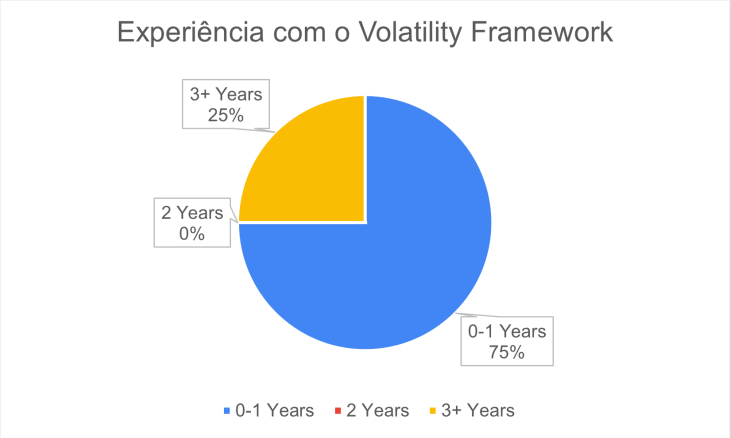


Figure 42: Question - "Experiência com o Volatility Framework"

All of the multiple-choice questions have as options the numbers 1 through 5, with 1 being bad or No and with 5 being good or Yes.

The first question asks the user if they would rather use VolatilityGUI instead of Volatility. This was done the get a general knowledge of what the user would rather use. The average result is 4,75 . This means that most users would prefer the usage of VolatilityGUI since the average of answers was 5. This means that almost all of the users prefer the developed software. Figure 43 shows a distribution of the given answers.

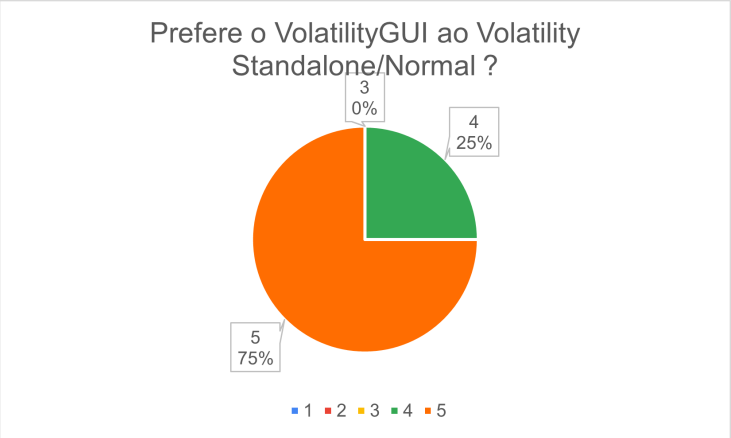


Figure 43: Question - "Prefere o VolatilityGUI ao Volatility Standalone/Normal ?"

The second question involves asking the user if they had trouble finding functionalities. The justification behind this was to see if there was any improvement that could be made to the tool's design. The result says that 3,5 is the average answer. Some users had problems finding some of the tools, and this is reflected by the answers given. The Figure 44 shows a distribution of the given answers for this question.

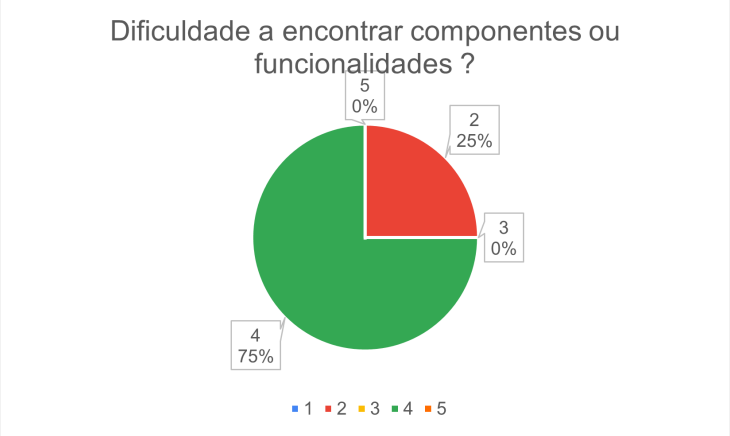


Figure 44: Question - "Dificuldade a encontrar componentes ou funcionalidades ?"

The software's velocity was considered during the development of the VolatilityGUI, and this was the focus of the third question. This question was created because velocity is something important in an investigation. The average result is that 4. This means that most users were satisfied by the speed of the tool and its functionalities. Figure 45 has the distribution of the given answers.

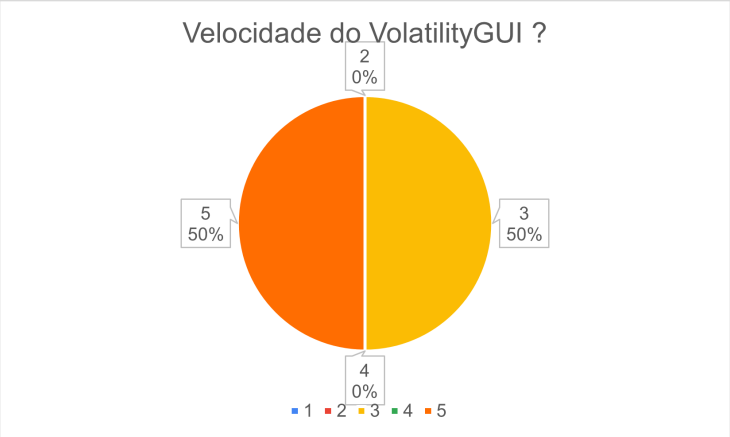


Figure 45: Question - "Velocidade do VolatilityGUI ?"

The next question focus on how the user perceives the design of the software. The result tells that 4,25 is the answer that is given most of the time. There existed

some restrains in the design of VolatilityGUI due to the technology that was used to develop this software, despite that most of the users were satisfied with the outcome. Figure 46, presents a distribution of the given answers.

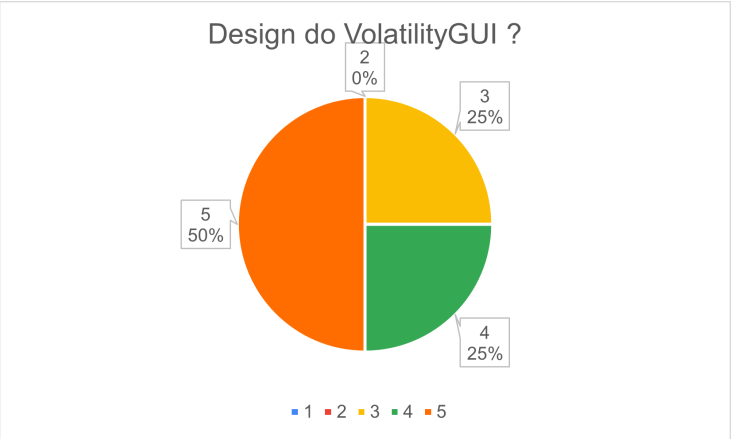


Figure 46: Question - "Design do VolatilityGUI ?"

The fifth question asks the user if they found it helpful to be able to use the Tutorial on the first tab and the information about every command useful. This question was formulated due to this information on Volatility being a bit harder to find. The result says that 4,25 is the average between the answers. This means that the average of users could take something useful away from the Tutorial presented in the beginning. Figure 47 contains the distribution of the given answers for this question.

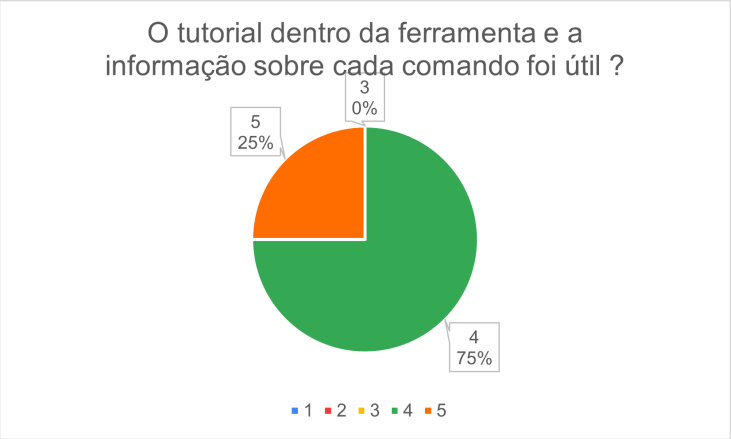


Figure 47: Question - "O tutorial dentro da ferramenta e a informação sobre cada comando foi útil ?"

The next question focus on the "Find" functionality inside of the VolatilityGUI. This question was created because normally, the ability to find strings inside of

text is hidden behind the knowledge to know how to use Windows OS commands, something that this software tries to solve. The result is 4,5. With the average of the answers almost at 5, this proves that there is a necessity to find certain keywords in the outputs of the command in an easier way, and the answers to this question prove that. The distribution of the given answers can be seen in the Figure 48.

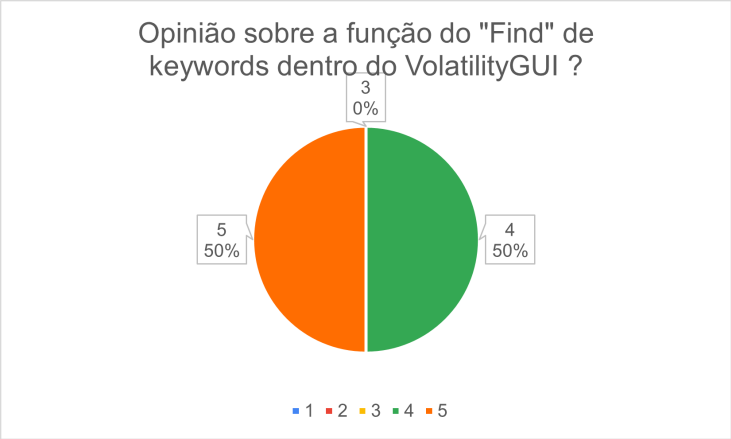


Figure 48: Question - "Opinião sobre a função do "Find" de keywords dentro do VolatilityGUI ?"

The seventh question talks about the extraction of selected DLL files. This functionality was created due to the large number of files contained in each RAM image and its difficulty in extracting the required ones. Due to this, this functionality became very important, and this question focuses on this. The result says that 4,75 is the average. All of the users found it helpful to have an easier way to select and extract, in this case, DLLs, from the image, and the developed functionality provides that, and the users seem to approve it. Figure 49 presents the distribution of the given answers.

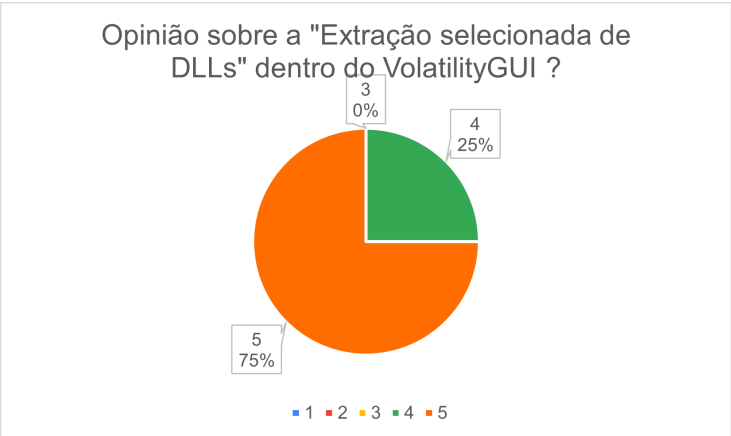


Figure 49: Question - "Opinião sobre a "Extração selecionada de DLLs" dentro do VolatilityGUI ?"

The integration with VirusTotal came about with the finality to try and save time for the final user when checking files for malware. This question focuses on this integration that made the user not have to leave VolatilityGUI to be able to scan a file and check for any presence of virus or malware. The result says that 5 was the only given answer. A normal analysis with VirusTotal requires a lot of steps. In VolatilityGUI, this was reduced to some clicks, and the users who answered the quiz seem to prefer it, with the only answer given being 5. In Figure 50 is possible to see a distribution of the given answers for this question.

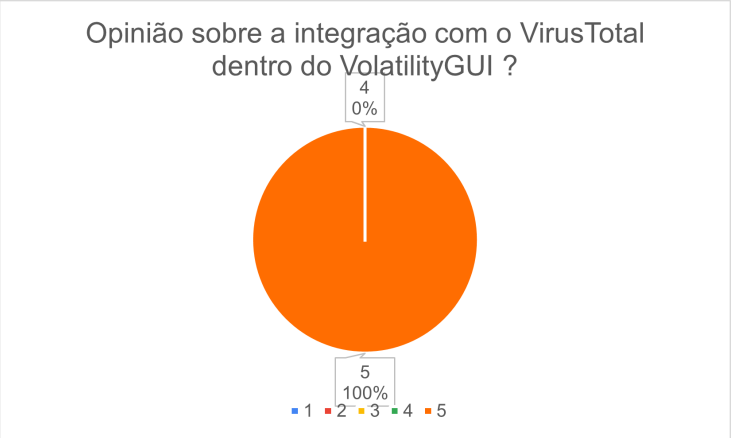


Figure 50: Question - "Opinião sobre a integração com o VirusTotal dentro do VolatilityGUI ?"

The ninth question is about the functionality "Forensic Investigation". Like previously mentioned, this functionality is in a very early development stage, but it was still necessary to get some feedback from the users so it could be improved in the future. The result says that 5 was once again the only given answer. Despite being experimental, this functionality can give an overall idea of the investigation. The users liked it since it can save some time and reach conclusions easier and faster. In the following Figure 51 a distribution of the given answers is presented.

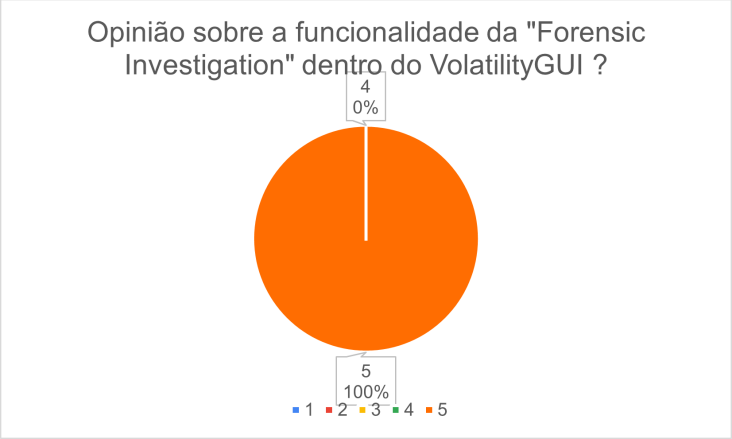


Figure 51: Question - "Opinião sobre a funcionalidade da "Forensic Investigation" dentro do VolatilityGUI ?"

At the halfway of the questionnaire, the user is questioned about saving and loading any project on VolatilityGUI. Because this is something that currently can't be done in Volatility and nowadays is something required in almost every software, it was important to ask this, especially users who have been using Volatility how they felt with this ability to save and load the previously done work. The result says that 5. The save and load of projects is common in almost any kind of software, so sometimes users can be surprised by the lack of it. Because of this, users found that 5 was the only answer to give. In Figure 52 the distribution of the given answers can be seen for this question.

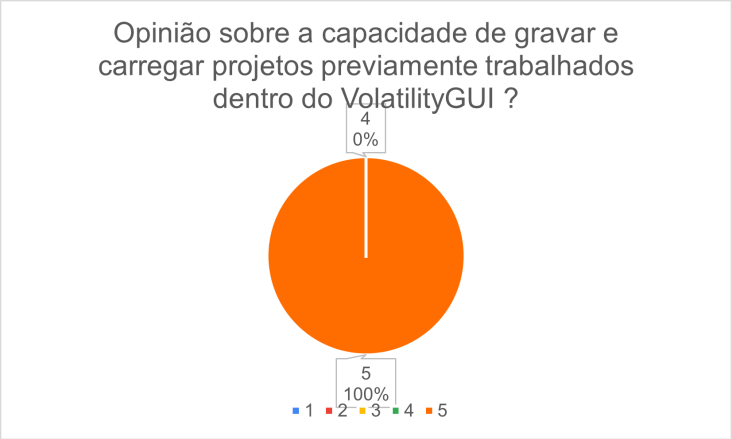


Figure 52: Question - "Opinião sobre a capacidade de gravar e carregar projetos previamente trabalhados dentro do VolatilityGUI ?"

The eleventh question probably focuses on what differs the most from the Volatility: the ability to "Load" RAM images into the software. While using Volatility to run the RAM image, it is only necessary to reference its location, so it was essential to

get feedback on how this works in VolatilityGUI. The result says that 4,75 was the average in the given answers. With these answers, most users were satisfied with a load of a RAM image into the software works, despite being very different from the normal Volatility. Figure 53 contains a distribution of the given answers.

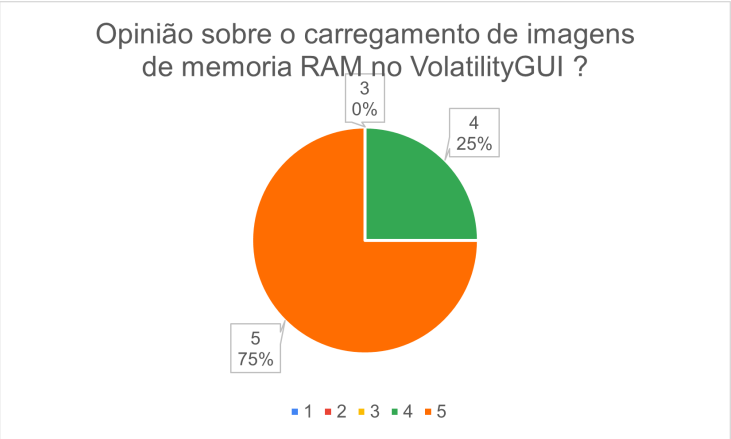


Figure 53: Question - "Opinião sobre o carregamento de imagens de memoria RAM no VolatilityGUI ?"

The next question asks about the ability to create reports. This was an idea that existed from the start due to many software from the digital forensic analysis having the ability to create this kind of report, so it was necessary to see how the users feel about this way to produce content. The result says that 5 was the only given answer. One of the very earlier ideas was received by the users with excellent feedback, and the users liked the ability to build reports and the options that were given. In the following image, Figure 54, a distribution of the given answers is presented.

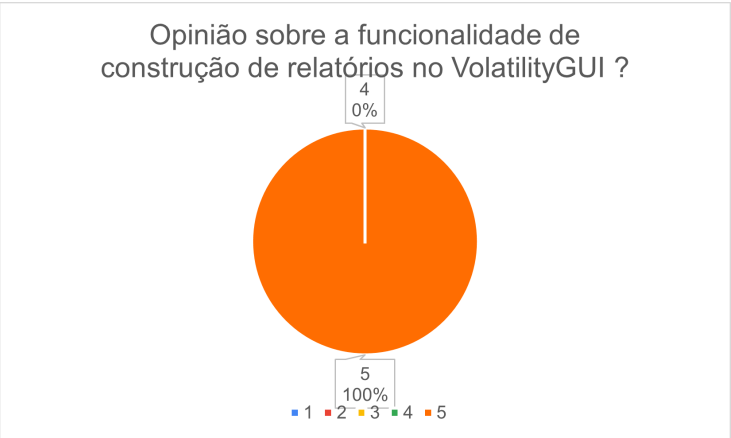


Figure 54: Question - "Opinião sobre a funcionalidade de construção de relatórios no VolatilityGUI ?"

The thirteenth question is focused on the created report file. Because the report is something that normally is very appellative to the final user, it was taken into consideration how the users felt from the outcome of the report. In terms of structure, design, formation, and functionalities. The result says that 4,75 was the average. Most of the users were satisfied with the output of the built VolatilityGUI report. In Figure 55 is possible to see a distribution of the given answers.

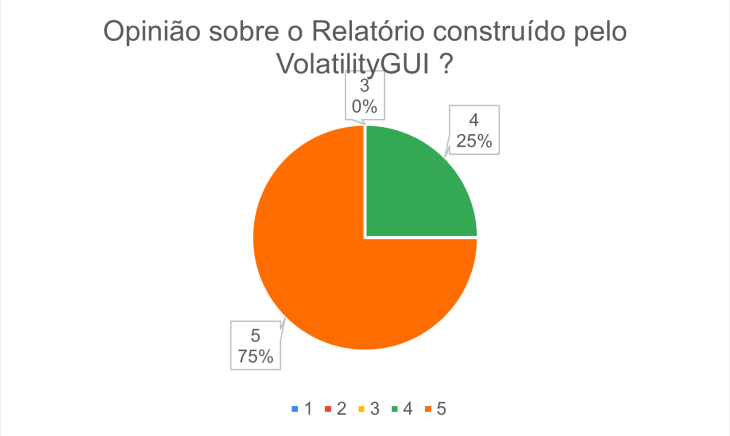


Figure 55: Question - "Opinião sobre o Relatório construído pelo VolatilityGUI ?"

The next question asks the user how they felt about the choice of commands. Like previously mentioned, the commands had to be divided by categories, something that a more experienced user might not be used to. So the question focus on how these users felt having these commands distributed this way. The result says that 4 was the average between the answers. Due to the sheer number of commands that Volatility has available, it was impossible to simply put all of the available commands in a single box, so it's good to see that most users were not dissatisfied with this separation by categories. Figure 56 displays a distribution of the given answers for this question.

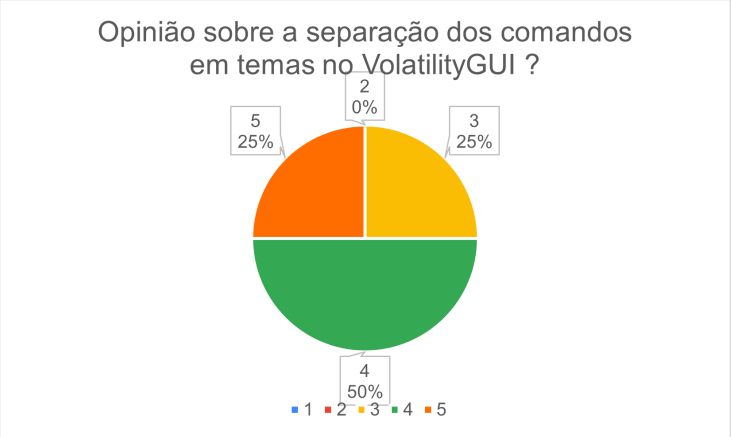


Figure 56: Question - "Opinião sobre a separação dos comandos em temas no VolatilityGUI ?"

The fifteenth question asks about one specific category from the previous question: "Advance Commands". Because some commands require parameters and can't be run like other commands inside the VolatilityGUI, this specific category had to be created. This works very differently from Volatility in the command line, so getting the users' opinions about this topic was relevant. The result says that 4,25 was the average. Because sometimes it is hard to replicate a command line's options, it's good to see that the average of the users was satisfied with these commands. Figure 57 shows a distribution of the given answers.

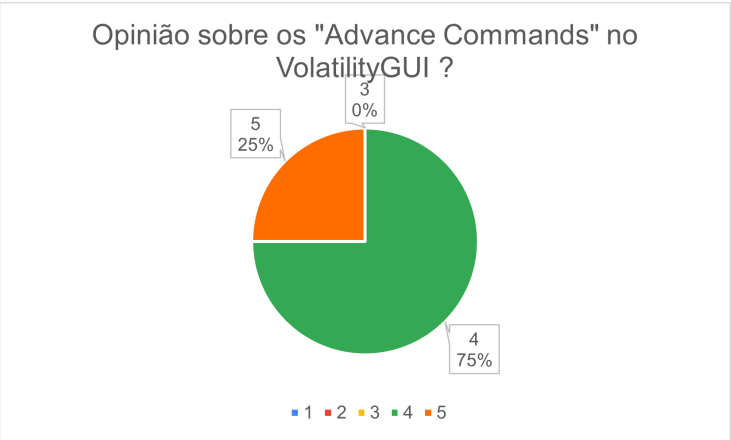


Figure 57: Question - "Opinião sobre os "Advance Commands" no VolatilityGUI ?"

The next question is about the ability to run manual commands in VolatilityGUI. Because it was important not to lose any functionality from Volatility, it was relevant to give the user the ability to run even more specific commands but still use all of the functionalities from VolatilityGUI. The average of the answers was 4,75. Almost all of the users liked the possibility to run manual commands inside of the

VolatilityGUI and use all of its functionalities. In the following image, Figure 58, gives a distribution of the given answers.

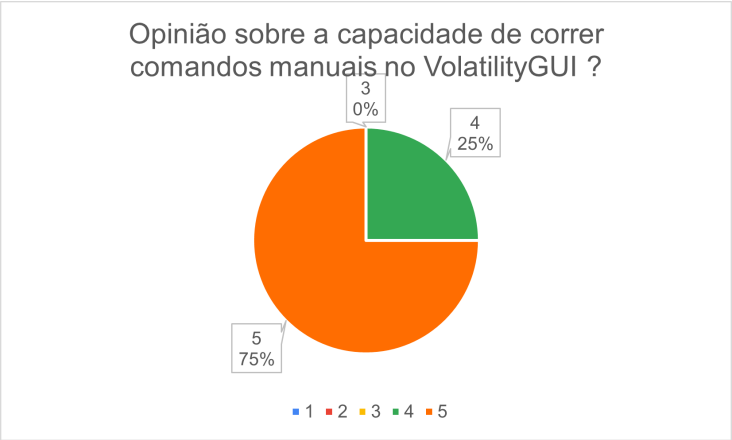


Figure 58: Question - "Opinião sobre a capacidade de correr comandos manuais no VolatilityGUI ?"

The seventeenth question is on the line of the "Advance Commands" category and asks if the ability to choose from a list of options for the necessary parameter. This was done to help less experienced users make fewer mistakes while choosing the required parameters for a command or if the time that these lists take to fulfill doesn't compensate for the usefulness. The average result was 4,5. Most of the users liked that all of the commands that require a second parameter were given the options for this parameter from a list since it leads to fewer errors and mistakes. Figure 59 contains a distribution of the given answers to this question.

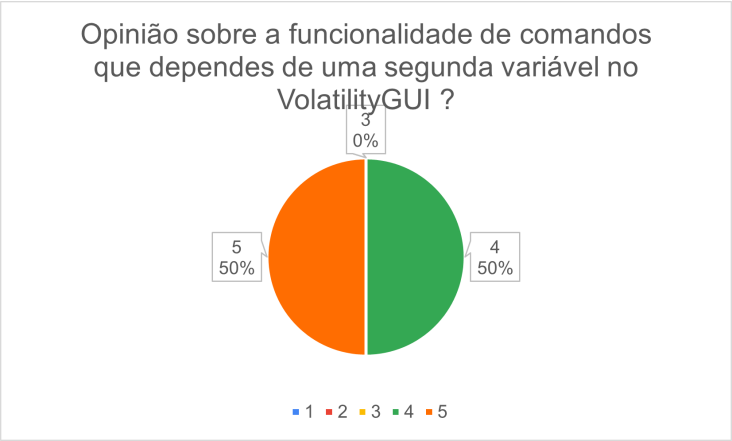


Figure 59: Question - "Opinião sobre a funcionalidade de comandos que dependes de uma segunda variável no VolatilityGUI ?"

The next question asks about the installation of the VolatilityGUI. Because Volatility can be somewhat complicated to start, the objective was to simplify this

and get feedback from the user about this topic. The result says that 4,75 was the average between the given answers. Especially the less experienced users liked the installation of the VolatilityGUI. This was probably due to being much more standard than the setup of Volatility since users are more accustomed to this. The feedback was almost the maximum possible. In Figure 60 it's possible to see a distribution of the given answers to this question.

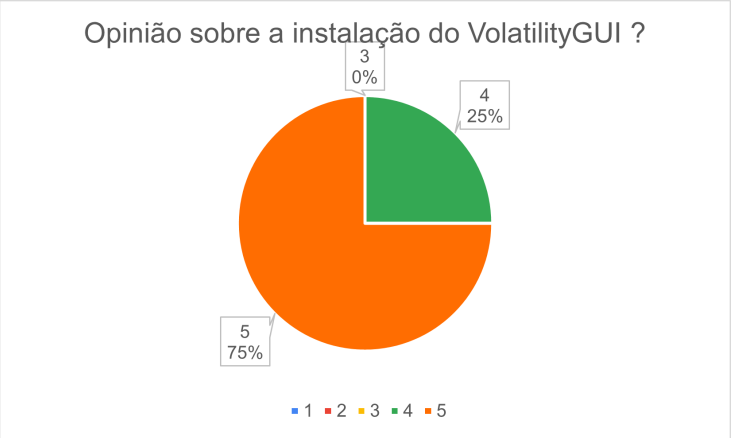


Figure 60: Question - "Opinião sobre a instalação do VolatilityGUI ?"

The next question follows the trend of the ability to save the project and asks the user if they feel that the save file is secured. This was asked to get some feedback about the security of the file and if this problem had to be even more developed in the future. The average was 4,75, which means that most users were satisfied and confident with the security of any personal data inside their saved projects' files. Figure 61 shows a distribution of the given answers.

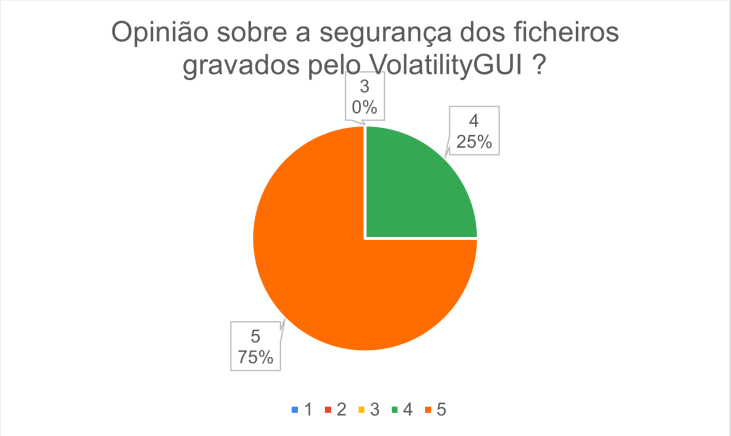


Figure 61: Question - "Opinião sobre a segurança dos ficheiros gravados pelo VolatilityGUI ?"

In the following question, the objective was to ensure that the "Historic" functionality was helpful to the user, to check previously done actions and their timestamps. The result says that 4,75 is the average. And this means that most users found it useful to see previously taken actions. In Figure 62 is possible to see a distribution of the given answers to this question.

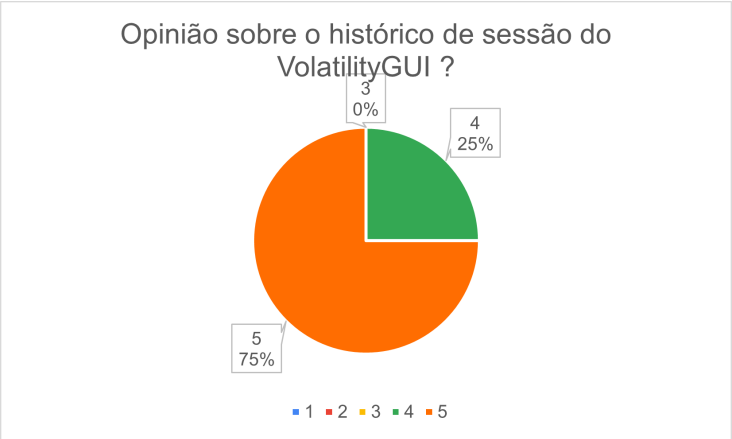


Figure 62: Question - "Opinião sobre o histórico de sessão do VolatilityGUI ?"

The last question from the multiple-choice focuses on asking the user if they will continue to use Volatility or if they would rather use VolatilityGUI. And the final result was 5 as the only given answer. This means that all of the users who took this quiz will use VolatilityGUI instead of Volatility. Figure 63, presents a distribution of the given answers to this last question.

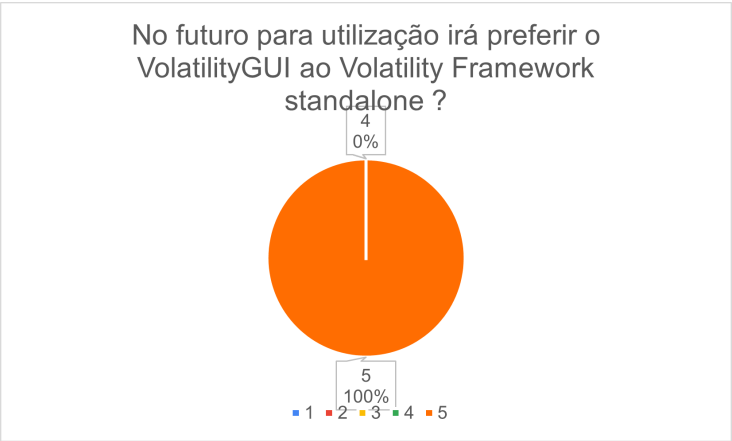


Figure 63: Question - "No futuro para utilização irá preferir o VolatilityGUI ao Volatility Framework standalone ?"

After all the multiple-choice questions, the focus was shifted to asking three open answer questions to the user. This first question was about problems or bugs

encountered during the testing of VolatilityGUI. The second question was about opinions and aspects to develop further, and the last question focused on any necessary or missing functionalities from VolatilityGUI.

In terms of bugs and problems, there was one single mention about the trouble of finding specific commands, something that can be resolved in future work.

In terms of things to develop further in the future, users wished there was more option to build reports, a more modern interface, and a scheme in the tutorial page with the structure between the commands category and the commands. Something can be easily achieved in future work.

And finally, in the question about necessary or missing functionalities that didn't exist in VolatilityGUI, the answers given were the inability to analyze RAM images of more recent operating systems. This solely depends on the Volatility Foundation, and of the writing of this report, there is no news on this functionality. Despite this, there exist some external teams creating plugins to solve this issue.

And this concludes the testing and results in the analysis chapter. In general, from the obtained results, it was highly positive, with most of the answers being almost close to perfect and the difference between the times to do tasks in both tools was on average four times faster. Meaning that any investigation or learning path in the area of RAM Forensic Analysis with this tool can save time, leading to more precise results and less time spent in each investigation. In summary, it can be considered that the development of the VolatilityGUI was highly positive and that in the future, it can be used instead of Volatility. This tool can be used in investigation centers, classrooms, and everywhere else. This also means that the required knowledge necessary to enter the area of RAM analysis is much less and that the learning curve is also much smaller than previously. It was proved that in several levels of experience, the usage of VolatilityGUI can speed up and improve any investigation, getting results that previously required a much deeper level of knowledge of the results of the commands.

5.4 SYNTHESIS

In this chapter, it was explained every test that has been done, including the results achieved.

Firstly, the tutorial was presented that users could follow to go through every functionality of VolatilityGUI and the questionnaire associated with it. The timed

test that users were submitted to was also presented, where users had to use both Volatility and VolatilityGUI to test their capability of solving simple tasks. The users who were submitted to this test were also invited to partake in the questionnaire. This questionnaire involved 20 questions where the answers went from a scale of 1 to 5.

Finally, this chapter ends with a critical analysis of the results obtained where it's possible to see that the users were both quicker and preferred the use of VolatilityGUI instead of Volatility. The results also show that users were satisfied with the functionalities exclusive to VolatilityGUI, where the results were overwhelmingly positive. This result translates into a faster adaptation of new users to VolatilityGUI, which means faster future investigations of RAM images.

6

CONCLUSION

The forensic analysis of digital devices will, without a doubt, continue to grow in the future. As a result, more and more investigation into RAM images will be necessary. Again, this area is where VolatilityGUI can shine.

This work involved the development of a graphical user interface for the tool Volatility Framework with several improvements.

In this work, it was proposed to develop a tool that could be an easy way to introduce new and less experienced users to the area of memory forensic analysis.

VolatilityGUI was elaborated and developed to reach the proposed objectives that could do everything an everyday forensic investigator needs and more without losing any function from the original tool. With specific contributions that VolatilityGUI could bring to this area, like modern functionalities missing from the original tool from the ability to create and load projects to the ability to get results in an easy to use graphical interface. This work explained what RAM forensic analysis is and its primary purpose. It also described the similar software already existing, and each contribution is brought to the RAM analysis area. Many existing tools, like VolatilityGUI, use the Volatility Framework for this process. Because of this, a detailed explanation of what the Volatility Framework is and how it works is also very important, which is what this work tried to show.

Because this work involved software development, it was necessary to explain the work and the architecture to be implemented, completed with mockups and user stories to assist with the development methodology chosen for this project.

Every functionality proposed to be done was achieved. This document explains every technical aspect of it in detail, from all the implementations done to the reasons for specific functionality to existing, to the reason behind each development. These functionalities not only will be what makes VolatilityGUI different from every existing tool, but it's also what the user can use to assist in an investigation. These developments were done to bring the Volatility Framework many needed functionalities that one could consider normal in any other type of software of digital forensic analyses area like, integrations with API, a graphical interface, the

ability to build reports, and many others. Despite this, in the end, what matters is the opinion of the user. That is why this document also addresses that point, with both a questionnaire and a timed test, which both give an overwhelmingly positive response. Meaning that users liked the already existing functionalities from Volatility and the unique functionalities of VolatilityGUI. And that by using VolatilityGUI, the user can be faster, on average four times faster, that could translate into an investigation be done in 1/4 of the time.

In conclusion, the result of this work was VolatilityGUI. A custom GUI for Volatility, easy to install and easy to use, that users like to use, faster than the Volatility Framework and with more functionalities. That anyone can use doesn't matter the experience and can be used to enchant the work of current Volatility users with an all-in-one tool that can do the work that previously had to be done by external software, like finding text or an antivirus analysis. That can help accelerate and streamline the entire process of any investigation.

In terms of future work, the continuity of this work will be the implementation of the GUI in a cross-platform environment and the introduction of Linux and macOS specific commands. The continuation will also fall upon the inclusion or development of more tools that could be used in a typical investigation. The developed software works with version 2 of Volatility but will not work with version 3 of Volatility due to the changes in how version 3 works. However, if version 3 ever releases as a standalone, VolatilityGUI can be easily adapted to this new version. This can also be future work to adjust the software to any new version. The continuation of this work could also involve the conversion of the VolatilityGUI to the new version, Volatility3.

Finally, another idea for future work is the inclusion of artificial intelligence on the VolatilityGUI. This artificial intelligence could help the user on each step, providing and highlighting important info and creating better reports.

To any developers that want to contribute to the development of this project, the following link contains the source code of VolatilityGUI, the setup, and all of the documentation: VolatilityGUI. Hopefully, this work can be improved upon or be the basis for a new tool, and this link to the VolatilityGUI GitHub can provide everything necessary to accomplish this.

In the end, everything that it was proposed to do was done, and VolatilityGUI is ready to be used in a real scenario with space to continue to receive support and to help grow the forensic analysis community.

BIBLIOGRAPHY

[1] Kristine Amari. «Techniques and Tools for Recovering and Analyzing Data from Volatile Memory». In: *SANS* (2009). DOI: https://www.sans.org/white-papers/33049/.

[2] Michael W. Andrew. «Defining a Process Model for Forensic Analysis of Digital Devices and Storage Media». In: *Second International Workshop on Systematic Approaches to Digital Forensic Engineering* (2007). DOI: https://ieeexplore.ieee.org/abstract/document/4155347.

[3] M. Hale Ligh A. Case J. Levy A. Aron. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. 2014.

[4] Muhammad Shamraiz Bashir and M. N. A. Khan. «Triage in Live Digital Forensic Analysis». In: *The International Journal of FORENSIC COMPUTER SCIENCE* (2013). DOI: http://www.ijofcs.org/V08N1-PP05-TRIAGE-IN-LIVE-DIGITAL.pdf.

[5] E. M. Chan. «A framework for live forensics». In: (2015). DOI: https://www.ideals.illinois.edu/bitstream/handle/2142/24365/Chan_Ellick.pdf.

[6] *Cybersecurity Using Volatility Framework For Analyzing Physical Memory Dumps*. URL: https://www.apriorit.com/dev-blog/662-cybersecurity-using-volatility-framework-for-analyzing-physical-memory-dumps. (accessed: 00.06.2021).

[7] *Desktop OS Market share*. URL: https://www.statista.com/statistics/218089/global-market-share-of-windows-7/. (accessed: 00.06.2021).

[8] Fortinet. *Cybersecurity Statistics*. URL: https://www.fortinet.com/resources/cyberglossary/cybersecurity-statistics. (accessed: 00.06.2021).

[9] Pradeep Kumar Bhatia Gaurav Kumar. «Impact of Agile Methodology on Software Development Process». In: *International Journal of Computer Technology and Electronics Engineering (IJCTEE)* (2012). DOI: https://www.researchgate.net/profile/Gaurav-Kumar-175/publication/255707851_Impact_of_Agile_Methodology_on_Software_Development_Process/links/00b49520489442e12d000000/Impact-of-Agile-Methodology-on-Software-Development-Process.pdf.

[10] *Interesting Files Module*. URL: https://sleuthkit.org/autopsy/docs/user-docs/3.1/interesting_page.html#:~:text=The%5C%20Interesting%5C%20Files%5C%20module%5C%20allows,when%5C%20certain%5C%20things%5C%20are%5C%20found.. (accessed: 00.06.2021).

[11] J. Burić K. Hausknecht D. Foit. «RAM data significance in Digital Forensics». In: *MIPRO* (2015). DOI: https://ieeexplore.ieee.org/abstract/document/7160488.

[12] *memgator*. URL: http://www.orionforensics.com/forensics-tools/memgator/. (accessed: 00.06.2021).

[13] *Memoryze*. URL: https://www.fireeye.com/services/freeware/memoryze.html. (accessed: 00.06.2021).

[14] Mrs. H.K.Khanuja Mr. Vivek Ravindra Sali. «RAM Forensics: The Analysis and Extraction of Malicious processes from memory Image using GUI based Memory Forensic Toolkit». In: *2018 Fourth International Conference on Computing Communication Control and Automation(ICCUBEA)* (2018). DOI: https://ieeexplore.ieee.org/abstract/document/8697752.

[15] *PiL Library*. URL: https://web.archive.org/web/20150315041249/http://hg.effbot.org/pil-2009-raclette. (accessed: 00.06.2021).

[16] Atisha Wakdikar Pooja Salave. «Memory Forensics: Tools Comparison». In: *Communication, Integrated Networks & Signal Processing-CINSP 2018* (2018). DOI: https://www.ijsr.net/archive/v6i6/ART20173839.pdf.

[17] *rekall forensic*. URL: http://www.rekall-forensic.com/. (accessed: 00.06.2021).

[18] M. Ligh S. Adair B. Hartstein M. Richard. *Malware Analyst's Cookbook*. 2010.

[19] Marko Schuba Steffen Logen Hans Höfken. «Simplifying RAM Forensics». In: *2012 Seventh International Conference on Availability, Reliability and Security* (2012). DOI: https://ieeexplore.ieee.org/abstract/document/6329239.

[20] *VirusTotal*. URL: https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors. (accessed: 00.06.2021).

[21] *Volatility Wiki*. URL: https://github.com/volatilityfoundation/volatility/wiki. (accessed: 00.06.2021).

[22] *wxhexeditor*. URL: https://kalilinuxtutorials.com/wxhexeditor/#:~:text=wxHexEditor%5C%20is%5C%20a%5C%20Hex%5C%20Editor,low%5C%20level%5C%20disk%5C%20editor%5C%20too.&text=wxHexEditor%5C%20could%5C%20edit%5C%20HDD%5C%2FSDD,raw%5C%20up%5C%20to%5C%20exabyte%5C%20sizes.. (accessed: 00.06.2021).

APENDIX

A

APPENDIX A

RAMGUI

Proposta de Projeto

1st Alexandre Monteiro
MCIF
ESTG IPL
Leiria, Portugal
2190231@my.ipleiria.pt

2nd Leonardo Silva
MCIF
ESTG IPL
Leiria, Portugal
2190232@my.ipleiria.pt

I. Introdução

A análise de memória RAM é cada vez mais um componente essencial de uma análise forense, principalmente na análise de *malware*, uma vez que muitos destes ficheiros maliciosos guardam informações importantes para uma investigação.

No entanto é necessário *software* específico para este tipo de análise, para isso existe a *framework* Volatility, mas este *software* não tem uma interface gráfica o que pode causar diversos problemas para utilizadores menos experientes, tanto na sua utilização como instalação em diferentes plataformas.

Nesta proposta apresentamos alguns trabalhos relacionados em que já foram feitos alguns desenvolvimentos para este tipo de interface, também falamos de algumas formas como criar e gerir dependências para que seja uma aplicação *cross platform*.

Apresentamos também quais os nossos objectivos no desenvolvimento deste projeto, que tem como fundamental facilitar o uso da *framework* Volatility para este tipo de utilizadores mencionados anteriormente.

No final apresentamos a metodologia de como será desenvolvida a aplicação, e o tempo de desenvolvimento para certos etapas, dentro disto também será apresentado um pequeno esquema de como será a aplicação. Com todos estes objetivos pretendemos que esta aplicação venha facilitar o uso da ferramenta Volatility, isto destina-se principalmente a pessoas menos experientes que não tenham grande conhecimento da ferramenta, mas no entanto precisem de fazer a instalação e uso da mesma.

II. Trabalho Relacionado

Nesta secção poderemos ver alguns do trabalho desenvolvido nesta área e que nos iram auxiliar ao longo do desenvolvimento deste projeto.

A. Cross Platform Development Workflow for C/C++

Mesmo com estandardização destas linguagens de programação por parte da *American National Standards Institute* (ANSI) e *International Standards Organization* (ISO), para adicionar a isto também a disponibilidade das librarias da linguagem C e *standard template library* (STL) simplificou bastante o desenvolvimento de aplicações independentes de plataforma, para os sistemas operativos mais comuns.

Neste documento é apresentada uma ferramenta, CMake, que permite o programador usar a seu ambiente favorito de desenvolvimento em cada sistema operativo, no entanto poupa o tempo de sincronização de ficheiros específicos a cada plataforma, fornecendo assim apenas uma fonte de descrição textual.

Com o KDE4, o CMake foi introduzido num projeto bastante popular. Neste artigo é proposto um fluxo de trabalho para facilitar o desenvolvimento de projetos *cross platform*, é mostrado como foi usado o CMake para criar uma aplicação OpenGL, de forma a demonstrar uma aplicação com janelas a correr em Windows, Linux e Mac OS X.[1]

B. CMake

CMake é sistema extensível, *open-source*, que permite a gestão do processo de *build* de forma independente para sistemas operativos e compiladores. Este sistema é desenhado para trabalhar em conjunto com o ambiente nativo. Ficheiros de configuração simples colocados em cada directório fonte, são usados para gerar ficheiros de *build* estandardizados, que são usados de forma normal.

CMake consegue gerar uma *build* no ambiente nativo que vai compilar o código fonte, criar librarias, gerar *wrappers* e criar ficheiros executáveis em combinações arbitrárias. Permite o uso de *in-place* e *out-of-place builds*, portanto pode suportar diversas *builds* a partir de uma árvore fonte.[2]

Também suporta *builds* dinâmicas e estáticas. Outra função que o CMake têm é a geração de um ficheiro cache que é

desenhado para ser usado por um editor gráfico. Por exemplo quando este sistema corre, localiza ficheiros, bibliotecas e executáveis, e pode encontrar diretivas opcionais para *builds*. CMake é desenhado para suportar hierarquias complexas de diretórios e aplicações dependentes de diversas bibliotecas. Por exemplo, CMake suporta projetos que consistem em múltiplos *toolkits*, em que cada *toolkit* podem conter diversos diretórios, e a aplicação depende destes *toolkits* mais código adicional.

Este sistema permite também situações em que executáveis têm de ser construídos para gerar código que é então compilado, e ligado a uma aplicação final. Uma vez que CMake é open source, e tem um *design* simples e extensível, CMake pode ser estendido para suportar novas funcionalidades.

C. Volatility GUI

Este documento apresenta o desenvolvimento de uma GUI e respectivas extensões para a *framework* Volatility. Este problema nasce do facto de neste momento a versão oficial do Volatility ser apenas em formato linha de comandos o que poderá apesar uma grau de dificuldade de utilização para o utilizador menos experiente.

Este trabalho apresenta uma funcionalidade de uma forma fácil de guardar resultados em bases de dados e correr comandos a partir da base de dados, que seria necessário fazer independentemente da ferramenta do Volatility. Este trabalho apresenta as principais funcionalidade do Volatiity como a extracção de artefactos digitais de memoria RAM, a possibilidade de ver os processos corridos, as *networks* existentes, entre outros.

Também são apresentados vários requerimentos que devem existir para consideração no desenho da GUI, entre eles:

i) O licenciamento que deverá existir em baixo da GNU GPL
ii) O GUI deverá correr na mesma versão de sistema operativo do que o Volatility para propósitos de facilitar a instalação
iii) O GUI deverá ser simples para melhor a facilidade de utilização da ferramenta tanto para utilizar as suas ferramentas tal como para fazer a extracção de resultados
iv) A capacidade de carregar facilmente *dumps* de memoria através do menu da GUI
v) O acesso aos resultados guardados da extracção deverá ser fácil de consultar
vi) O suporte para todos os *plugins* existentes do Volatility
vii) A extensibilidade significando que deverá ser fácil de estender as funcionalidades desta GUI para analisar e correlacionar dados guardados nas bases de dados existentes no GUI.

Esta GUI foi desenvolvida na linguagem de Java e corre com um excutável *standalone* do Volatility sem a necessidade de instalação do Python. Posteriormente é apresentado o desenho e a funcionalidade da GUI, como o *Layout* da GUI, a apresentação de resultados, a interface com outras Classes e a utilização de *plugins*.

Posteriormente este trabalho apresenta as suas funcionalidades como, o carregamento das imagens RAM, as suas funções de analise e o carregamento e carregamento de resultados.

Este relatório termina por apresenta possíveis futuras melhorias ao GUI tal como o *dump* simultâneo de todos os processos em executáveis para poder correr estes processos em outras ferramentas e uma discussão final sobre como esta GUI poderá auxiliar os investigadores e mencionado que se trata de um trabalho em desenvolvimento.[3]

D. Volatility Survey

Este documento apresenta um estudo comparativo, da *framework* Volatility para análise de *malware*. São usados dez *plugins* que ajudam nesta análise.

Apresenta a eficiência destes *plugins* e técnicas usadas, para tal estes *plugins* vão ser usados em *malware* conhecido [4].

Os dez *plugins* a ser analisados são os seguintes: Malfind, Callbacks, Devicetree, Svcscan, Yarascan, Ldrmodules, Apihooks, Psxview, Threads, e Timers são ferramentas efetivas na análise de *malware* na memória RAM.

Com a crescente frequência de ataques de *malware* baseados em memória, este tipo de ataques conseguem infetar uma máquina sem que seja feito o *download* de qualquer ficheiro, o que torna este tipo de análise de *malware* bastante importante.

O crescimento deste tipo de ataques afeta cada vez mais áreas de negócio, e os *malwares* são cada vez mais sofisticados e perigosos. Ataques através de *malware* têm uma grande diversidade de alvos, pode ir de um indivíduo a uma empresa de segurança.

A investigação em código malicioso, foca-se principalmente em quatro áreas que são: comportamento, código, memória, e análise de inteligência. Cada um destes setores tem ferramentas especializadas, que ajudam os investigadores na análise deste código malicioso.

A análise de memória em particular pode ser complicada, e ferramentas específicas para a investigação são essenciais, mas não existe uma grande variedade. Uma ferramenta que é bastante usada para estas análise é a *framework* Volatility.

Neste documento é feita uma análise aos diversos *plugins*, apresentados anteriormente, de forma a analisar a sua eficácia e técnicas de utilização. Também é feita uma comparação com resultados encontrados noutros estudos da mesma área.

Para isto é usado o *malware* Stuxnet onde são feitos diversos testes com os *plugins*, e posteriormente é feita uma análise dos resultados obtidos com estes testes.

Com isto podemos avaliar o desempenho desta ferramenta na análise *malware* na memória RAM.

E. Objetivos

Apesar de já existir uma GUI para o Volatility desenvolvida em 2012 a partir de um projeto independente, esta GUI apresenta semelhante-mente ao Volatility uma curva de aprendizagem algo elevada.

O nosso projeto pretende inspirar-se em tudo o que já foi desenvolvido até à data e melhorar nos aspectos de facilidade de instalação e de utilização. Pretende-se que esta GUI funcione obedecendo a todos os aspectos legais de licenciamento da GNU GPL, pretende-se que seja *cross-plataform* para deixar de existir a limitação de sistema operativo, deverá ser facilmente instalado semelhante a um software como Notepad++ sem a necessidade de instalar outras dependências.

O GUI deverá fazer tudo o que o Volatility tem capacidade de fazer, e ser capaz de correr todos os *plugins* e extensões do mesmo, também deverá ser capaz de guardar resultados e e reabrir os mesmos para interpretação, para facilidade de utilização do *end-user*, deverá existir uma interface gráfica onde o utilizador poderá obter informação sem a necessidade de qualquer conhecimento prévio do Volatility e por último também para facilitar a utilização deverá existir uma opção que ao carregar a imagem da RAM será capaz de correr todos os comandos básicos e apresentar a informação mais crucial.

Estes objectivos serão apenas os objetivos principais e uma ideia do projeto sendo que poderá sofrer alterações ao longo do desenvolvimento do mesmo.

III. Metodologia

Nesta secção iremos tentar demonstrar uma ideia do projeto que iremos tentar desenvolver entre calendarização e algumas das ideias a representar no trabalho.

A. Tecnologia

Apesar de ainda não estar definido, a GUi será desenvolvida em Java ou C/C++ sendo estas as melhores tecnologias para Cross-Plataform. será utilizado o Volatility devido à GUI encontrar-se desenvolvida á volta desta ferramenta e por consequência a tecnologia Python devido à dependência do Volatility nesta tecnologia.

B. Plano Cronológico

Na seguinte tabela poderemos ver uma ideia do tempo necessário para o desenvolvimento de cada componente do trabalho.

Tabela I: Linha temporal relativa ao projeto a desenvolver

	Sept	Oct	Nov	Dec	Jan	Fev	Mar	Abr	May	June	July
Planeamento											
Desenho											
User Stories											
Desenvolvimento											
Testes de Utilizador											
Relatório											

C. Entregáveis

Espera-se que as seguintes entregas sejam produzidas após a conclusão bem-sucedida do projeto:

- Prova de conceito relativa aos princípios subjacentes à teoria por detrás desta GUI
- Aplicação funcional com todas as funcionalidades que serão definidas posteriormente
- Tese - Relatório Final

D. Detalhes

Tal como já mencionado anteriormente estes trabalho será dividido por dois elementos do grupo que por sua vez encontra-se dividido em duas partes. Sendo que a primeira parte do trabalho será o planeamento do trabalho em si tal como as tecnologias a utilizar, o seu desenho e arquitectura, as suas *user stories*, e os testes a realizar.

Sendo que este projeto irá ser começado do zero será necessário ao longo do seu desenvolvimento decidir a ideologia final da GUI tal como as suas funcionalidades. Porém algumas das funcionalidades que tencionamos ter na aplicação final. São uma instalação simples e que funcione em qualquer plataforma, a sua distribuição será decidida posteriormente. A interface da aplicação deverá ter um tipo de interface gráfica que permite a um utilizador usar algumas das funcionalidades do Volatility através da interacção com caixas relativas a processos da imagem de RAM presente, sendo esta uma das funcionalidades iniciais que deverá estar presente na aplicação. Outra funcionalidade deverá ser a execução automática de alguns comandos básico durante a inserção da imagem de RAM e a posterior apresentação de um relatório que dirá algumas das informações mais importantes. Sendo estas algumas das funcionalidades principais que deverão ser implementadas na aplicação.

Relativamente aos testes para além dos testes normais de utilizador que pretendemos fazer, também pretendemos efectuar testes com vários perfis de utilizadores e ver a eficácia desta

ferramenta em relação à *framework* Volatility puro. Iremos expandir sobre esta temática ao longo da próxima secção.

IV. Avaliação

A avaliação deste trabalho será baseada em 3 eixos para avaliar a ferramenta em si.

A. Eficiência

Aqui iremos tratar de verificar se a GUI desenvolvida será capaz de fazer todas as funcionalidades que a *framework* Volatility possui neste momento. E também se a GUI é capaz de trazer funcionalidades extras tal como a criação de relatórios, capacidades de guardar dados e restaurar estes entre outras funcionalidades que serão previamente discutidas.

B. Eficácia

Neste eixo iremos fazer testes com utilizadores. Aqui pretendemos usar vários perfis de utilizadores com vários níveis de experiência. Entre eles utilizadores sem qualquer conhecimento do Volatility, utilizadores com alguma experiência e utilizadores bastante familiarizados com a ferramenta. O teste que pretendemos fazer neste momento trata-se da comparação do tempo da utilização da *framework* Volatility, para o processo de instalação da ferramenta já existente e da GUI desenvolvida. Dentro deste teste também pretendemos efectuar uma simples tarefa relativa a um processo e novamente medir o tempo que cada utilizador demorou, a efectuar em cada respectivo *software*. Com este teste iremos conseguir medir a eficácia de cada componente que será desenvolvido comparando a nossa ferramenta com a *framework* original.

C. Utilidade

Neste último eixo será efectuado um questionário aos utilizadores testados para verificar a interacção entre a GUI desenvolvida e o utilizador final. Medindo assim a preferência de utilização deste utilizadores em relação ás ferramentas testadas.

V. Resultados Esperados

Prevê-se que com o desenvolvimento deste projeto para além da ferramenta Volatility fique disponível a mais utilizadores, pretende-se também que a área de estudo da análise de memoria RAM se torne mais fácil de começar.

O principal foco desta aplicação será a introdução de novos perfis de utilizador para esta área, e auxiliar os utilizadores que trabalham com esta ferramenta tornando o seu trabalho mais rápido, e sem a necessidade da utilização de ferramentas externas, contendo assim todo o software necessário para uma análise de memória RAM dentro da nossa aplicação.

Referências

[1] M. Wojtczyk e A. Knoll. ≪A Cross Platform Development Workflow for C/C++ Applications≫. Em: (2008).

[2] *CMake*. Jun. de 2020. URL: https://cmake.org/overview/.

[3] Marko Schuba Steffen Logen Hans Höfken. ≪Simplifying RAM Forensics: A GUI and Extensions for the Volatility Framework≫. Em: *https://ieeexplore.ieee.org/document/6329239* (ago. de 2012).

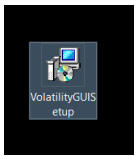
[4] Carl Henry. ≪Survey and Use of Ten Volatility Framework Plugins for Malware Analysis≫. Em: (2017). URL: https : / / search . proquest . com / openview / da58c300f6c0da13f57acfbc5325b0d3 / 1 ? pq - origsite = gscholar%5C&cbl=18750%5C&diss=y.

B

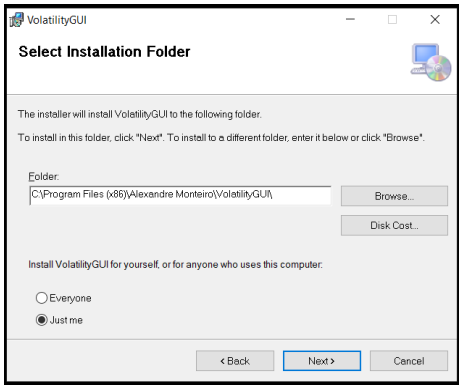
APPENDIX B

VolatilityGUI Tutorial

1. Executar o ficheiro de setup.msi, anexado a este documento e instalar o software VolatilityGui (VoGUI).

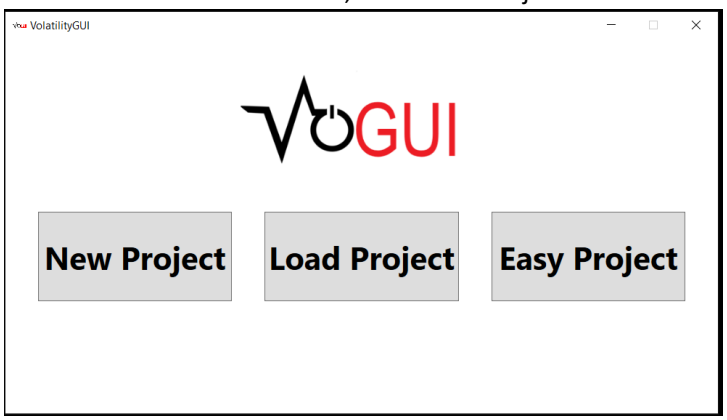


2. Continuar com a instalação do software com as preferências pretendidas.



3. Depois de instalado deverá aparecer um atalho no Ambiente de Trabalho para abrir o software.

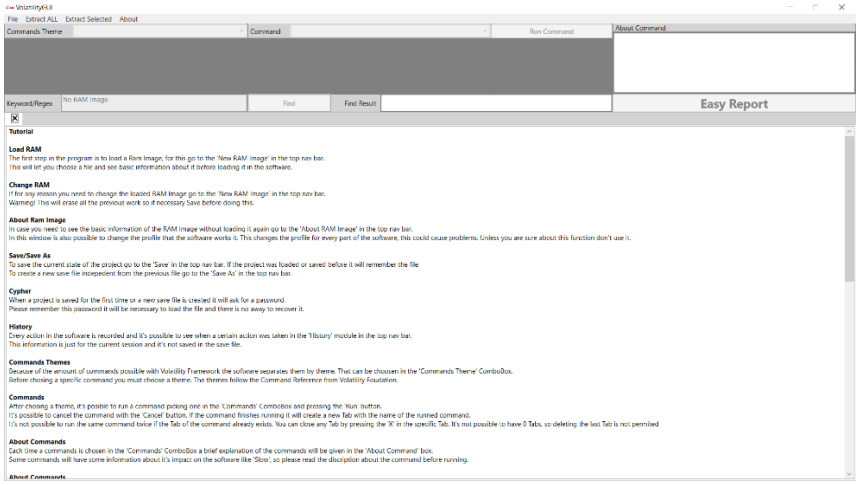
4. Ao clicar duas vezes no atalho, deverá abrir a janela do software VoGUI.



5. Caso isso não aconteça, clique no botão direito do rato em cima do atalho e escolha “Executar como administrador”. Ou ir a C:\Program Files (x86)\Alexandre Monteiro\VolatilityGUI e escolher o ficheiro VolatilityGUI.exe e clique no botão direito do rato em cima do atalho e escolha “Executar como administrador”.

Novo Projeto

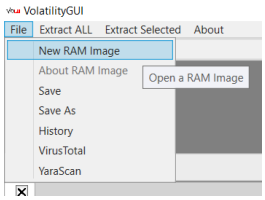
6. Para começar um novo projeto, clicar no botão **New Project**.
7. Ao clicar neste botão deverá abrir a seguinte janela.



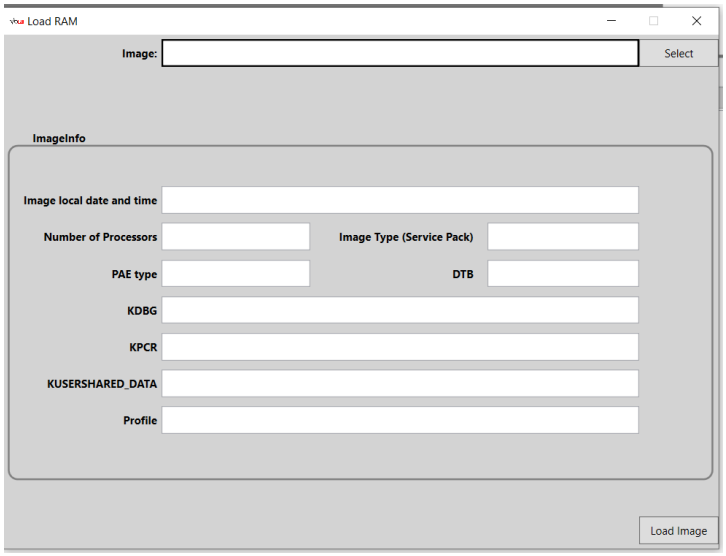
8. Nesta janela principal aparece um pequeno tutorial de todas as funcionalidades do software.

Carregar Imagem de Memória RAM

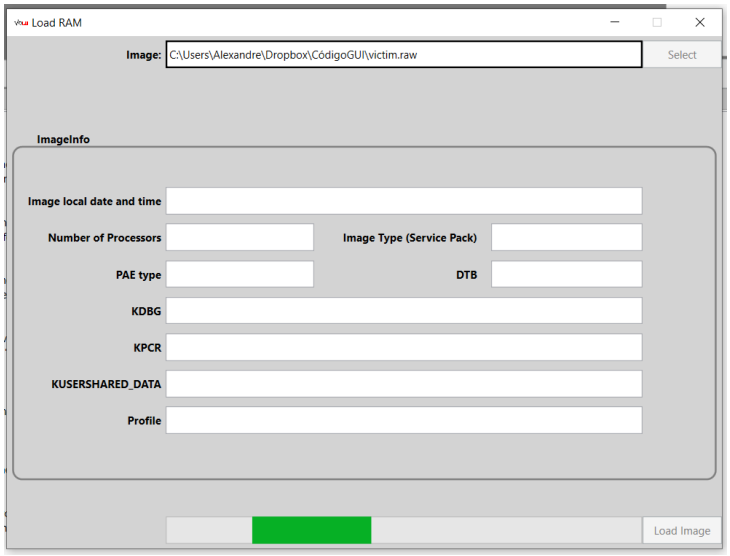
9. Para carregar uma imagem de memória RAM, basta clicar no menu **File > New RAM Image**, como mostra a seguinte imagem.



10. Depois de clicar deverá aparecer esta nova janela.



11. Para selecionar a imagem de memória pretendida basta clicar no botão **Select** o qual irá abrir o explorador do Windows onde será possível carregar todos os formatos suportados pelo Volatility.
12. Depois de selecionada a imagem de memória, o processo irá começar, podemos ver que o processo está a correr pela barra de processo a correr no fundo da janela como mostra a seguinte imagem.



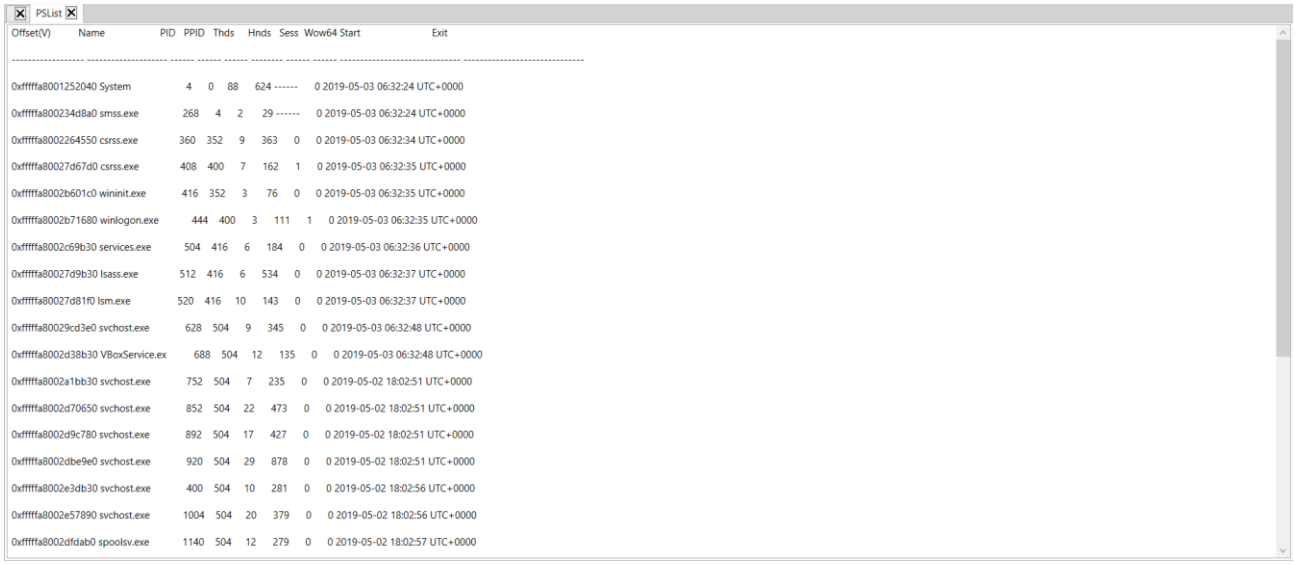
13. Depois do processo de importação da imagem de memória ter finalizado é possível ver algumas informações relativas à imagem de memória carregada. O campo Profile é automaticamente escolhido para facilitar a utilização do software, mas este campo poderá ser alterado mais tarde. Para carregar esta imagem de memória no software basta clicar no botao **Load Image** que ficará disponível quando o processo de importação terminar.
14. Ao clicar neste botão, esta janela é fechada e é apresentada a janela principal onde já deverá ser possível correr comandos do Volatility.

Categorias de Comandos

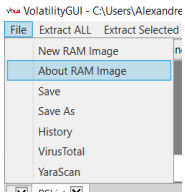
15. Devido ao número elevado de comandos do Volatility estes estão divididos por categorias. Exemplo: Para executar o comando **PSList** sobre a imagem de memória carregada basta escolher dentro da caixa **Commands Theme** o tema **Processes and DLL**. Na caixa seguinte com o nome **Commands** escolha o comando **PSList**. Esta janela deverá ter a seguinte apresentação.

16. Para executar o comando clique no botão **Run Command.** Depois de clicar, aparece uma barra de progresso diretamente abaixo deste botão onde é possível cancelar o comando carregando no botão **Cancel**.

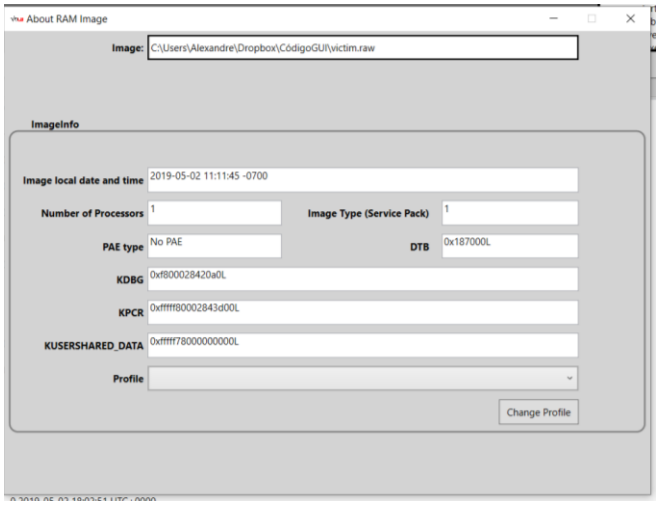
17. Quando o processo relativo ao comando termina, é criada uma nova **Tab** na parte inferior da janela principal com o nome do cabeçalho igual ao do comando executado. Tal como todas as outras *tabs*, esta poderá ser removida carregando no '**X**' da *tab*. O software exige sempre a presença de uma *tab* ativa, no mínimo.



18. Apesar de não ser possível executar o comando **imageinfo** (informações sobre a imagem de memória carregada) da mesma forma que todos os outros comandos, o resultado dessa informação está disponível no menu **File->About RAM Image**.



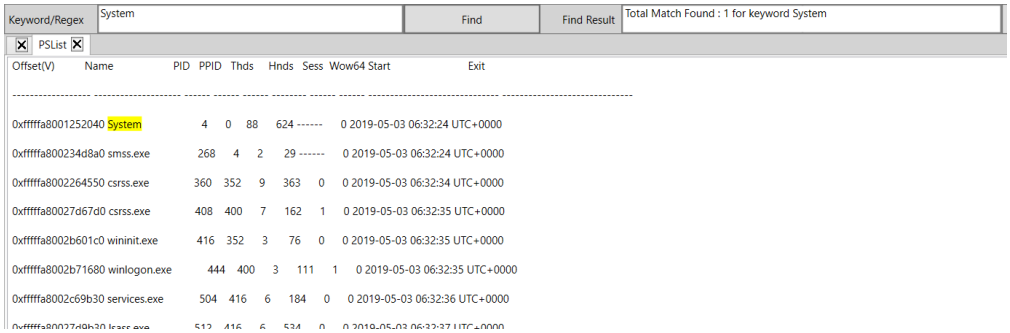
19. É também nesta janela, que é possível alterar a **Profile** da imagem de memória carregada, escolhendo a nova profile na caixa **Profile** e clicando no botão **Change Profile**.



Procurar

20. Na janela principal é possível utilizar a caixa da última fila para encontrar *keywords* ou utilizar regras de regex para encontrar informação na *tab* selecionada/visível. Como exemplo, é possível procurar pela palavra "System" inserindo a palavra na caixa de texto à direita da label **Keyword/Regex** e posteriormente clicando no botão **Find** para começar o processo.

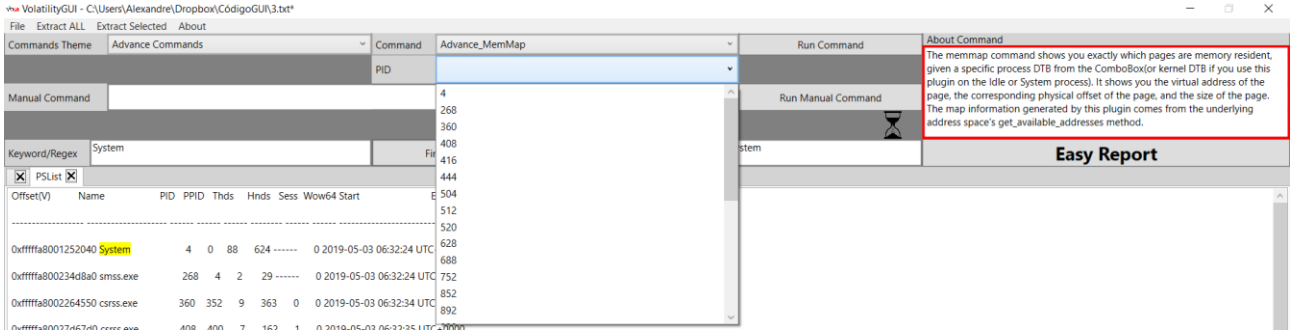
21. O resultado irá ser demonstrado na *tab* selecionada através de um sublinhado amarelo. O número de ocasiões em que o processo encontrou um resultado será apresentado na caixa de texto ao lado da label **Find Results**.



Comandos Avançados (lentos ou com parâmetros adicionais)

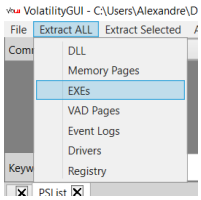
22. Na caixa dos **Commands Categorie** existe uma opção de tema: **Advance Commands**. Estes comandos são normalmente lentos ou precisam de uma segunda variável. A escolha desta categoria também dá a opção de executar comandos manuais caso seja necessário. Para isso deve ser escrito o comando manualmente começando pelo plugin. Não é necessário escrever qual é a imagem de memória e o *profile* a ser utilizados no comando manual. Isso é realizado automaticamente pelo VoGUI.

23. Na caixa **Commands** existe o comando **Advance_MemMap.** A execução deste comando irá preencher uma nova caixa inferior com outras opções que servirão de segunda variável ao comando principal. Isto tem como objetivo facilitar a escolha e eliminar erros de comandos manuais. Para executar o comando com a segunda variável, basta selecionar um valor da caixa inferior secundária, e clicar no botão **Run Command**.

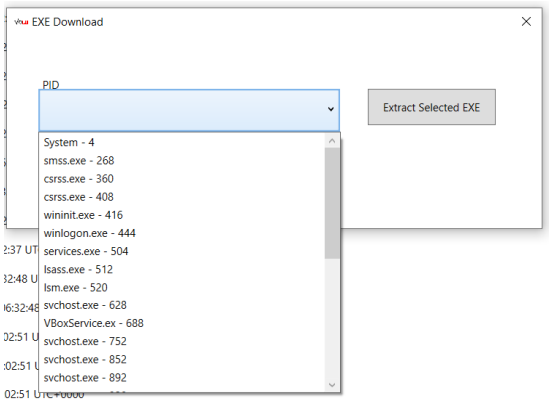


Menu Extract ALL e Extract Selected

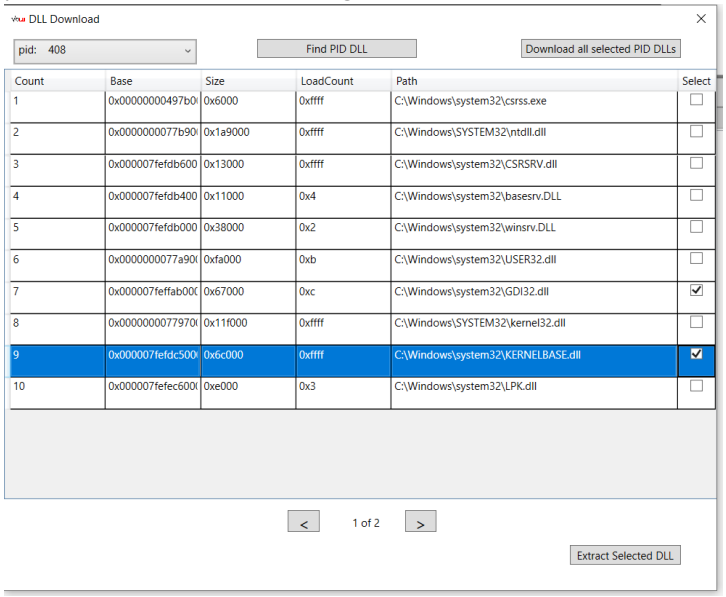
24. No menu existem dois tipos de extração, **Extract All** e **Extract Selected**. Ao clicar em **Extract All** é possível ver todos os tipos de ficheiros que podem ser extraídos da imagem de memória carregada. Ao clicar numa destas opções irá ser aberto o explorador do Windows para ser escolhido um diretório para o qual serão gravados/exportados os ficheiros.



25. Ao clicar na opção **Extract Selected** irão aparecer várias opções. Clicando no **EXEs** (Menu->Extract Selected->EXEs) irá ser aberta uma nova janela, na qual será possível escolher o processo do qual se pretende extrair o tipo de ficheiro escolhido. Para começar a extração basta selecionar o processo e de seguida clicar no botão **Extract Selected EXEs**.

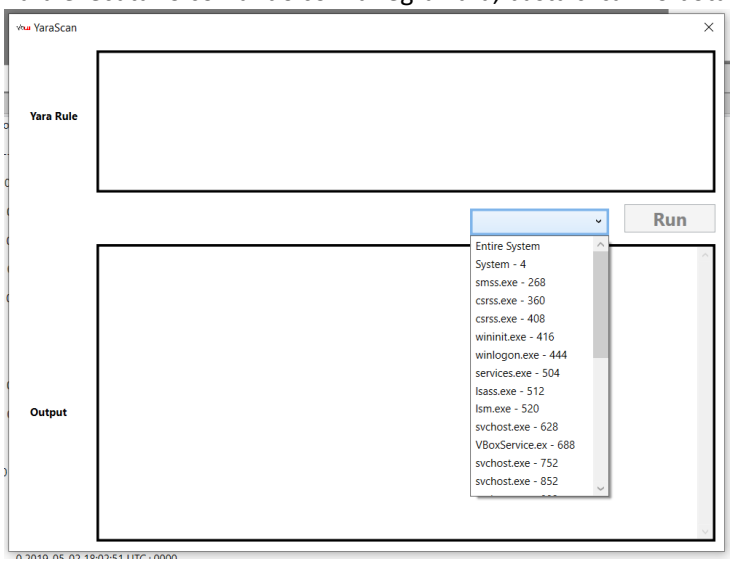


26. No menu **Extract Selected** é possível verificar que os **DLLs** funcionam de uma forma um pouco diferente. Selecionando esta opção, é aberta uma nova janela na qual é possível o utilizador escolher um processo (PID), o qual, irá preencher automaticamente a lista inferior com os DLLs associados a este processo (botão **Find PID DLL**) e apresentar informações detalhadas de cada DLL. Caso o utilizador pretenda extrair apenas alguns processos, pode selecioná-los e descarregá-los clicando no botão **Extract Selected DLL**.



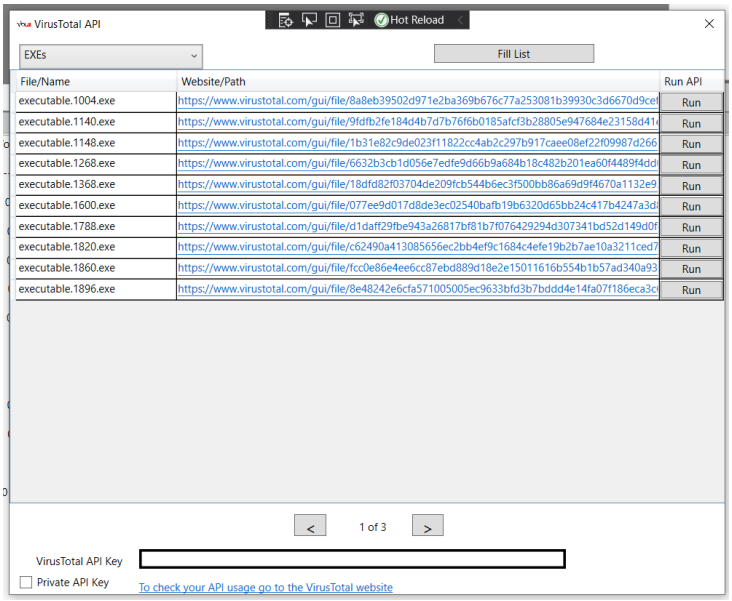
Yara Rules

27. Na janela principal, existe a opção **YaraScan** no menu (File->YaraScan). Ao clicar nesta opção é possível escrever a regra Yara pretendida e ver o seu resultado dentro do software VoGUI. Também é possível executar a regra Yara sobre todo o sistema ou sobre um processo específico selecionando este na caixa de seleção no centro do ecrã. Para executar o comando com a regra Yara, basta clicar no botão **RUN**.

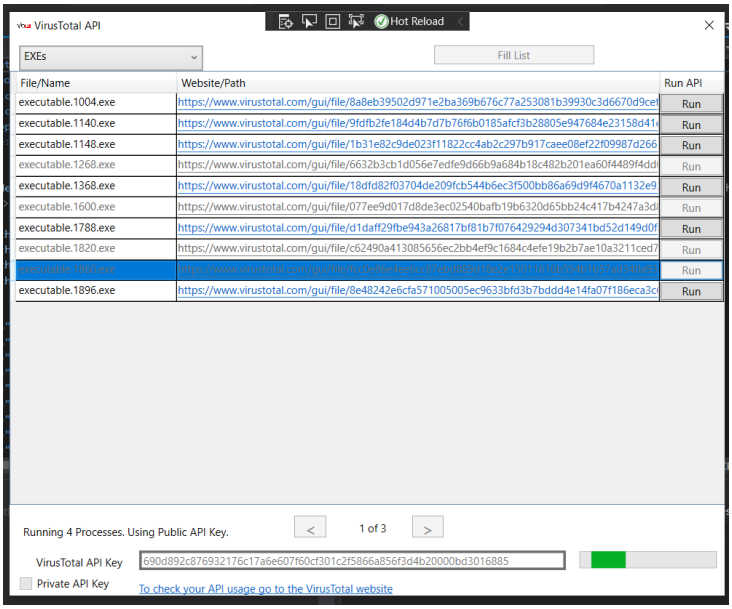


Integração com portal VirusTotal

28. Este software também conta com uma integração externa do portal web VirusTotal. Ao clicar na opção **VirusTotal API** (File->VirusTotal API), irá ser aberta uma nova janela, na qual é possível preencher a lista com todos os **EXEs** ou **DLLs** do sistema ou com os **DLLs** de um processo específico.
29. Exemplo: Ao selecionar a opção **EXEs** na caixa de seleção e clicando o botão **Fill List** o processo irá executar e preencher automaticamente os caminhos para o portal VirusTotal. Quando terminar a janela deverá ter um aspeto semelhante ao da seguinte imagem.

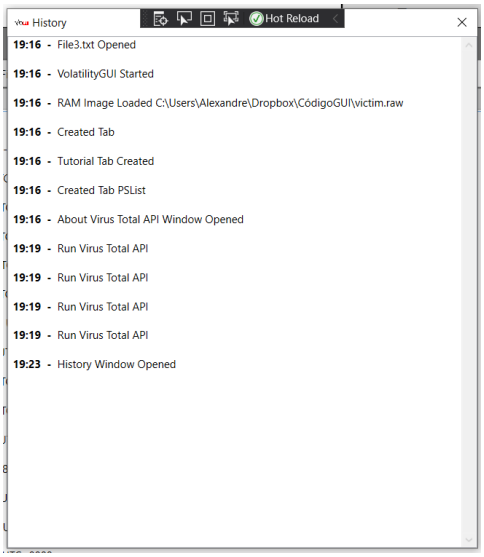


30. Cada ficheiro fica com link atribuído no qual é possível clicar para abrir no *browser* predefinido do utilizador o relatório sobre este ficheiro no portal VirusTotal, caso este ficheiro já tenha sido identificado previamente. Caso contrário é possível inserir uma API Key do VirusTotal (disponível com todas as contas criadas no VirusTotal) na caixa de texto com o nome: **VirusTotal API Key**. Caso esta *key* seja privada, deve ser selecionada a *checkbox* **Private API Key** para não ter limitações a nível de API. Com esta *key* configurada no software, é possível clicar no botão RUN de cada ficheiro para realizar um pedido de relatório à API do VirusTotal. O relatório ficará disponível no portal do VirusTotal após 30 segundos. Uma chave publica tem a limitações a nível de 4 pedidos por minuto que o software controla e cerca de 500 pedidos por dia que o software apenas controla se não for encerrado.



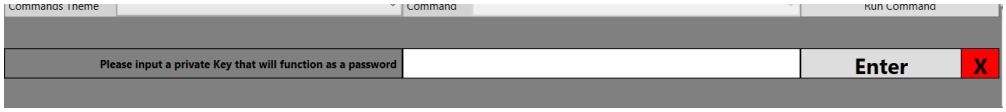
Histórico

31. Ao clicar na opção **History** no menu **File** (File->History) irá ser aberta uma nova janela que apresentará um histórico de todas as ações executadas, ordenadas temporalmente durante a sessão de trabalho.



Gravar a sessão de trabalho do VoGUI

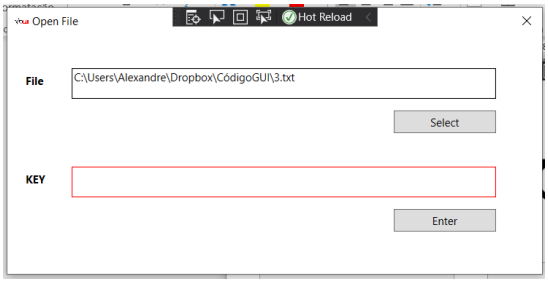
32. Para gravar a sessão atual, basta selecionar uma das opções: **Save** (File->Save) ou **Save As** (File->Save As). Se a sessão atual ainda não tiver sido gravada, nesse caso é indiferente qual das duas opções. Para este caso iremos pressionar a opção **Save** que irá mostrar mais alguns componentes do software como podemos ver na seguinte imagem.



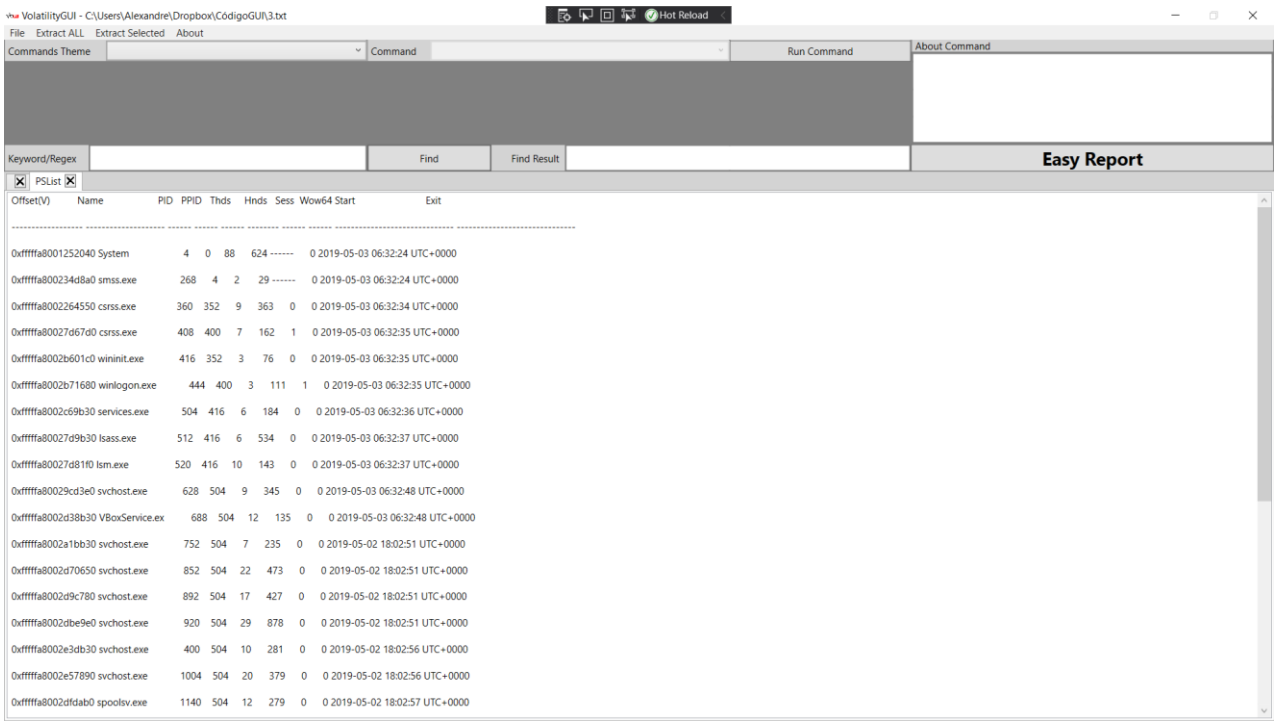
33. Para gravar o ficheiro será necessária inserir uma chave de cifragem e clicar no botão **Enter** para escolher um locar onde gravar o ficheiro da sessão de trabalho. Após realizar com sucesso a primeira gravação para ficheiro, é possível realizar as gravações subsequentes, apenas clicando na opção **Save**.

Abertura de sessões de trabalho anteriores

34. Ao voltar a abrir o software é possível realizar o carregamento da sessão de trabalho anterior guardada em ficheiro. Carregando no botão **Load Project** irá abrir uma nova janela onde será necessário escolher o ficheiro que acabamos de gravar e a sua chave, pressionando no botão **Select** para escolher o ficheiro e **Enter** para carregar o ficheiro, caso a chave não seja a correspondente a este ficheiro irá mostrar uma mensagem de erro.



35. Se este último passo for feito com sucesso todo o trabalho previamente feito irá ser apresentado com algumas exceções (ex. listagem de PIDs em comandos avançados) de elementos mais voláteis de fácil acesso.

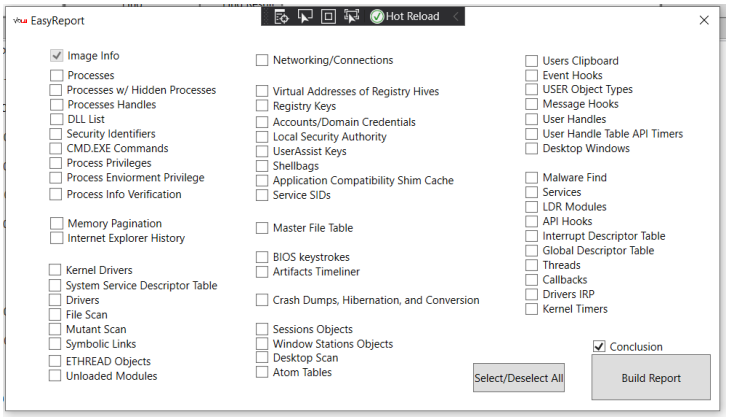


Segurança dos ficheiros de projeto

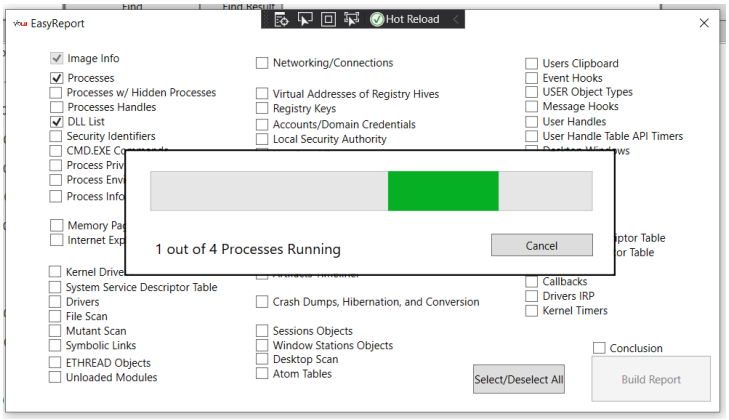
36. Ao fechar o software e consultar o ficheiro guardado é possível verificar que este encontra-se cifrado.

Easy Report

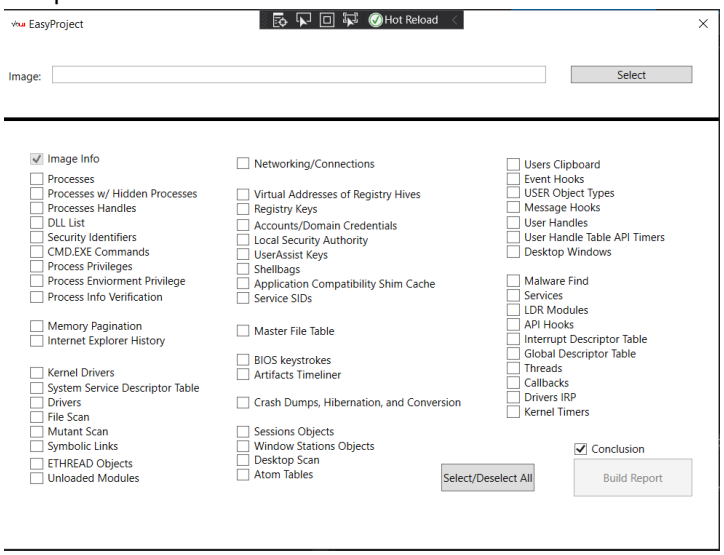
37. Voltando com o trabalho que foi feito anteriormente ao clicar no botão **Easy Report**, este irá abrir uma nova janela com dezenas de opções para contruir um relatório. Cada uma destas opções contém *tooltips* com a informação do resultado dessas escolhas.



38. Ao selecionar as opções **Processes**, **DLL List** e **Malware Find**, também é possível desselecionar a opção **Conclusion** pois esta demora bastante tempo a ser construída. Depois de ter estas opções selecionadas, clicar no botão **Build Report**. Este processo irá começar a executar passo a passo todos os processos, podendo ser cancelado a qualquer momento.



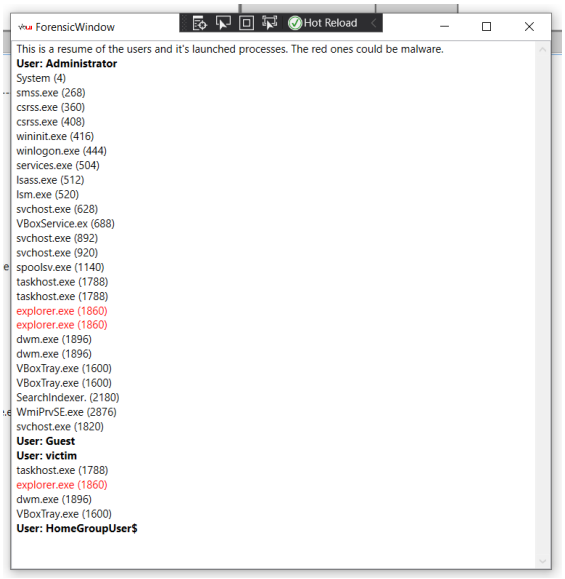
39. Este processo ao terminar irá automaticamente abrir uma página web com as informações escolhidas no relatório com a opção de chegar rapidamente ao tópico pretendido através da NavBar. Ao clicar no tópico **Malware Find** (Malware->Malware Find) a página irá deslocar-se automaticamente para este tópico.
40. Ao abrir uma nova instância do software a opção **Easy Project** também é disponibilizada através de um botão. Este botão abre uma janela bastante semelhante à anterior, com a diferença de poder carregar a imagem da memoria RAM diretamente nesta janela podendo assim obter toda a informação necessária sem muita complexidade.



Forensic Investigation

41. Por último dentro do menu File é possível encontrar a componente **Forensic Investigation** (File-> Forensic Investigation). Ao clicar nesta opção, será aberta uma nova janela, bastante experimental que tenta relacionar os

utilizadores presentes no sistema com os processos executados, possíveis conexões e a presença de *malware*. Esta função é experimental e poderá não funcionar a 100%.



Obrigado pela conclusão deste pequeno tutorial do software VolatilityGUI, semelhantemente ao próprio Volatility a única limitação deste software é o que o utilizador pretenda fazer com ele. Agradeço o tempo dispensado e gostaria de pedir mais 5mins para preencher o seguinte questionário: https://forms.gle/eaAhoL8StcC5xRFs8

Qualquer funcionalidade extra ou em falta poderá ser pedida na última parte do questionário. Qualquer input é bem-vindo.

Obrigado, Alexandre Monteiro 2190231@my.ipleiria.pt

C

APPENDIX C

Volatility v.s. VolatilityGUI – Timed Test

Note: The RAM image will be provided, which is called ‘victim.raw’. The setup for VolGUI and the Volatility Standalone will also be provided. The maximum time for each task is 5 minutes. Each task should be replicated with both software’s start always with Volatility. This means that for each task 2 tests should be done. Basic questions can be asked. The Volatility framework website will be opened from the start of the test. The time starts after the question is read.

1. With the provided image find the "Image local date and time". (Test the basic usage of each software)

*After this question 3 minutes will be provided to know each software

2. With any profile on the provided image find the list of processes of the system.
3. With a profile different from the second task on the provided image find the system loaded DLLs.
4. On the results from the third task find the string "SSCORE.DLL".
5. On the provided image, extract just the DLL file "SSCORE.DLL".
6. Using the website VirusTotal find if the EXE file from the process 'csrss.exe', from the provided image, contains any malware.
7. Using Yara Rules, on the provided image, write the rule "string" to run on the process 'csrss.exe'.
8. On the provided image run the command 'hashdump'.
9. On the provided image run the command 'hivedump' on the offset for Security.
10. Find what command is responsible for "helping to find hidden or injected code/DLLs in user mode memory".

*Extra question 10 minutes to resolve.

11. Find, on the provided image, what processes the user 'victim' is responsible for launching and if any of this software possibly contains malware.

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Interface gráfica para a ferramenta forense de análise de RAM"*, é original e foi realizado por Student Alexandre de Sousa Monteiro (2190231) sob orientação de Professor Doutor Luís Alexandre Lopes Frazão (luis.frazao@ipleiria.pt).

Assinado por: **ALEXANDRE DE SOUSA MONTEIRO**
Num. de Identificação: 14859564
Data: 2022.01.10 16:50:05+00'00'

Leiria, November 2021

Student Alexandre de Sousa Monteiro