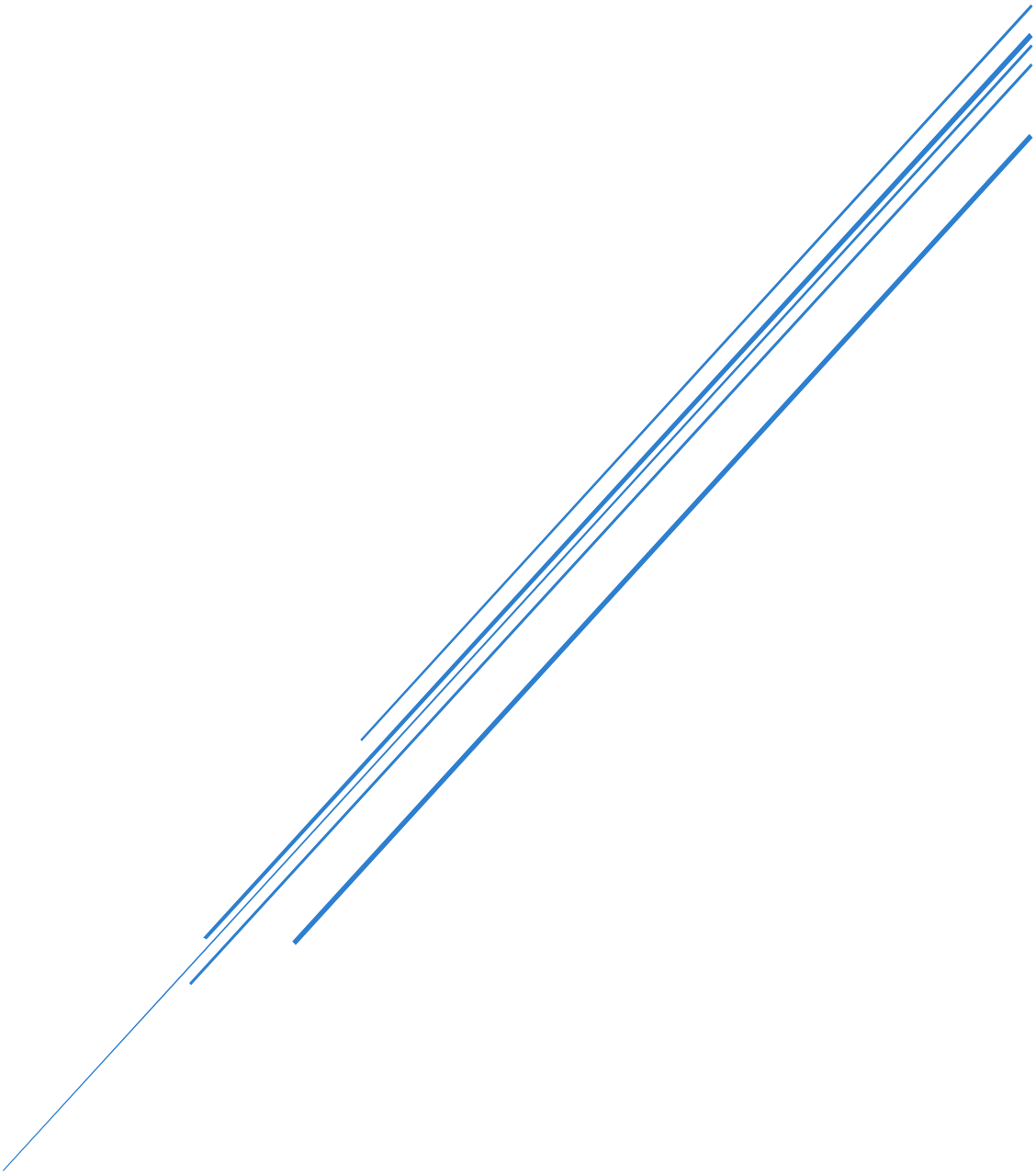
P6 - WINDOWS ADDRESS SPACE SIGNATURES

Forensics Project



Pilar López Bodoque
Victor Fresno Gómez

Index

Introduction

The objective of this project, titled "Windows Address Space Signatures," was to analyse the stability of various fields within critical kernel data structures in a Windows 10 64-bit environment. Over the course of the project, multiple snapshots were taken at different times from the same machine to determine which fields in structures such as _PEB, _TEB, _ETHREAD, _EPROCESS, and _MMVAD remained constant and which ones changed over time.

To facilitate this analysis, we employed Volatility 3, a powerful memory forensics framework, utilizing its plugins to scrutinize process-related data. However, Volatility plugins alone could not provide a comprehensive view of all relevant fields. Thus, we also made extensive use of volshell, Volatility's interactive shell, which allowed us to delve deeper into the memory dumps and inspect fields that the plugins did not cover.

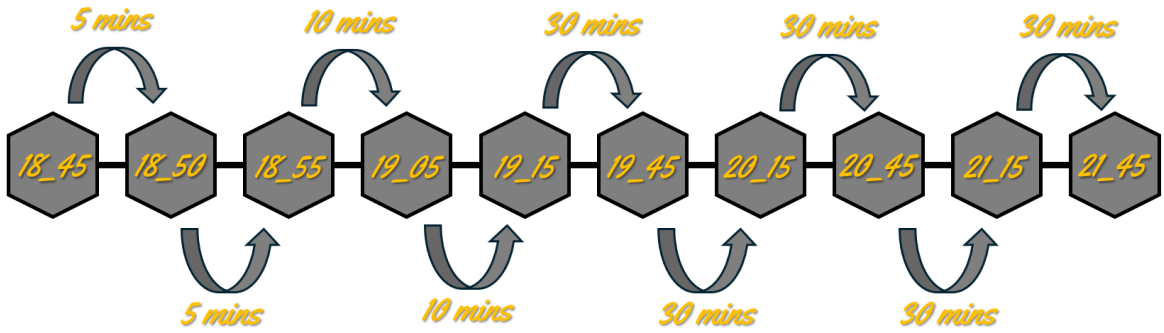
During the project's development, we received guidance from Andrea Olivieri, who advised us against focusing on the _MMVAD field due to its complexity, which could render the project unmanageably intricate. Instead, he recommended concentrating on the _EPROCESS field when using volshell, as it provided a more manageable and informative scope for our analysis.

By integrating Volatility plugins with volshell, we aimed to achieve a comprehensive understanding of the temporal stability and variability of key kernel data structures, providing valuable insights into the behavior of the Windows 10 operating system.

Memory dumps capture

The project requires us to take memory dumps from a Windows 10 system, we took 10 memory dumps. To take them we used a **Windows 10 VMWare virtual machine**, and we installed in it the program **FTK Imager**. FTK Imager is a forensic tool used by digital investigators to capture and analyse data from computers and other devices. It helps in creating exact copies (images) of hard drives, flash drives, and even CDs without altering the original data.

When we had the VM ready we rebooted it. As soon as it turned on, we took the first memory dump. For the next nine we waited: 5 minutes, then 5 minutes, then, 10 minutes, then 10 minutes and for the next five we waited 30 minutes. The output format of the dumps is .mem and the name is the hour_minutes we took the memory dump.



Volatility 3

To analyse the previously taken dumps we started using volatility3 plugins and comparing them. We used the plugins info, pslist, thrdscan and vadinfo. However, we compared only pslist and thrdscan, as we considered info did not have relevant information that could change with time, and we did not compare vadinfo as we had a meeting with Andrea Olivieri who advised us against focusing on the _MMVAD field due to its complexity**.** To achieve this, we created three scripts:

- **analyze_snapshots.py:** this python script runs the four volatility plugins mentioned above (info, pslist, thrdscan and vadinfo) in all the memory snapshots. It saves the results in a folder called “volatility_outputs”, which will be in the same directory of the script, and it will save the results with the following format: snapshot-name_plugin.txt. For example, the result of the plugin pslist for the snapshot 18_45.mem would result in a txt file named 18_45_pslist.txt inside the folder “volatility_outputs”. Inside the txt files the results of the plugins will be stored in JSON format. To run this script, it is necessary to modify the path to the memory dumps to analyse and the path to volatility3 at the beginning and at the end of the script.
- **pslist_compare.py:** this python script must be **run after running analyze_snapshots.py**. It compares the txt files containing the results of the pslist volatility3 plugin in all the snapshots. This script compares the pslist result of a snapshot with the pslist of the snapshot taken before in time. It compares all the snapshots with their previous one and saves the results of all the comparisons in a txt file called pslist_comparison_results.txt. This file contains for each comparison the number of changed processes, the number of created processes, the number of terminated processes and three types of tuples: tuples with the PID of the processes that changed, with the field that changed and its previous and current value; tuples with the PID of the processes that were created and all the fields for those processes; tuples with the PID of the processes that were terminated and all the fields for those processes. To run this script, it is necessary to modify the path to the “volatility_outputs” folder that the previous script “analyze_snapshots.py” generated.
- **thrdscan_compare.py:** this python script must be **run after running analyze_snapshots.py**. It compares the txt files containing the results of the thrdscan volatility3 plugin in all the snapshots. This script compares the thrdscan result of a snapshot with the thrdscan of the snapshot taken before in time. It compares all the snapshots with their previous one and saves the results of all the comparisons in a txt file called thrdscan_comparison_results.txt. This file contains for each comparison the total number of changes in the threads, the number of new threads, the number of terminated threads, the number of

distinct TIDs that have changed, the number of changes in each field and three types of tuples: tuples with the TID of the threads that changed, with the field that changed and its previous and current value; tuples with the TID of the threads that were created and all the fields for those threads; tuples with the TID of the threads that were terminated and all the fields for those threads.
To run this script, it is necessary to modify the path to the “volatility_outputs” folder that the previous script “analyze_snapshots.py” generated.

Volshell

The second part of the project consisted in analysing the **EPROCESS** with Volshell to obtain all its possible fields and have a more complete solution. To achieve this, we created three scripts:

- **automate_volshell.py**: The purpose of this script is the automatic execution of a certain Volshell script in all the different memory dumps and the organization of the generated files. To make it run you should modify: the path where the memory dumps are store, the path to the volatility3 folder (in the executed command) and the path to the script you want to execute.
- **EPROCESS_script.py**: This script must be executed from the previous script or independently in a single memory dump. This script obtains all the fields of every EPROCESS in a memory dump and store them in a JSON format in a folder. To make it run you should modify the path to store the JSONs.
- **EPROCESS_compare.py**: This script must be executed after running **automate_volshell.py.** The scripts compare the JSONs generated temporally, this means, the first one with the second one, the second one with the third one and so on. The scripts generate two files: a complete comparison and a brief comparison. The brief one contains, for each comparison: the number of processes added, number of processes terminated, number of processes changed, number of changes per field, the added PIDs, removed PIDs and changed PIDs. The complete version contains the same but also what are the changes between one memory dump and another. To make it run you must modify the path where the memory dumps are stored.

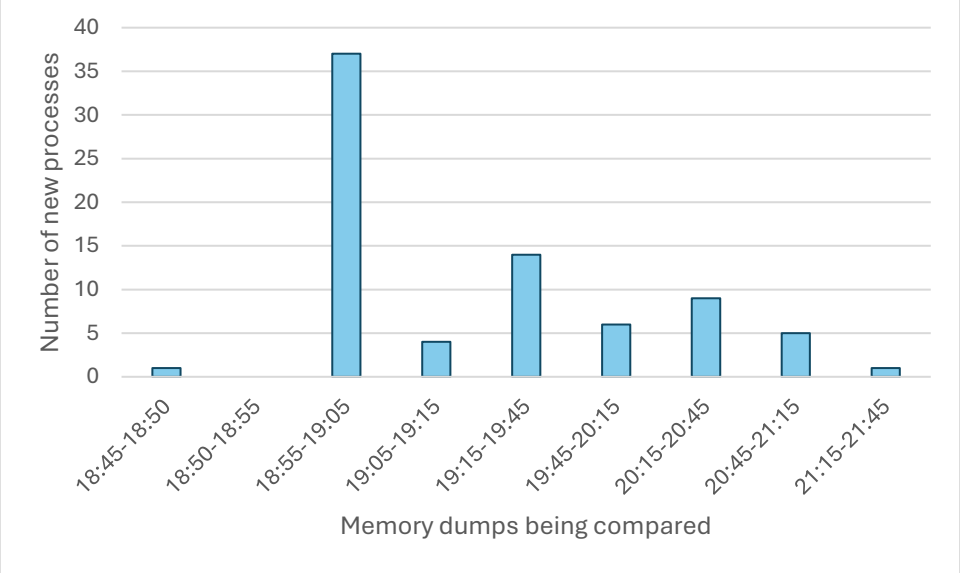
We tried to make the same with the PEB. We managed to obtain the PEB of each EPROCESS. Some had some didn´t have. However, when we tried to obtain the values of their fields we obtained the following error: PagedInvalidAddressException: Page Fault at entry 0x0 in table page directory pointer. After some investigation we found that this might be an error related to the memory dumps, an irreparable mistake.

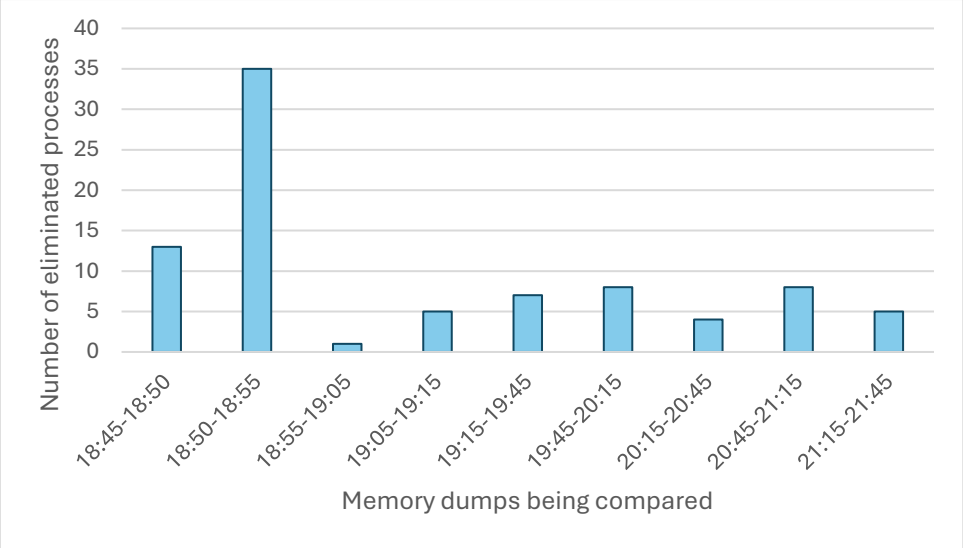
Results

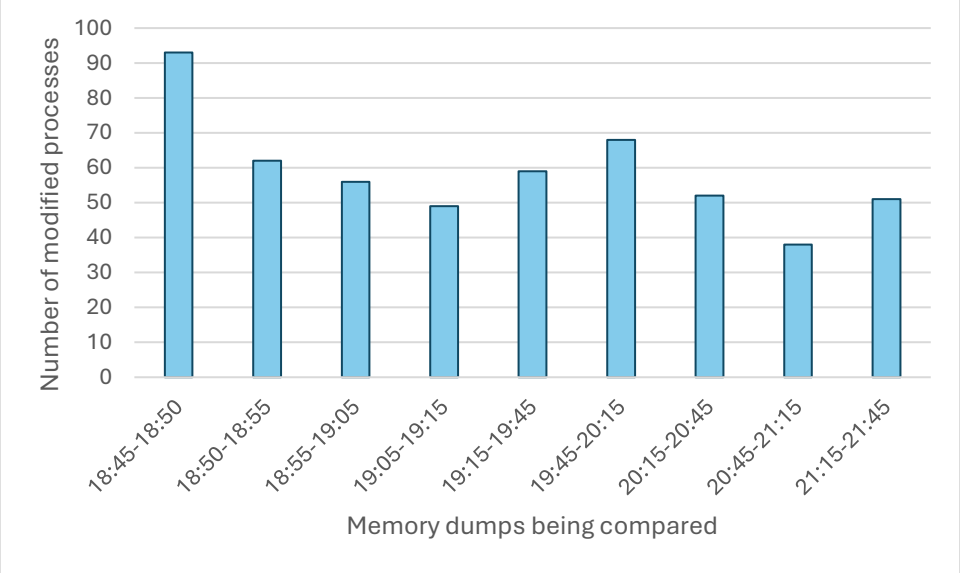
The volatility and volshell scripts and all the files produced by them with **the results have been uploaded to the following Github repository:** vfg27/Windows_address_space_signatures (github.com). Memory dumps have been uploaded to **OneDrive** due to their heavy weight: archive.7z

In this section we will present some graphs with the results obtained in volatility and volshell. However, to have a clear insight of the results the best way is to look at the files “thrdscan_comparison_results.txt” and “pslist_comparison_results.txt” for the volatility results and look at the files "comparison_results_complete.txt" and "comparison_results_brief.txt" for the volshell results. All these files have been uploaded to the Github repository.

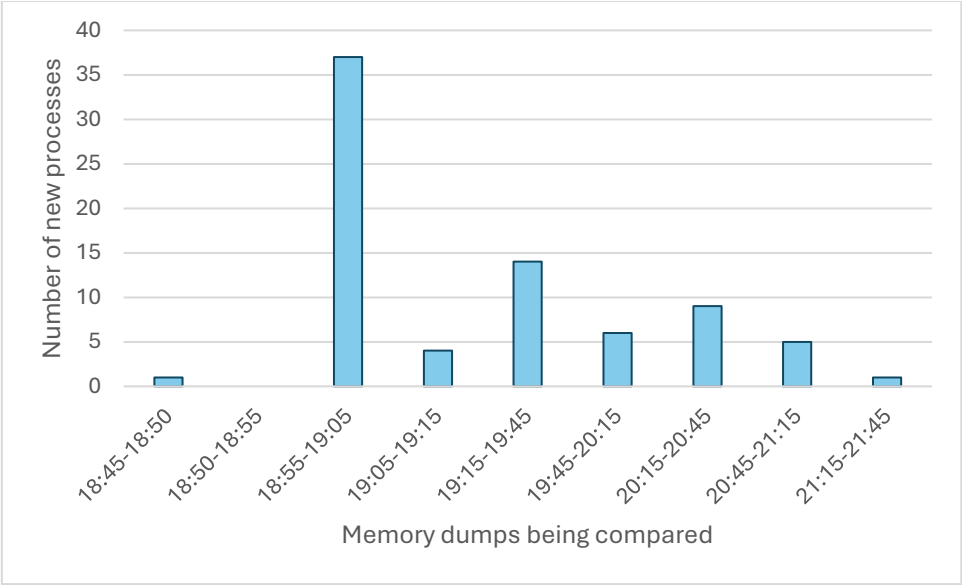
From the results obtained with the volatility scripts, three graphs were created: one representing the evolution with time of the number of new processes, another one with the evolution of the number of eliminated processes and finally one with the evolution of changed processes.

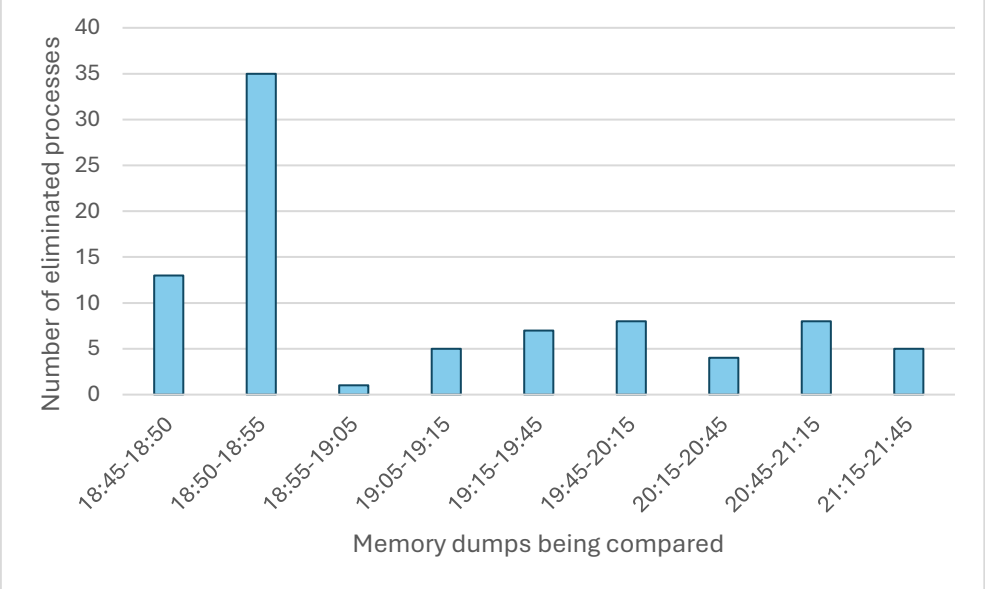


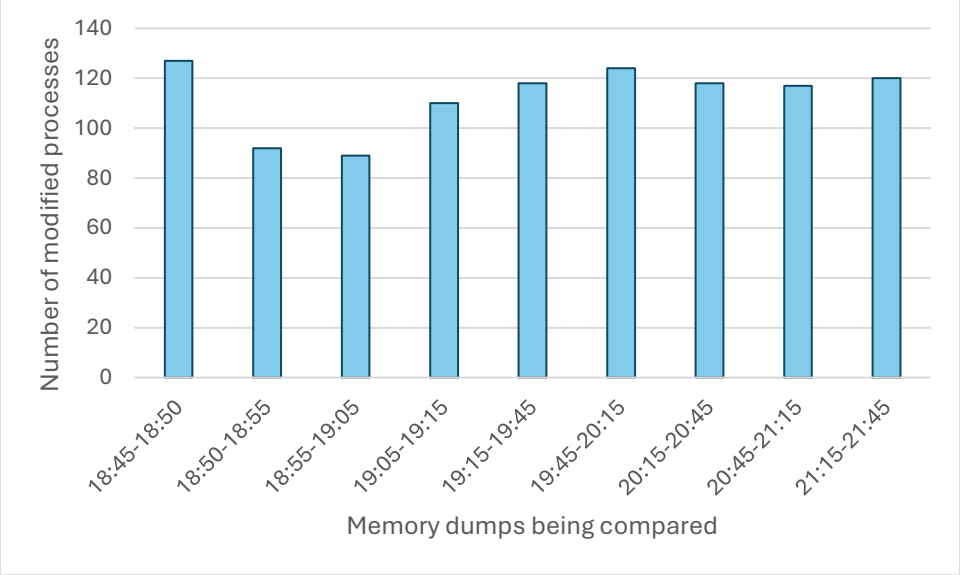

Number of eliminated processes
40
35
30
25
20
15
10
5
0
18:45-18:50
18:50-18:55
18:55-19:05
19:05-19:15
19:15-19:45
19:45-20:15
20:15-20:45
20:45-21:15
21:15-21:45
Memory dumps being compared


Number of modified processes
100
90
80
70
60
50
40
30
20
10
0
18:45-18:50
18:50-18:55
18:55-19:05
19:05-19:15
19:15-19:45
19:45-20:15
20:15-20:45
20:45-21:15
21:15-21:45
Memory dumps being compared

With the data obtained from the volshell scripts we created some graphs. In them we can see the evolution of added, eliminated and modified EPROCESS:







The number of modified processes is greater in Volshell because it collects more data. Consequently, the data obtained with Volshell is more precise and should be prioritized.