

Lab 5

Windows Memory Analysis

ITSC205: Operating Systems Internals

**NAME**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

[Lab Outcome(s) 2](#_Toc472777602)

[Reading 2](#_Toc472777603)

[Introduction 2](#_Toc472777604)

[1.0 Memory Information and Settings 3](#_Toc472777605)

[2.0 Memory Analysis with System Internals tools 6](#_Toc472777606)

[3.0 Memory Analysis with Windows Debugger](#_Toc472777606) ……………………………………………………9

[3.0 Windows Stack overflow - Debugger](#_Toc472777606) …………………………………………………………12

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

|  |  |  |
| --- | --- | --- |
| Examine Memory Information and Settings | 15 |  |
| Memory Analysis with System Internals Tools | 20 |  |
| Memory Analysis with Windows Debugger -Windbg | 15 |  |
| Windows Stack Overflow analysis with Windbg | 15 |  |
| TOTAL MARK | 65 |  |

Lab Outcome(s)

* Examine Windows memory settings.
* Examine Windows memory management techniques.
* Analyze memory behavior of normal system operations.

Reading

* Textbook sections Windows chapter -21.3.5.2 ( Virtual Memory Manager)

Introduction

Virtual memory is a partition in secondary storage that stores programs’ pages that are not required by a process at running time. When a process is started the operating system only loads into RAM the pages that are needed (demand paging) at that time. These pages are stored in secondary storage because physical memory RAM is never enough to store all programs, applications, processes running at a giving time

When CPU requests a page, it will check cache (TLB) to find the physical location of the page, if the translation is not in cache it will check for the address translation in the process’s page table. Once the page is founded in RAM it is processed by the processor. If the page is not founded in RAM in the active, legal region, the operating system is trap to start page fault process (check on secondary storage) if page is founded in secondary storage the operating system will start swapping process, if not it will generate a memory error such as access violation

1. Memory Information and Settings \_\_\_/15
2. Use the Windows System Information tool to record the following values (including units):

|  |  |
| --- | --- |
| *Physical Memory* | *Virtual Memory* |
| Total | Total |
| Available | Available |

1. What is the purpose of Windows **Page File**?
2. What is the Page File Space of your system?
3. Where is the Page File located (path) ?
4. Use performance monitor to analyze memory usage per process
5. Start notepad process.
6. Add a counter and select **process** object
7. Under instances, select Notepad instance.
8. On the counters list, **select** the following counters and explain each one. Check the box “Show description” and write down the description of each of the following:
   * 1. Page faults
     2. Pool Nonpaged Bytes
     3. Pool Paged Bytes
     4. Private Bytes
     5. Working Set
9. Select page fault, working set and private bytes **only** for notepad process. After adding these counters analyze the graph and results. Use notepad (type something and try to save the file).Analyze the process page faults and working set changes.
10. Attach a screen capture to demo notepad activity. Graph with page faults , working set and private bytes for notepad process **only**
11. Start Task manager and add the following columns: Working set, page faults. Write down the working set size for notepad process. If each page is 4KB how many pages are in used (working set) by notepad process?
12. Clear performance monitor and add a new counter, select **Memory Object**. This object displays system memory behavior.
13. Change the vertical scale maximum on the graph to 1000.
14. Add the following memory features: commit limit, Demand Zero Faults/sec
15. Create some activity in the system to see the memory performance while running different process (you can start notepad and access different web sites with animation).
16. Select the following memory features and answer the questions .You can find the definitions by checking the show description box.
    1. What is commit limit?
    2. Demand Zero Faults/sec. What are zeroed pages?

* 1. What is **Modified Page** List Bytes?
  2. Page Faults. What is the difference between **soft fault** and **hard fault**?

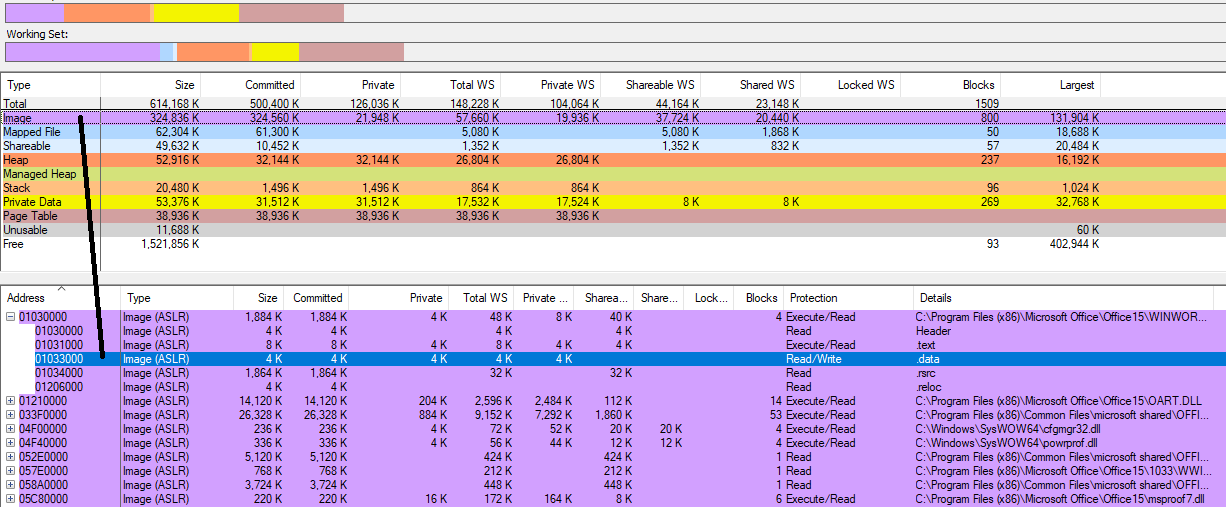
1. Add memory counters and start only one process such as the browser that uses lots of memory and analyze page faults of this process. Attach the screen capture to demo results

1. Memory Analysis with System Internals tools \_\_\_/20

**A. Vmmap tool**

The VMMap utility from Sysinternals can show you a detailed view of the virtual memory being utilized by any process on your machine, divided into categories for each type of allocation.

1. Start notepad or a Web browser process.
2. Run Vmmap utility from System Internals Suite directory.
3. To analyze virtual memory of a process , click on File > Select process and select one process: notepad or web browser or any other process that you want to analyze
4. Yu can use help to find the definition of each column or category of this utility. Click on Help > Quick help
5. Analyze base addresses, sizes, permissions of the different memory section (Image, heap, stack, private data) of selected process as follows:
   1. Click on **Image** section and in the panel at the bottom you will see the details of the image. Click on the + to display address offsets and the sections such as Header, .text, .data. Notice the first file is the executable followed by its dependent libraries(.dll).



* 1. Record the address and permissions (protection) of **.text** for the executable
  2. Is the address randomized? How do you know it?
  3. Now analyze the heap. What is the size of the largest block assigned to this process?
  4. Is this a heap with private or shared data?
  5. Now analyze the stack section. What is the stack type?
  6. The threads permissions are read/write/guard. Read <https://docs.microsoft.com/en-us/windows/win32/memory/creating-guard-pages> and briefly explain what is the meaning of page guard protection

1. To demo results attach the screen capture(s) of Vmmap for the process you analyzed
2. Do not close Vmmap you will use it with the next tool.

**B. RAMMap tool**

The movement of pages from process working set to the modified page list and then to the standby page list can also be observed with the Sysinternals tools RAMMap. This tool also displays logical and respective physical address conversion for all active processes in the system.

1. From system Internals suite , start RAMMap tool
2. Click on use count TAB and observe the usage of different page states
3. Select the processes TAB and analyze the usage of standby, modified pages and page table per process
4. If standby memory consumes all free memory it will generate **memory leak**. If the system loses gradually free memory then you should analyze standby memory behavior. It could be the cause of **memory leak**
5. Select **Physical Pages TAB**. Analyze one process (browser such as firefox, notepad or any process that is active in your system) and attach the screen capture that demo the results (virtual address and respective physical address, offset if it is there and page states of the process.
6. Find the respective physical address using RAMMap tool of a virtual address display in VMMap as follows:
   1. Go back to previous Vmmap tool results where you analyzed different sections of a particular process.
   2. Find and record the following based virtual address:
      1. **.data** for the executable
      2. **.data** for one library (.dll)
      3. Select any address from Private Data section
   3. On RAMMap tool under Physical Page TAB select the same process that you analyzed with VMMap tool and click on top of the Virtual Address column to display the addresses in ascending order
   4. Identify the respective physical address of the virtual addresses you recorded above
   5. Attach the screen capture that demo the results
   6. Now that you know the physical addresses you can use debugger windbg !dc command to analyze the content of these addresses as follows:
      1. Start windbg in kernel mode
      2. Under lkd> **!dc physical-address** ( replace physical address with each physical address you found above in RAMMap
   7. Attach the screen capture that demo results

C. Process Explorer tool

You can use process explorer from system internals tools to analyze processes and its loaded libraries. You can also verify the support of ASLR and DEP. A process can be vulnerable by just having only one loaded process’ DLL unprotected (without ASRL support)

* + - 1. Start Process explorer tool from system internals
      2. Select columns from view and from process memory TAB add Working set size and page faults columns. Which process has the highest page faults?
      3. Select columns from view and from process images TAB add the DEP status and ASLR Enabled to verify if processes are protected or not.
      4. Start a new process such cmd or notepad and verify if DEP and ASLR are enabled or not for this process
      5. Based on the working set size of this process how many pages are in used or (active) if the page size is 4KB?
      6. To display libraries associated with this process select “Show Lower Pane” from View.
      7. To analyze DEP and ASLR for the libraries select columns from view and from DLL TAB add DEP, ASLR and Base Address columns. Verify if all libraries for this process are protected or not. Explain the meaning of ASLR for a particular exec or library .dll.
      8. Identify and record the Base address of the executable for example the based address of notpad.exe if you are analyzing notepad.
      9. Close the process you analyzed and initialize it again. Verify the Based address. Is it the same address as recorded in 8.?
      10. Reboot the computer and repeat the process. Start Process explorer and start the same process you analyzed before. Is the Based address the same as you recorded in 8 ? why
      11. Attach the respective screen captures to demo the results

1. Memory Analysis with Windbg \_\_\_/15

Kernel debugging can be used to investigate user mode processes as well as internal kernel structure. Debugging tools for Windows support remote and local kernel debugging.

Instructions:

* + 1. Access C:\Program Files (x86) 🡪 Windows Kits 🡪 10 🡪Debuggers🡪 x64
    2. Run windbg as Administrator
    3. Click on File🡪 Kernel Debugging and select Local (we will debug locally).
    4. Once in local kernel –debugging mode, many commands that start with ( ! ) known as bang can be used to display the contents of internal processes/threads and memory data structure.
    5. Use Lived kernel debugger lkd > !vm to verify memory information
    6. How many page files types are there? What is the difference among them?
    7. Start notepad and use the kernel command !process 0 0 notepad.exe to find the virtual address of this process.
    8. To find the respective physical address examine the PTE table of notepad process using the following command:

lkd > !pte process-address (replace process-address with respective notepad address) and analyze the results. Notice the virtual address of the different page table levels and its contents. PFN is the data base that contains physical address with the respective flags.

Attach the screen capture that demo notepad virtual and physical address results

a. Why are the first four digits of notepad virtual address ffff ?

b. How many page table levels are here?

c. Based on the results what is the physical address of notepad including offset. Remember the offset of the virtual address is the same for the physical. PFN only displays the physical conversion without offset.

d. What are the PTE flags? Provide the meaning of the letters. e.g K is kernel code

Debugging tools can be used to attach to a user-mode process and examine or change process memory. To this point we used windbg kernel to analyze kernel code. Now you will use Windbg to analyze user mode processes.

9. Use Windows debugger in user mode **to attach a process for analysis**. To display the threads stack and modules loaded of attached process in **user mode** do the following:

* Start a user process such as: notepad
* If the debugger is running click on Debugg 🡪 Stop Debugging
* Click on File 🡪 Attach to a Process
* Select notepad
* Use ~ command to display the threads of this process

To understand the output of this command

Read Windows docs <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/---thread-status->

* Use **k\*** commands to display stack frame of a given thread
* Use **lm** command to display the loaded modules by this process
* Use **bu notepad!WinMain** to create a breakpoint at this point
* If you do not know function name (symbol) you can use **x** command to examine symbols that match specific pattern. e.g  **x notepad!** It will display all functions that contain notepad ( in this case we attached notepad process) if you attach example calculator process then you may search for **x calc!\*** For more details on how to use x check <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x--examine-symbols->
* You can also use **bp** to create a break point such **as: bp $exentry** ( $exentry is the address of the entry point of the first executable of the current process )
* You can use **u $exentry** to disassemble it and find call instructions
* Use **bl** command to verify breakpoint
* Click on View🡪 Disassembly to display assembly language for this process

You can also use the u or ub command to display assembly instructions

* Use qd to quit debugging and detach from Notepad process
* Access the following site <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/commands> to explore Windbg commands

10. Start a process to analyze (cmd, browser, calculator or any user proces). Attach the process in user mode and use the respective commands to do the following:

a. create a breakpoint and display the memory address,

* 1. display assembly language (disassembly)
  2. display thread’s status and stack
  3. use dd and dc commands to display the content of a range of memory addresses
  4. What is the purpose of dd esp ?
  5. What is the difference between dd eax and dq eax?
  6. use dd command to display the address of a register then use u command to display the assembly of founded address
  7. display the modules loaded by the process,
  8. display the content of based memory address for KERNEL32 or ntdll modules
  9. Attach respective screen captures that demo the results

1. To display memory layout for this process use **!address.** Identify the different memory regions (sections) and its respective features. Observe the different page permissions (protection).
2. To analyze the details and header of any address you can use

**!address address-to-analyze**. Use this command to analyze one address of this process and display the details with the headers (headers for PE). To display the section headers click on the hyperlink **!dh** and attach the screen capture that demo results

1. Find and record the **base address** of notepad.exe
2. Use windbg memory command ***dc*** to explore the content of the base memory founded before (based address of notepad.exe). ***dc base-address (replace base address with respective founded address)***
3. You will see in the output **MZ** . Investigate and explain the meaning of MZ. You can access the following web sites as references: <https://en.wikipedia.org/wiki/DOS_MZ_executable>

<https://asecuritysite.com/forensics/magic>

1. Attach the respective screen captures to demo the results

**4.0 Windows Stack Overflow \_\_\_/15**

Stack overflow is still the most common memory technique used to exploit operating systems. It is important to understand how stack overflow works and the tools that can be used to analyze and detect the problem. Read the following Microsoft documentation to learn more about debugging a stack overflow.

<https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-a-stack-overflow>

We will install code blocks for windows as IDE to edit a small overflow C code. To install code blocks do the following:

1. Access the following web site <http://www.codeblocks.org/>

1. Click on Downloads > Download the binary release
2. Select codeblocks-20 03mingw-setup.exe and download it from FossHub



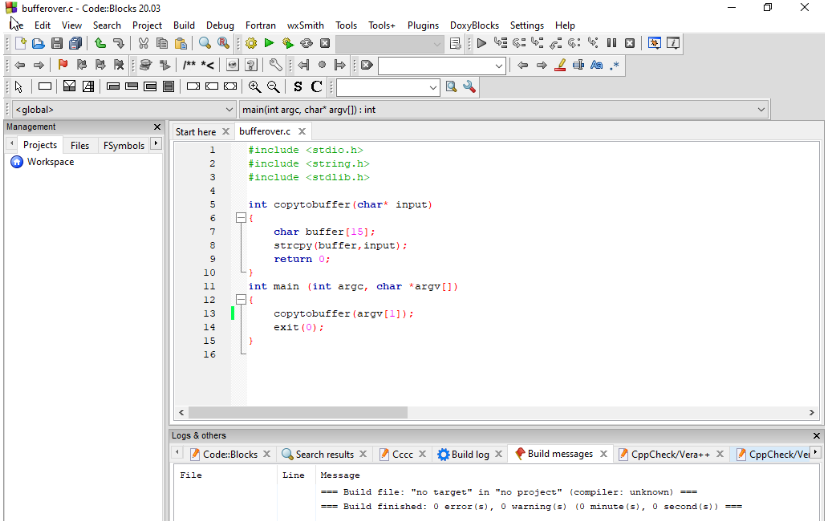
5. Save the file and install it

6. At the first boot a window will pop up to configure the compiler. Select GNU GCC Compiler, click set as default, OK. Then IDE interface will start.

7. Let’s try our vulnerable code that you used in Linux using Code::Bloks

8. On code::blocks select **File > New > Empty file** and write the following code

9. Save it as bufferover.c



10. To build the code Click on **Build > Build** Click at the bottom **Build messages** TAB to verify compilation erros

11. The compiler will generate and exec and **.o** files. Verify the path of the executable by clicking at the bottom **Build Log**

12. Provide a screen capture of created code

13. Locate the path and name of the exec.

14. Open a terminal and move to that directory and run the executable with input characters e.g **bufferover.exe aaaaaaaaaaa.** The system should return the prompt. To generate overflow and analyze the results start Windbg as Administrator

15. On Windbg click on **View** and open Command, Memory, Disassembly and Registers windows. Organize them vertically

16. Click on **File > Open Executable** and locate the bufferover executable you created before. In the same window before you click on open under arguments input 14 a(s) aaaaaaaaaaaaa now click on open and analyze the results on the windows open

17. Click on **Debug > Step Into** (instead of **Step Into** you can also use in the command window ***t*** command) to move to next line and observe the assembly register instructions and address of rsp, rbp and rip registers. Repeat **Step Into** few times to observe the changes while moving to next line of code.

**18. In the command windows use:**

a. ***k*** command to verify the stack

b. ***dc rsp-address*** (replace rsp address with the respective address from

registers window) to verify the content of this region

***!address*** to verify the memory layout of this process

c. ***!address address-of-rsp-register*** (replace address-of-rsp-register with

the address **of rsp address located in the register window).** From this results you can see the stack addresses range (Base and End address) . Record these

addresses

d. Use ***dc*** command and the ranges of stack address you recorded before to see the memory content of stack region**. e.g dc 8ef000 8f0000** (you can remove the leading zeroes)

e. Use **g** (go) command to continue and observe the final output and how registers values changed

* 1. ***Use k command to verify stack content and Return Address(RetAddress)***
  2. ***dc rsp-address*** (replace rsp address with the respective address from registers window) to verify the content of this region
  3. Notice the difference now in the stack

**19**. Attach the screen captures that demos the windows open and the commands you used to analyze results.

20. Stop debugging. **Click on Debug > Stop Debugging**

21. Now repeat previous steps to analyze stack overflow. Open the executable

with argument of 27 (a)s and 4 b(s) aaaaaaaaaaaaaaaaaaaaaaaaaaabbbb

22. what is the ASCII code of character a and b ?

23. What is the code of Access Violation?

24. Search Microsoft documentation and explain briefly this type of error (Access Violation)

24. What is the address of rsp, rbp and rip?

25. After using ***g*** command use ***k*** command to verify stack content and record the RetAddress

26. Attach the screen captures that demos the windows open and the commands you used to analyze results of stack overflow. Display the content of the stack using stack ranges of addresses (Based and End address)