Pagefile.sys Forensic Analysis

Dinis Cruz
DETI
Universidade de Aveiro
Aveiro, Portugal
cruzdinis@ua.pt

Lucas Sousa
DETI
Universidade de Aveiro
Aveiro, Portugal
joselmdbsousa@ua.pt

Tiago Oliveira
DETI
Universidade de Aveiro
Aveiro, Portugal
tiago.srv.oliveira@ua.pt

***Abstract*—This document details the usage of Belkasoft Evidence Center and the usage/development of proprietary scripts in order to extract forensic artifacts from the Windows pagefile.sys. The recovered artifacts contain browser searches, images, database tables, system hardware information, partial file system reconstruction, requests, emails, usernames and passwords. It was also discussed the value of these artifacts, contextually, in a real life forensic investigation.**

Index Terms*—*pagefile.sys*, *RAM*, *Ransomware*, *Virtual Machine (VM)*, *FTK Imager*, *Belkasoft Evidence Center (BEC)*, *Regex

I. Introduction

This document details the extraction of significant information for forensic purposes using the pagefile.sys file. It was created for the first year of MSc Cybersecurity at the University of Aveiro's Forensic Analysis of Computational Systems course taught by Professor Artur Varanda.

RAM analysis is an important part of computer forensics as it helps the investigators in finding out what actions (criminal or non-criminal) that were taken on a computer. The pagefile.sys is more volatile than RAM, considering it's just random memory pages stored in the disk, but it can yield valuable forensic artifacts as will be shown in this document.

A test case was created from a Windows 7 Virtual Machine with minimal RAM (2 GB) to force the usage of page memory and highlight the relevance of what information is recorded in the pagefile.sys. To extract the data, FTK Imager Version 4.5 was utilized. FTK Imager is an open-source software by AccessData designed to generate accurate copies of original evidence without changing it; in this case, it was used to create a memory dump of the Virtual Machine's pagefile.sys.

II. Objectives

In this report the team will explore how to do a forensic analysis on a page file. Through a memory dump, a lot of information can be recovered from valuable artifacts. Besides that, virtual memory analysis allows you to perform memory forensics without a memory dump.

III. Contextualization

A. Pagefile.sys

There are a few records on a filesystem that contain memory data: pagefile.sys, swapfile.sys, and hiberfile.sys. This document further explores the topics of the pagefile.sys.

The primal objectives of this report are the documentation of the pagefile extraction and analysis. Some objectives are: locating browser searches, downloaded images and software, database information, the filesytem structure, emails and user contacts, and eventually machine requests.

The pagefile controls how Windows NT uses demand paging to manage virtual memory. Demand paging is a method through which the operating system can substitute hard disk space for actual RAM. The Memory Management Unit (MMU) on the CPU is responsible for this. Memory is divided into page frames, which are little pieces of information. In this context, "memory" refers to the space in pagefile.sys that exists both in RAM and on the hard disk [4].

RAM fills up as code and data from the hard disk are loaded into it. When Windows NT determines that RAM is at full-capacity and that more RAM is required to complete tasks, the operating system examines the page frames in RAM and determines which of these page frames have not been used recently. When Windows NT has gathered all of the least recently utilized page frames in RAM, it saves them to the hard disk in the pagefile.sys file [4]. This way, Windows NT frees up the RAM that these frames were using. The vacant space is then used by Windows NT to load code and data that could not previously be loaded due to a lack of RAM.

B. Strings command

As previously explained, pagefile.sys is a binary file that contains several data from the computer RAM about the user: files data, browser searches, etc. Being a binary file, its content is not readable by a human, thus the usage of the string command is fundamental to get out the readable data (strings).

When the strings command is run on pagefile.sys, the output becomes the content which will be visually analyzed. This way, the data can be parsed and analyzed with our proprietary scripts.

IV. Analysis

A. Belkasoft Evidence Center

Belkasoft Evidence Centre is used for analysis of the pagefile.sys to study the carved file. Belkasoft Evidence Centre is a forensic tool that collects, analyzes, and extracts digital evidence.

Both PC and mobile forensics are supported by the Belkasoft Evidence Center (BEC). The same BEC program can help

you comprehend what information is held within whether you have a laptop, an iPhone, a desktop computer, or an Android tablet. BEC aids in the acquisition and analysis of computers, mobile devices, clouds, and RAM.

The following subsections will report the relevant information that was found using this tool [5].

1) Browser Search: Once the processing pagefile.sys is in the BEC there is a section about browser information that has information about searches in every browser available in the machine. the available browsers, in this case, are: Chrome, Internet Explorer, Opera, and Firefox. The browser with more information was google chrome (as it is usually the most used nowadays) and the browser with less information was opera.

Google Chrome:

- Login in reddit
- Login in amazon
- Navigated to a suspicious website, url: http://www.qqhudong.cn/usersetup.asp?action=
- Login in twitter
- Login in google
- Login in youtube
- Login in github
- Navigated to a weapon online store, url: https://www.bestbuy.com/
- Navigated to US army website, url: https://www.goarmy.com/
- Login in OLX
- Login in University of Aveiro
- Login in caixa geral de depósitos
- Login in netflix
- Search on how to perform cyber attacks, url: https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html

Firefox:

- Search on how to use ransomware tool “wanna cry”
- Searched about how does ransomware tool “wanna cry” works
- Searched for ftkimager
- Searched for “wanna cry” ransomware tool download
- Downloaded ”wanna cry” ransomware tool
- Search on how to download browser tor

Internet Explorer:

- Login with username ”Cruzd”
- Login in msn
- Login in linkedin
- Search on how to infect a computer with worm virus, url:https://www.bing.com/search?q=worm virus how to infect&FORM=IE8SRC&pc=EUPP_

Opera:

- Search on how to infect a computer with worm virus, url:https://www.bing.com/search?q=worm virus how to infect&FORM=IE8SRC&pc=EUPP_

2) Timeline: The BEC tool has a timeline section which details a timeline of the user’s online presence:

- Login in browser with username ”Cruzd”
- Searched for cyber-attacks, url: https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html
- Searched about how to infect with worm virus, url:https://www.bing.com/search?q=worm virus how to infect&FORM=IE8SRC&pc=EUPP_
- Searched for “wanna cry” which is a ransomware toll, url: https://www.google.com/search?q=Wanna
- Searched about how does ransomware tool “wanna cry” works
- Searched for “wanna cry” ransomware tool download
- Downloaded ”wanna cry” ransomware tool
- Searched how to use ransomware tool “wanna cry”
- Downloaded TOR browser

3) Images: In the images section there were a lot of social-media logos and browser images which is what is loaded when someone is navigating on the internet, however, there were also a lot of suspicious images. Ransomware tool images and cyber-attack banner images were found, that suggests the user navigated through pages about these topics. There are also TOR browser images that suggest that this browser was used but not detected as a browser by BEC.

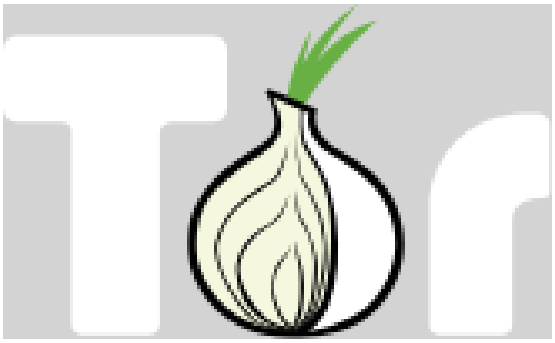

Fig. 1. Tor browser logo

Fig. 2. WannaCry ransomware

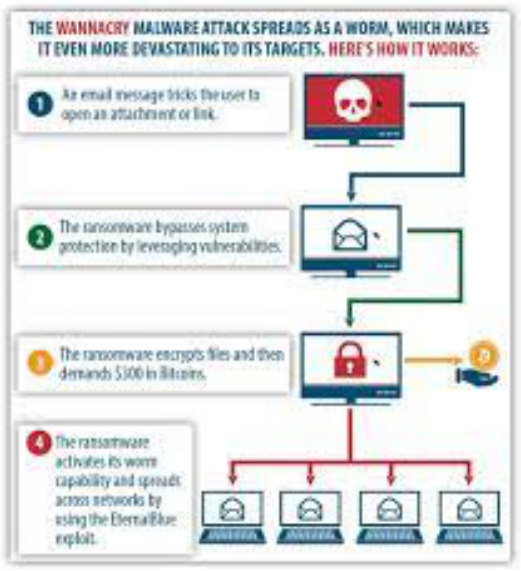

Fig. 3. Wanna cry attack

Fig. 4. Worm virus

4) Data base information: SQL tables that were saved on the machine were discovered in the database section. Although the team was able to recover the database tables' structure, we were unable to recover their contents. The tables in this situation do not include anything suspicious based on what we retrieved, there is still a method of recovering dangerous content from a pagefile in other cases.

ID	Column name	Type	Default value	Not null
0	id	Int32		True
1	originAttributes	String		True
2	name	String		False
3	value	String		False
4	host	String		False
5	path	String		False
6	expiry	Int32		False
7	lastAccessed	Int32		False
8	creationTime	Int32		False
9	isSecure	Int32		False

Fig. 5. Example of a table recovered from the database

ID	Column name	Type	Default value	Not null
0	name	String		True
1	origin	String		True
2	version	Int32	0	True
3	last_vacuum_time	Int32	0	True
4	last_analyze_time	Int32	0	True
5	last_vacuum_size	Int32	0	True

Fig. 6. Example of a table recovered from the database

ID	Column name	Type	Default value	Not null
0	id	Int32		True
1	icon_url	String		True
2	fixed_icon_url_hash	Int32		True
3	width	Int32	0	True
4	root	Int32	0	True
5	color	Int32		False
6	expire_ms	Int32	0	True
7	data	Byte[]		False

Fig. 7. Example of a table recovered from the database

B. System Information

The pagefile.sys file can be used to retrieve system information, which includes everything from the operating system to the hardware level. The type of information retrieved goes as follows:

- Operating System
- Processor Identifier
- Processor Level
- Computername
- Number of processors
- Processor Arquitecture
- Homepath
- Main storage Unit

The information was retrieved using a file containing the strings from pagefile.sys, and it began with a simple textual glance through the file, which allowed us to understand that this information existed and how it was shown.

With this, a Python parser was constructed that stores the identifiers of lines where this information occurs and matches every line to every stored information; if a match is found, the line is cleaned (special characters removed) and put in a dictionary with the number of times it occurred.

The information retrieved revealed several inconsistencies, as the strings command would occasionally catch only part of the system information or other strings in each line adjacent to this information. This was a problem because there would be repeated and useless results, because the correct lines occurred way more frequently only the most popular results are shown. The information from our case study showed the following results:

```
OS                          Windows_NT
PROCESSOR_IDENTIFIER        Intel64 Family 6 Model 158 Stepping 10, GenuineIntel
PROCESSOR_LEVEL             6
COMPUTERNAME                WIN-RIB2EUSD9GJ
NUMBER_OF_PROCESSORS        1
PROCESSOR_ARCHITECTURE      AMD64
HOMEPATH                    \Users\Cruzd
HOMEDRIVE                   C:
```

Fig. 8. System information results

C. FileSystem

When initially looking at the strings retrieved from the pagefile.sys, there were a lot of folder and file locations embedded in this data. From the information in the preceding subsection IV-B, the main drive is disk C, which means that files with the pattern "C:\\..." will contain the most relevant information from the file system. A regex pattern is used in the script to match every line of the strings file with that pattern, resulting in a collection of all folders found. [2]

To make a more functional version, these folders must be separated, and the team can establish a hierarchy of directories and reconstitute a part of the system from the Virtual Machine by using nested dictionaries. The output of this reconstruction is printed in a tree-like format to aid understanding of the system.

It was evident after executing this script with a pattern that tried to match all directories on the C drive that a lot of meaningless data was retrieved (dll files and other standard windows files and directories). As a result, another pattern was used, this time with only the directories that were the most interesting.

The script is flexible enough to reconstruct other file systems with different drives and different patterns for directories. These are the ones:

- C:\Users\Cruzd - This would be previously identified as the main user so it is worth considering and probably where most personal information will be stored
- C:\Users\Public - Not as useful in our case but with multiple users and with sharing of information this directory can become extremely useful
- C:\Program Files - Access to installed programs with 64-bit architectures
- C:\Program Files (x86) - Access to installed programs with 32-bit architectures
- C:\Program Data - Access to possible stored data by programs

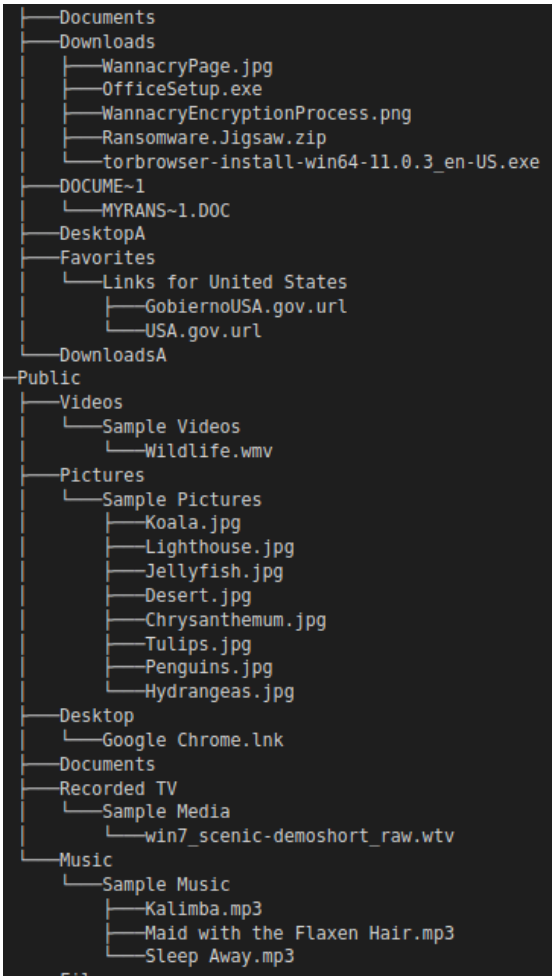

Fig. 9. Truncated program output

D. Common Search

The team questioned what could be extracted from all of this data because this type of memory displayed a lot of web pages, JavaScript, and url content. The issue stems from the fact that JavaScript data is essentially one continuous string of scripts, making matching this large string with certain patterns difficult because the team would have to sift through it all to get the important information. This led to the creation of a simple script that would search the strings file for specific strings and display the next x characters as well as the x characters behind the information (With x being customized). This can be used to find information such as:

- Passwords
- Usernames

Any other information can be collected using this script. If an analyst previously knows what we he is looking for this can be even more useful.

- Are there any references to a certain website?
- Are there some suspected parts of certain credentials?
- Is this a virtual machine? What if a search for "VMware" is made?

These are only a few examples and as will be discussed after, there might be a lot of data the team didn't consider because the information just wasn't considered.

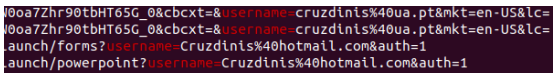

Fig. 10. Example of looking for the string "username="

E. Emails

While examining the strings file of pagefile.sys, the team found several references of email addresses throughout the file. These references can be found in HTTP requests, isolated on a line, in environment variables... Some of these emails, especially those found on HTTP requests show evidence of two things:

- An internet account being used by the PC-user
- Accounts the user interacted with (email, Skype, Messenger, etc...)

So, finding the emails in memory is fundamental to establishing a baseline of both user accounts and accounts that the user interacted with.

A script was developed to parse the memory text file and extract the emails. Fig. 11 is an example of some data that the script extracted. The text file is read line-by-line and a regex matcher finds all references, in a line, to a regex pattern - the resulting output is then written to a file.

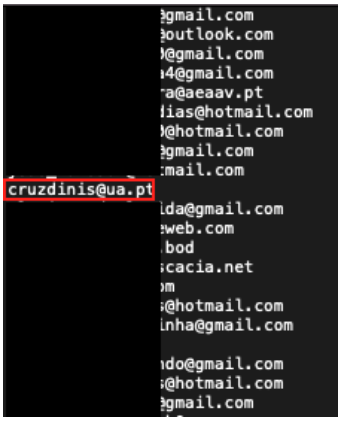

Fig. 11. Example emails extracted (user email outlined in red), other emails censored because of GDPR.

In section V the value of this type of information will be discussed and analyzed.

F. Requests

In section 11, it was debated that the memory contains urls, and the content of the urls was parsed to find usernames and passwords. In section IV-A1 the Belkasoft tool managed to extract some information about browser searches and actions. This can be useful for a generalist view of the user's web interactions.

But what about general HTTP requests? Having a broader view of the HTTP requests stored in this memory can also help to establish a baseline of the user's behavior. For instance, an analyst can see what websites were visited and can trace the path within the website: since each time the user changes webpage a new request is made. Having this raw data extracted allows for a more detailed (but also harder) investigation.

V. Forensic Value

This section will examine the forensic value of artifacts collected from pagefile.sys, and will try to demonstrate the vast number of paths that can be taken with this information, and that any forensic investigation can be based on any of these paths, and that by combining all of these pieces of information, we can get a really detailed set of information.

A. Browser Search

Looking into the browser's information that was obtained by analyzing pagefile.sys if can be concluded that the user had some strange activity looking for ransomware tools and even downloading them. He mainly used the firefox browser for these suspicious activities. Google chrome (which is his main browser) is used for his daily routine where he stores his accounts and in this one it was discovered that he has an account with the name "Cruzd".

B. Timeline

By looking at the timeline in section III, the user logged in the browser with his username "Cruzd" and started looking for some suspicious content about ransomware tools and cyber-attacks and even downloaded some of them. Then he realized that he was in a normal browser and his acitivity would be registered in the laptop, so he then downloaded TOR and maybe started navigating there because there is no more suspicious content after that there is no way to recover TOR searches.

C. Images

In the images that were recovered there were a lot of dubious content. Many ransomware tool images and cyber-attack banner photos were uncovered, indicating that the user looked through pages discussing these topics. There are also photos of the TOR browser, indicating that it was used.

D. Database Information

The tables presented in the section III were just examples of what was recovered, more table structures were recovered but they were similar to those. The team believes that no truly useful information (refering to databases) was in this pagefile because only browser databases were captured.

E. File system

The file system can be used to extract valuable data, which can include both system programs and files on disk.
Identified programs installed:

- Office
- Access Data FTK Imager

- Firefox
- Internet Explorer
- OneDrive
- Tor Browser
- Windows Media Player
- Microsoft Teams
- Java (jre1.5.5)

This can assist to form a broad view of what the user is doing with this computer, and it can be used with other evidence to build a solid case that the user was involved in criminal behavior. Interesting files encountered:

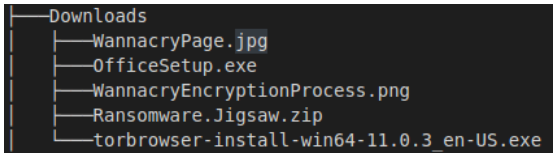

Fig. 12. Download folder collected from the pagefile.sys

The image of what this machine was used for continues to form with access to the download folder. Because the pagefile was prepared by us, not all of the information present within the computer was included in this extraction; this is due to the volatility of the information included inside the pagefile. Even so, several crucial files were discovered.

F. System Information

The results of the system information are interesting because they help us categorize the type of computer we're looking at and what it can perform, especially the number of processors, which helps us comprehend its processing capabilities.
It's also helpful to know who the system's primary user is and what the primary disk is. Because some analyst now knows the user and the main drive, it can make greater use of scripts like the file systems script.

G. Common Search and Miscellaneous Results

As previously said, these results are contained within a study that may provide useful information. This data may include usernames for specific websites or, better yet, a username and password. If only credentials are found, try's using previously encountered website can be used to get access to an account. Certain suspicions may be validated as a result of this type of data recovery, such as knowing a specific website and looking for it, or knowing bits of a password or an email and attempting to reconstruct the remainder. One interesting use was a search for "VMware" [1] that yielded a slew of hits; while this isn't proof that a user is within a virtual machine, it can lead to a better search in that regard.


```
"hdd":{
  "binary":{
    "model":"VMware, VMware Virtual S",
    "revision":"1.0","type":"HDD"},
  "profile":{
    "model":"VMware, VMware Virtual  S",
    "revision":"1.0",
    "type":"HDD"},
 "system":{
    "model":"VMware, VMware Virtual S",
    "revision":"1.0",
    "type":"HDD"}}
```


This is part of a Firefox process that was placed within the pagefile.sys, leading us to believe that this extraction was inside a VM, which is correct.

The erratic nature of data from pagefile.sys makes this search difficult and unreliable on all systems. But continues the trend of searching for things believed to be true.

H. Emails, passwords, and usernames

Sections IV-E and IV-D explore the topic of emails and passwords/usernames, respectively. This is some of the most fundamental data, for forensic purposes, found in the page-file.sys.

This leads us to two topics that can be explored with this information: OSINT and account access.

1) OSINT: The emails extracted from the pagefile.sys allow a forensic analyst to establish a network of people (other emails) that the user has interacted with. Because the emails of these people are on the pagefile.sys it means the user has, in some way, interacted with the email - the email can be in the HTTP request of some messaging website, it can be a Skype link with the email in it, etc... Also, with further OSINT investigation, an analyst can find social-media usernames associated with an extracted email and then try to extract other information on pagefile.sys where that username is present.

2) Account Access: The emails, usernames, and passwords extracted can lead to account access. Joining this information extracted on sections IV-E and IV-D can be crucial to the development of a forensic investigation. If an analyst has extracted the computer's user email and it's password it's possible to access,for example, its email account - this is highly valuable.

On section IV-A1, there are references to websites that the user has actively-searched for and logged in. Cross-referencing this information with emails/usernames/passwords, can lead to multiple accounts being accessible, which further proves how valuable this data can be.

I. HTTP Requests

Figure 13 shows some HTTP requests that were extracted from the memory file. It shows that the user searched for hacking posts in Reddit and also was viewing malware binaries in GitHub. This can be useful, for example: Imagine the computer is from a hacker that was caught after hacking a company and infecting its systems with malware - knowing which malware or which methods were used can come a long way to help the company recover from the attack.

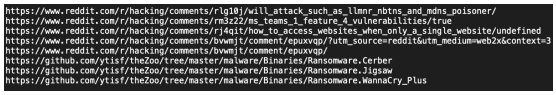

Fig. 13. Example extracted HTTP requests

VI. Future Work

Until now, the usage of pre-existing software tools like Belkasoft Evidence Center has been referenced throughout this paper. Even more, scripts (tools) were developed by the team in order to extract some information that was not represented in existing tools.

The developed scripts are somewhat basic - they only match a string from the file to a regex pattern. Technologies like machine learning could be implemented in order to not only better find these elements like emails, etc, but as well to filter the results - often, some of the results extracted had data that didn't contextually matter despite of matching the regex pattern: support emails, JS/Bootstrap/JQuery vendor HTTP requests, etc... These elements don't matter to forensics, and machine learning could help by filtering the data.

Besides this, the pagefile.sys might contain more interesting information. Being such a big file it's hard to notice which elements can be extracted or which ones can be useful for forensic purposes. Thus, in the future, further research should be done on what types of data can be extracted.

VII. Conclusion

RAM is a big part of forensic analysis, more specifically, page memory analysis. This work explores artifacts with software tools like Belkasoft Evidence Center and with proprietary scripts developed in order to extract artifacts that tools like these didn't extract.

By doing this work, the team learned a lot by exploring the pagefile with pre-made tools but also analyzing it ourselves with a "naked-eye" and truly trying to comprehend the data in the memory file. This led to the results detailed in previous sections which were more than satisfactory because they proved that it is possible to get lots of important data from a "simple" component like pagefile.sys: from emails, passwords, usernames, web surfing data, etc...

To conclude, this sub-type of forensic analysis proves to be a viable component of a complete forensic analysis of a computer. This, because throughout the report there is a direct correlation between what was done (in practice) in the computer, to what was found while doing the forensic analysis.

References

[1] Vmware. (2021). VMware. https://www.vmware.com/
[2] Regular Expressions. (2021). Regular Expressions Information. https://www.regular-expressions.info/python.html.
[3] FTK Imager. (2021). FTK Imager Access Data. https://accessdata.com/
[4] Oleg Skulkin, Scar de Courcier, 2017, Windows Forensics Cookbook, Birmingham-Mumbai, Packt Publishing Ltd.
[5] Knut Bellin, Social network forensics: using commercial software in a university forensics lab environment, https://www.researchgate.net/publication/258332972_Social_network_forensics_using_commercial_software_in_a_university_forensics_lab_environment

VIII. Glossary

pagefile.sys File where additional RAM memory is temporally stored when RAM full capacity is hit;

Random Access Memory (RAM) used to store data and machine code;

Ransomware Malware that employs encryption to hold a victim's information at ransom;

Virtual Machine (VM) Virtual emulation of a physical computer system;

FTK Imager Tool used to extract a memory dump with the pagefile.sys;

Belkasoft Evidence Center (BEC) Forensic tool for acquiring, locating, extracting, and analyzing forensic artifacts stored in pagefile.sys;

Regex Sequence of characters used to match string search patterns;

Appendix A
GitHub repository with all scripts developed

Link: https://github.com/Barroqueiro/Pagefile.sys-Analysis.

APPENDIX B
SYSTEM INFORMATION SCRIPT

```
import re

info = ["PROCESSOR_LEVEL=","HOMEPATH=","OS=", "NUMBER_OF_PROCESSORS=","HOMEDRIVE=",
"COMPUTERNAME=","PROCESSOR_ARCHITECTURE=","PROCESSOR_IDENTIFIER="]
information_string = ""
for i in info:
    information_string += i +"|"
information_string = information_string[:-1]
FILE = "strings2.txt"

set_information = {}

with open(FILE) as f:
    line = f.readline()
    while line:
        result = re.match(information_string, line)
        if result:
            line = line.replace("\n","")
            if line in set_information:
                set_information[line] += 1
            else:
                set_information[line] = 1
        line = f.readline()

set_information = dict(sorted(set_information.items(), key=lambda item: item[1],
                                                                    reverse=True))

count = 0
for i in set_information:
    splited = i.split("=")
    print("{:<30} {:<50}".format(splited[0], splited[1]))
    count += 1
    if count == len(info):
        break
```

APPENDIX C
FILE SYSTEM SCRIPT

```
import re

DISALLOWED_CHARACTERS = "*\n\r\t?;"
PATTERN = "C:\\\\Users\\\\Cruzd|
C:\\\\Program Files
|C:\\\\Program Files (x86)|
C:\\\\Users\\\\Public|C:\\\\ProgramData"
FILE = "strings2.txt"

def clean_list(path_list):
    res = []
for path in path_list:
        for c in DISALLOWED_CHARACTERS:
            path = path.replace(c,"")
        if path != "":
            res += [path]
```

```
    return res

def print_tree(directories, before):
    space =  '    '
    branch = '   '
    tee =    ''
    last =   ''
    count = 0
    length = len(directories)
    for d in directories:
        if length == 1:
            print(before+last+d)
        elif count == 0:
            print(before+tee+d)
        elif count < length-1:
            print(before+tee+d)
        else:
            print(before+last+d)
        if count == length-1:
            print_tree(directories[d], before+space)
        else:
            print_tree(directories[d], before+branch)
        count+=1

set_directories = set()

with open(FILE) as f:
    line = f.readline()
    while line:
        result = re.match(PATTERN, line)
        if result:
            set_directories.add(line)
        line = f.readline()

result = {}

for directory in set_directories:
    splited_directories = directory.split("\\")
    splited_directories = clean_list(splited_directories)
    last_keys = []
    for folder in splited_directories:
        if last_keys == []:
            if folder not in result:
                result[folder] = {}
            last_keys.append(folder)
        else:
            temp = result
            for last in last_keys:
                temp = temp[last]
            if folder not in temp:
                temp[folder] = {}
            last_keys.append(folder)

print_tree(result,"")
```

APPENDIX D
COMMON SEARCH SCRIPT

```
    import re
    import sys
    from termcolor import colored

    FILE = "strings2.txt"
    SIZE = int(sys.argv[2])
    PATTERN = ".*"+sys.argv[1]+".*"
    print("Searching for pattern: ",
        PATTERN)

    with open(FILE) as f:
        line = f.readline()
        while line:
            result = re.match(PATTERN, line)
            if result:
                a = re.search(r'\b({})\b'.format(sys.argv[1]),
                    line)
                if a != None:
                    start = a.start()
                    if start > SIZE:
                        print(colored(sys.argv[1],'red'),end="")
                        print(  line[start+(sys.argv[1]):start+len(sys.argv[1])+SIZE]
                                                                .replace("\n",""))
                    else:
                        print(line[:start].replace("\n",""),end="")
                        print(colored(sys.argv[1],'red'),end="")
                        print(line[start+len(sys.argv[1]):start+len(sys.argv[1])+SIZE]
                                                                .replace("\n",""))
            line = f.readline()
```

APPENDIX E
EMAILS AND REQUESTS SCRIPT

```
import re
import signal
from contextlib import contextmanager
fileToRead = 'strs'
class TimeoutException(Exception): pass
@contextmanager
def time_limit(seconds):
    def signal_handler(signum, frame):
        raise TimeoutException("Timed out!")
    signal.signal(signal.SIGALRM, signal_handler)
    signal.alarm(seconds)
    try:
        yield
    finally:
        signal.alarm(0)

def emailRegex(string):
    regex = r"^[a-z0-9]+[\._]?[a-z0-9]+[@]
        \w+[.]\w{2,3}$"
    emails = re.findall(regex, string)
    return [x[0] for x in emails]
```

```
def httpRegex(string):
    regex = r"(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)
        (?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+
        |(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:'\".,<>?\"'']))"
    url = re.findall(regex, string)
    return [x[0] for x in url]


def writeFile(listData):
    file = open(fileToWrite, 'w+')
    strData = ""
    for item in listData:
        strData = strData+item+'\n'
    file.write(strData)

if __name__ == "__main__":
    emails = []
    env_vars = []
    urls = []
    file = open(fileToRead, 'r')
    listLine = file.readlines()
    counter = 0
    for line in listLine:
        print(counter, line)
        counter += 1

        lines = line.split()
        for line in lines:
            try:
                with time_limit(1):
                    f_urls = httpRegex(line)
                    urls += f_urls
            except TimeoutException as e:
                print("Timed out!")

            try:
                with time_limit(1):
                    f_emails = emailRegex(line)
                    emails += f_emails
            except TimeoutException as e:
                print("Timed out!")

    fileToWrite = 'emailExtracted.txt'
    if emails:
        uniqEmail = set(emails)
        print(len(uniqEmail), "emails collected!")
        writeFile(uniqEmail)
    fileToWrite = 'urlsExtracted.txt'
    if urls:
        uniq = set(urls)
        print(len(uniq), "urls collected!")
        writeFile(uniq)
```