LAB 16+17

Thầy Mai Hoàng Đỉnh

Trường đại học FPT

Người thực hiện

Đặng Hoàng Nguyên

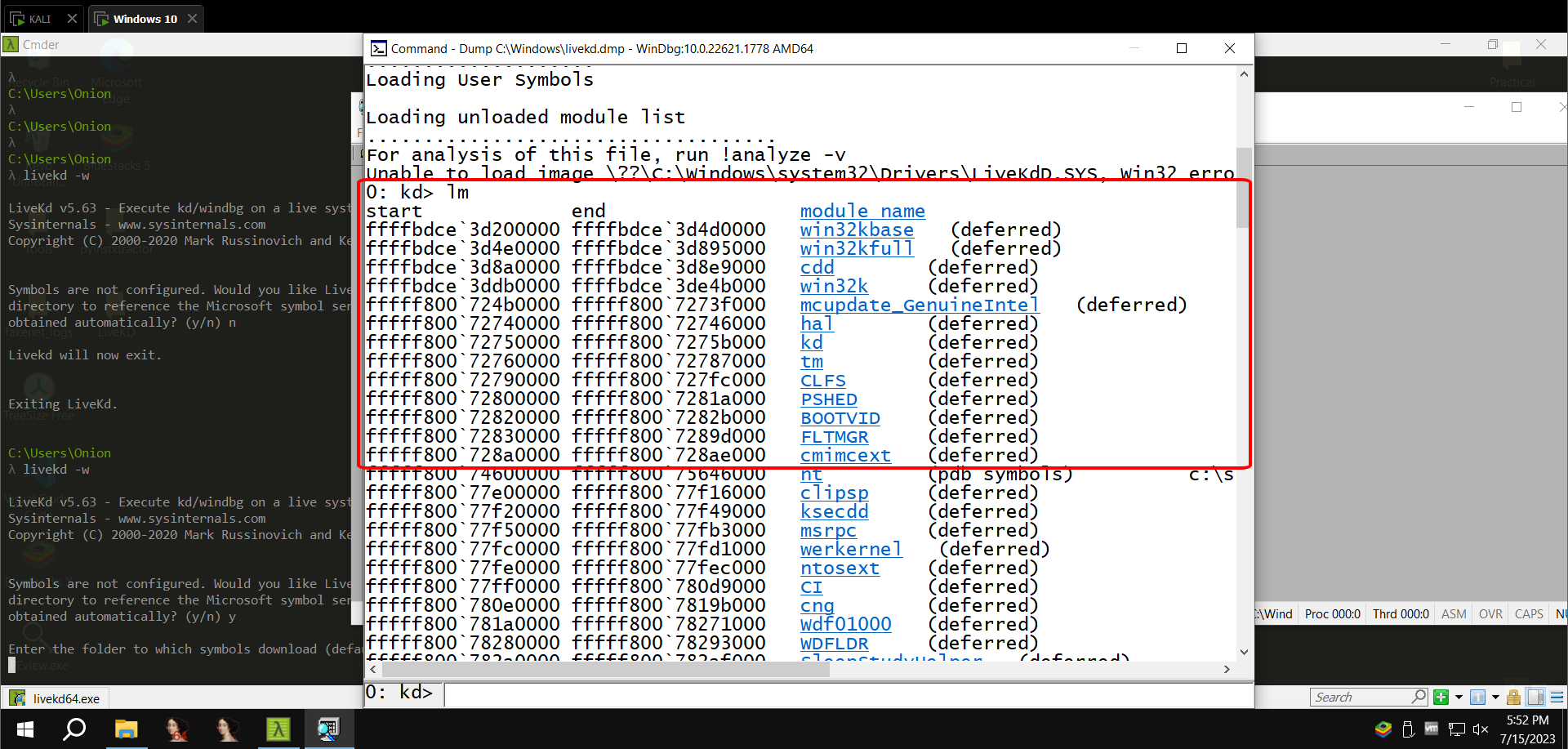
**What You Need**

A Windows 10 machine with Livekd working, as prepared in the previous project. This project should work on Win 7 or any later version, but I only tested it on. If you don’t have livvekd, download from this site:

* <https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/>

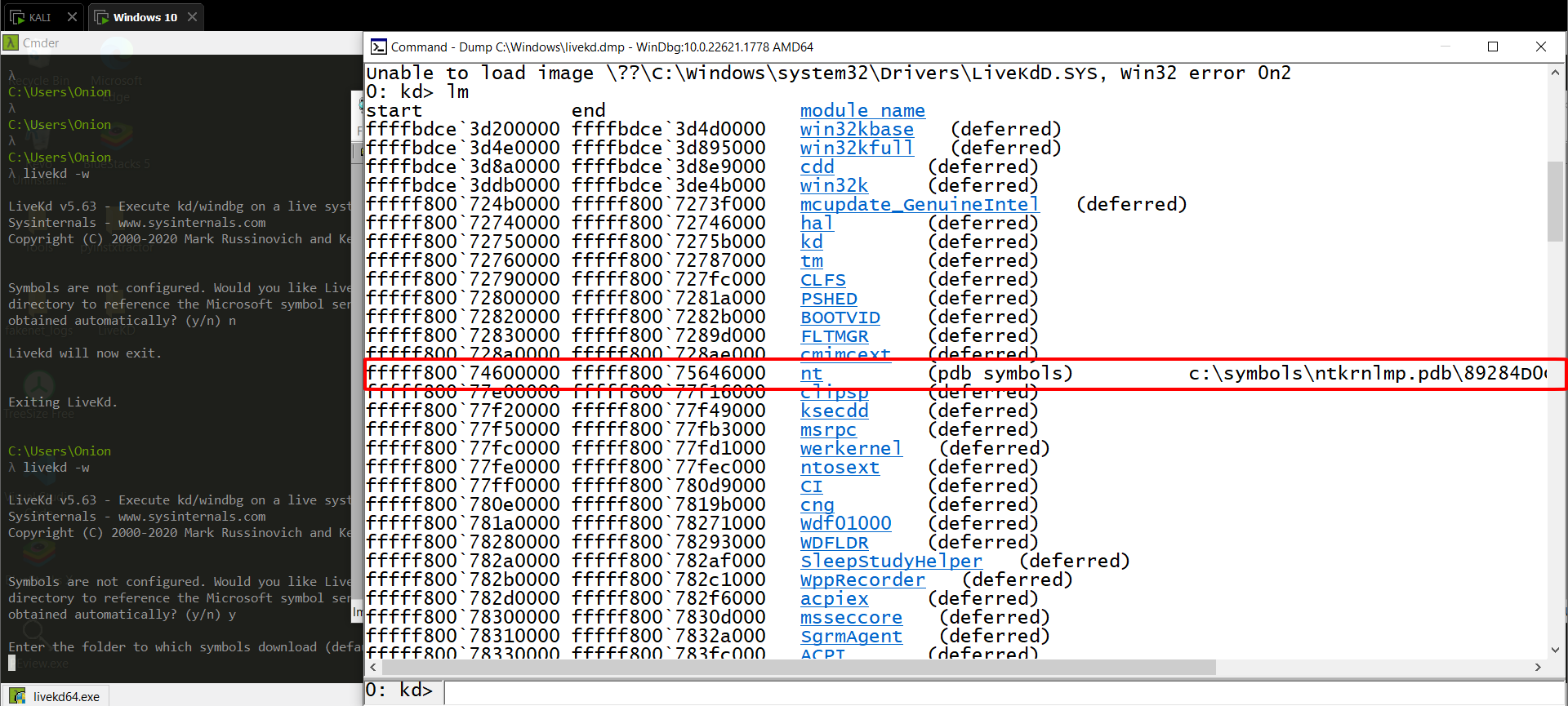
Listing Modules with lm:

At the bottom of the Command window, in the command bar, execute this command:



lm command will make us to see all the kernel process currently running on the computer

Scroll down to find the module named nt, as shown below. . It's easy to spot because it's one of the few modules that shows a Symbols path. Which is the main kernel module.



**Viewing Memory**

In WinDbg, execute this command:

* dd nt

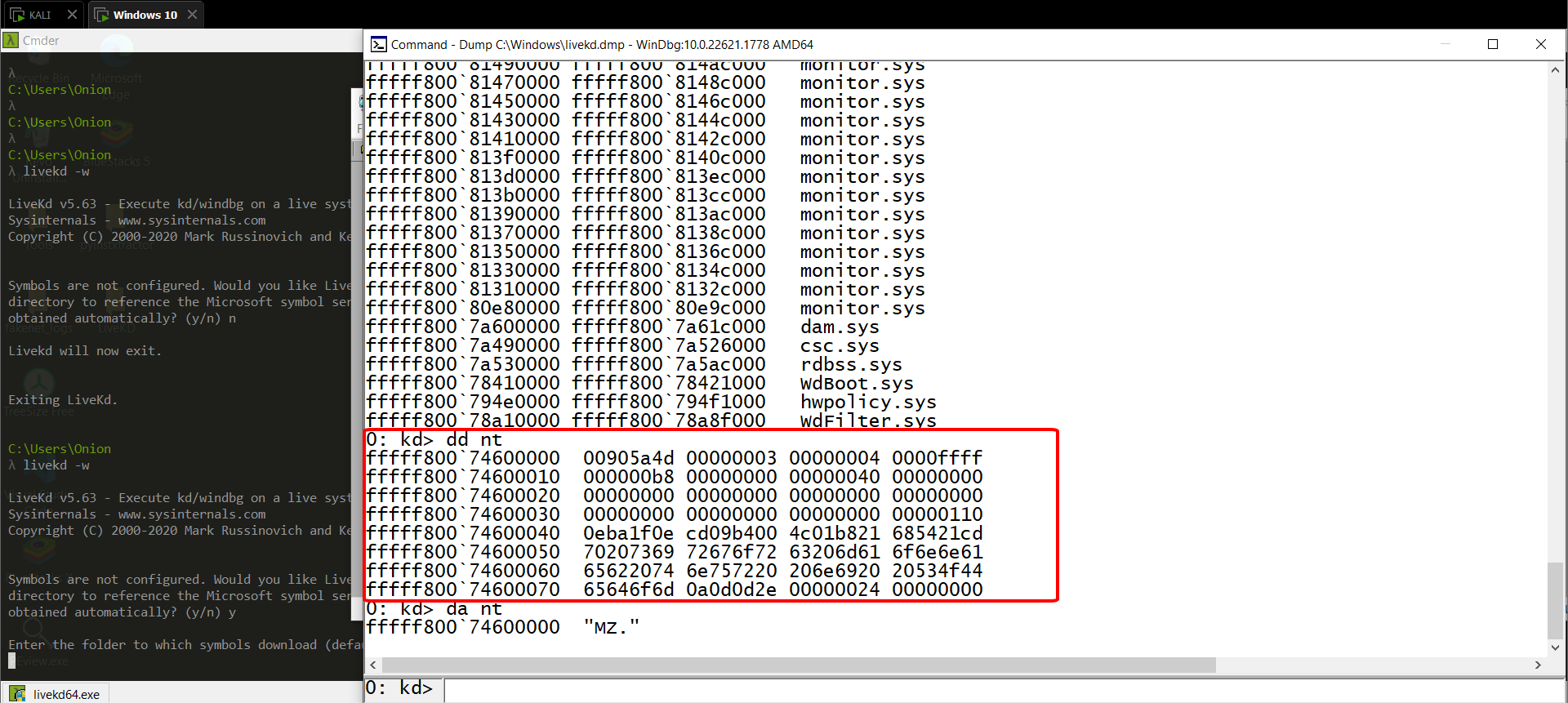
The dd nt command in WinDbg is used to display the contents of a memory region as a dump of NT structures. This can be useful for debugging kernel-mode code, as it allows you to see the internal structure of kernel objects and data structures.

The syntax for the dd nt command is as follows:

* dd nt [address] [length]

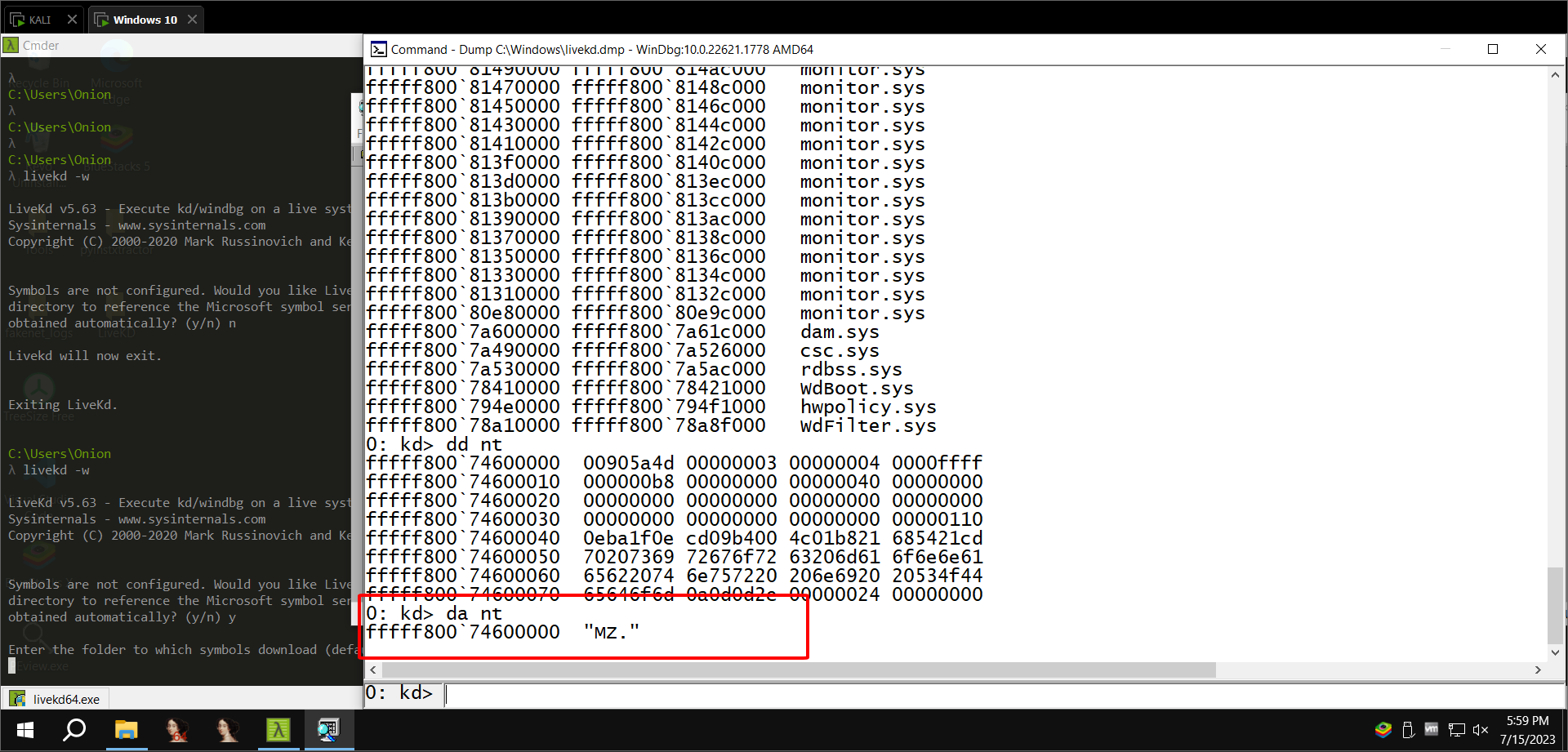
The address parameter specifies the starting address of the memory region to be dumped. The length parameter specifies the number of bytes to dump. If the length parameter is not specified, the entire memory region from the address parameter to the end of the region will be dumped.

You see the first several bytes of Ntoskrnl.exe, as shown below.



This may be more familiar in ASCII. In WinDbg, execute this command: da nt

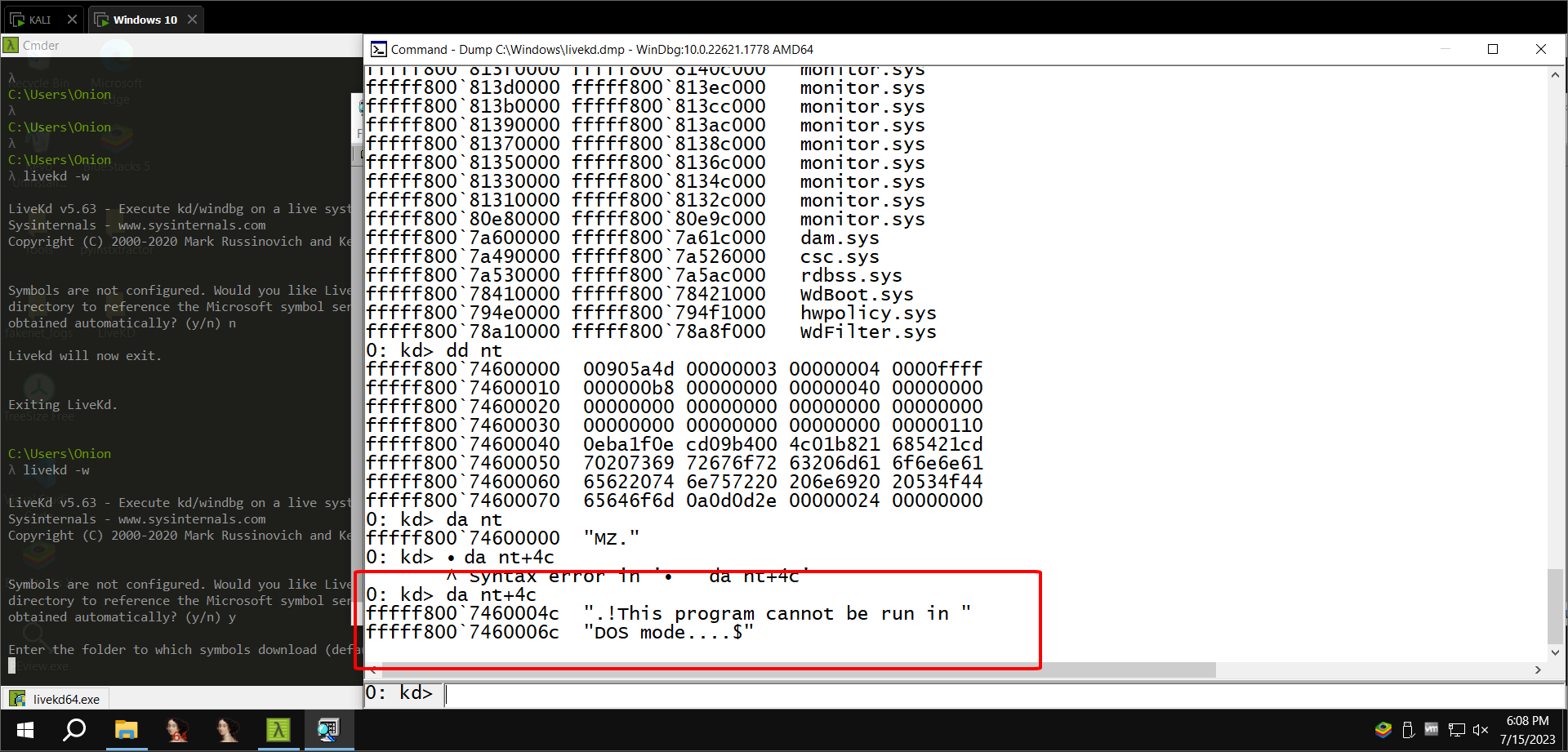
The **da nt** command in WinDbg is a shorthand for the **dd nt** command. It is used to display the contents of a memory region as a dump of NT structures. The syntax for the **da nt** command is the same as the syntax for the **dd nt** command.



In WinDbg, execute this command:

* da nt+4c

The **da nt+4c** command in WinDbg is a shorthand for the **dd nt+4c** command. It is used to display the contents of a memory region as a dump of NT structures, starting at a specific offset. The offset of 4c is typically used to dump the contents of the EIP register, which contains the address of the next instruction to be executed.

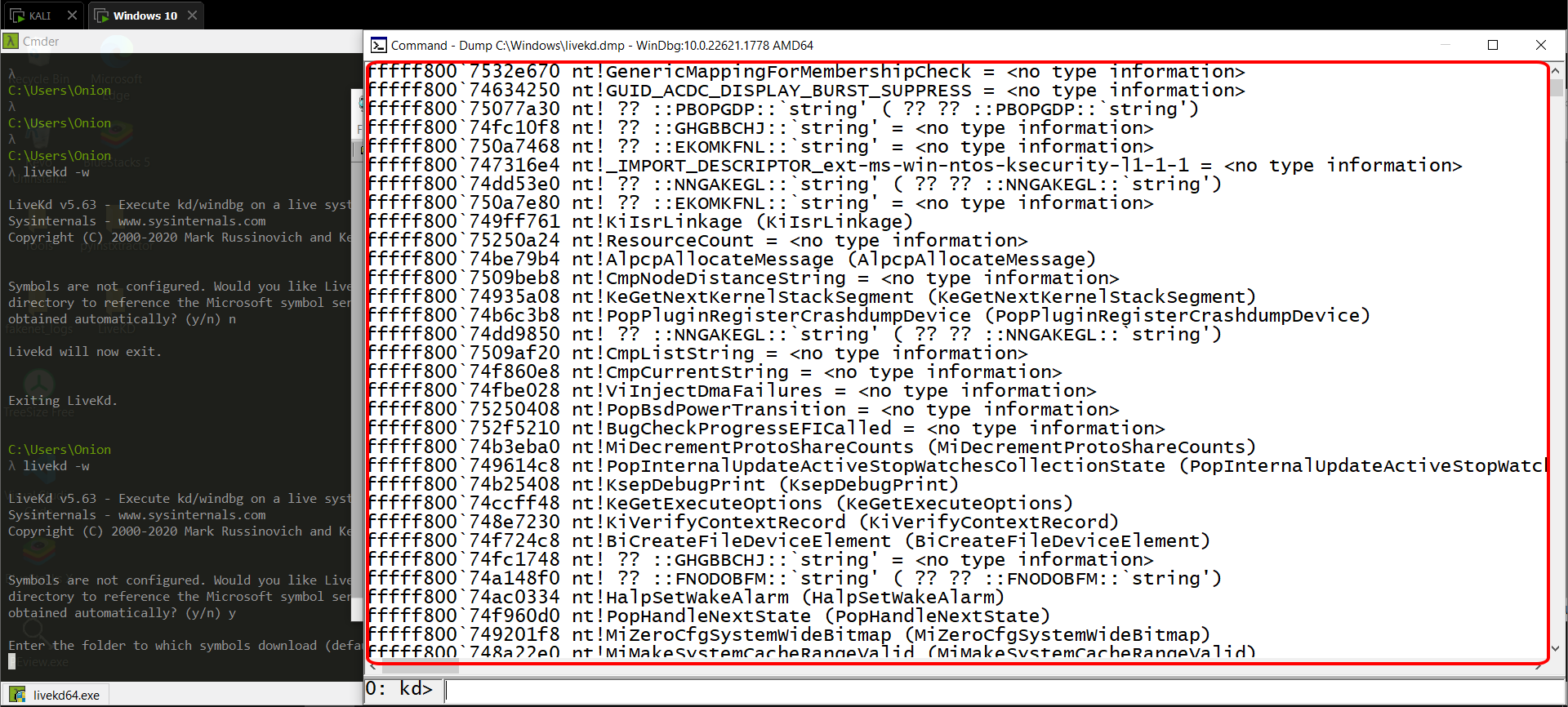


**Searching for Functions**

In WinDbg, execute this command:

* x nt!\*

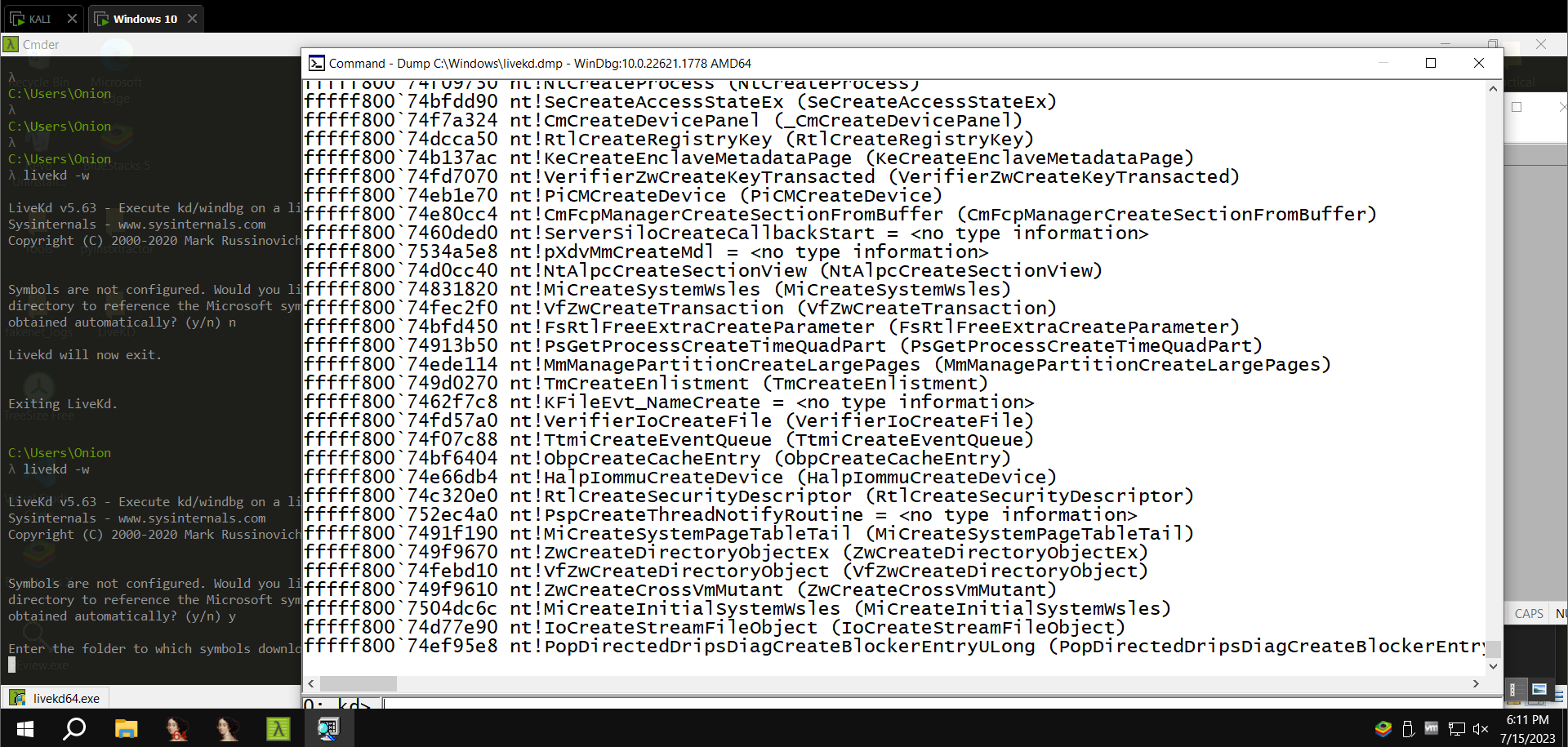
The **x nt!** command in WinDbg is used to display the contents of a memory region as a dump of NT structures, using the WinDbg symbol table to identify the structures and their fields. The **x nt!** command is a more powerful version of the **da nt** command, as it allows you to see the names of the structures and their fields.



In WinDbg, execute this command:

x nt!\*Create\*

This finds all the functions in Ntoskrnl that contain the word "Create".

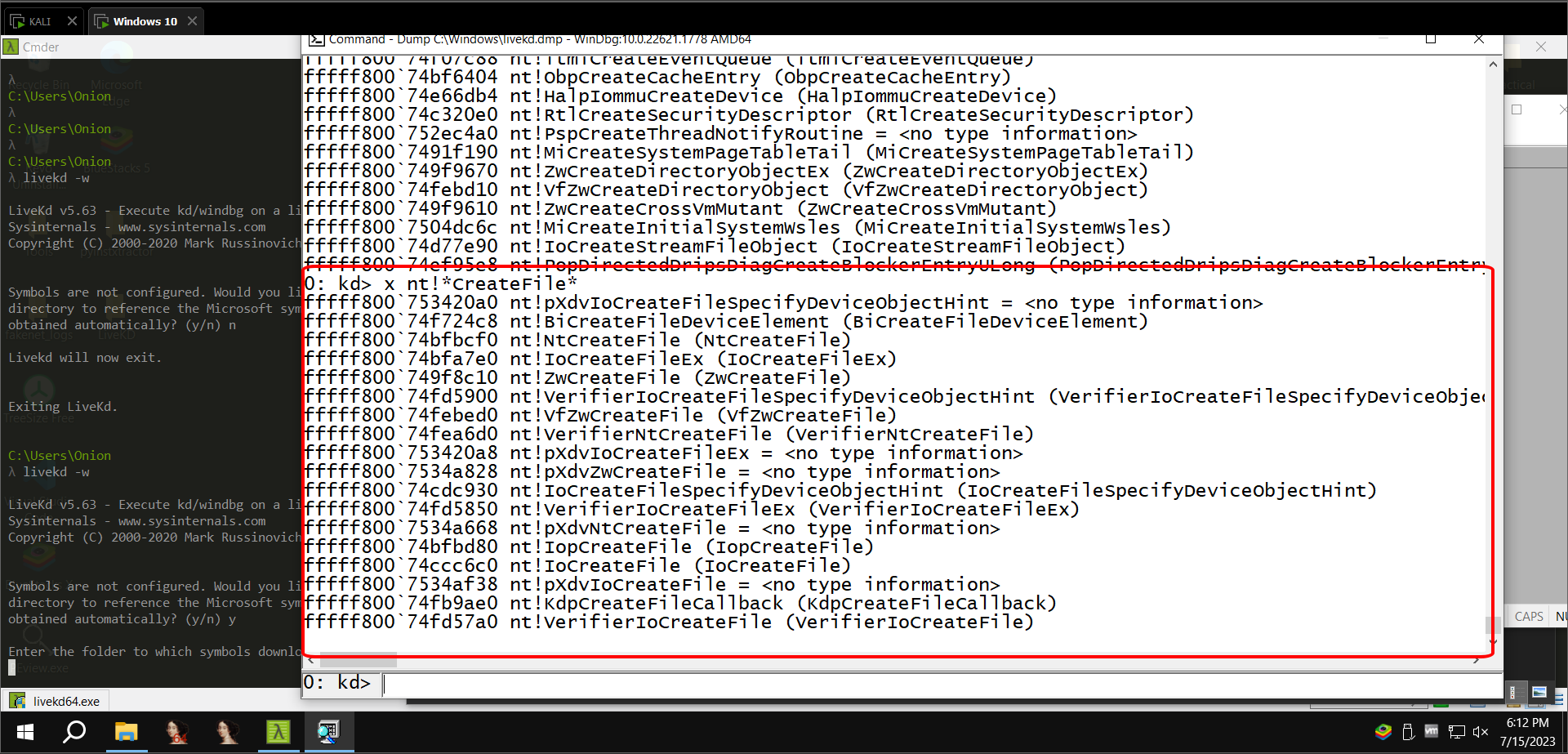


In WinDbg, execute this command:

* **x nt!\*CreateFile\***

This finds all the functions in Ntoskrnl that contain the word "CreateFile".

There are only about ten of those, including "nt!NtCreateFile", as shown below:

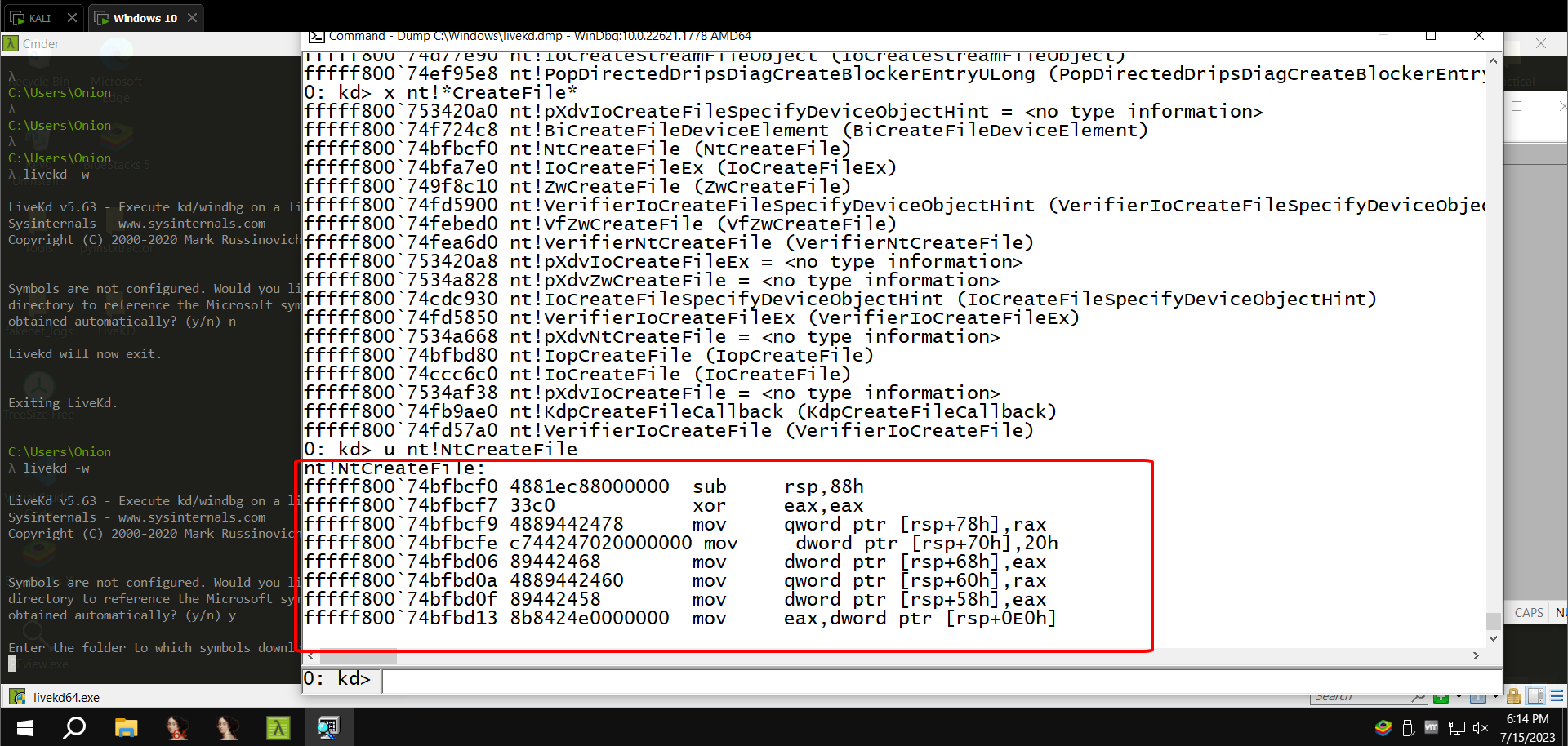


**Unassembling a Function**

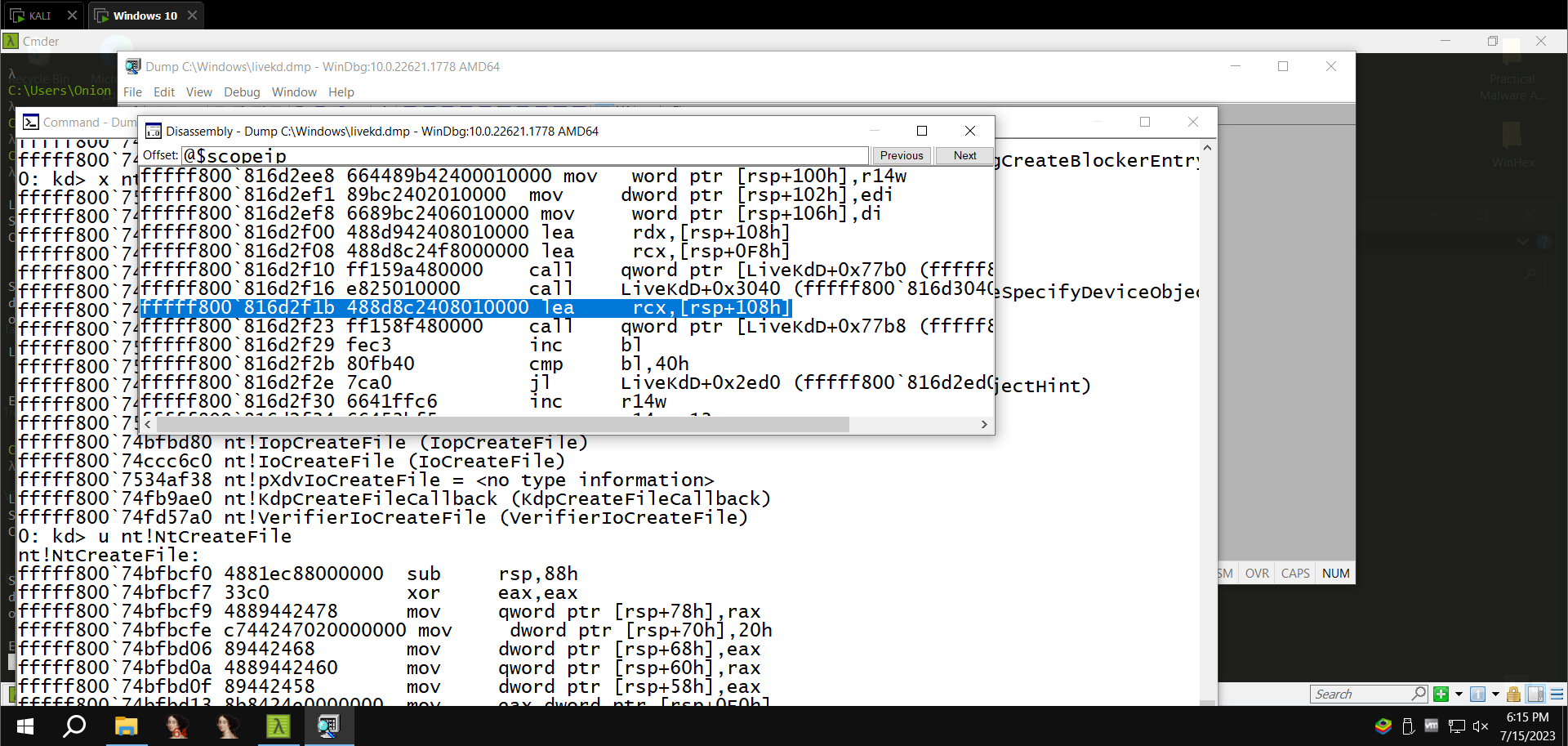
In WinDbg, execute this command:

* **u nt!NtCreateFile**

This shows the first few bytes of the function, disassembled, as shown below:



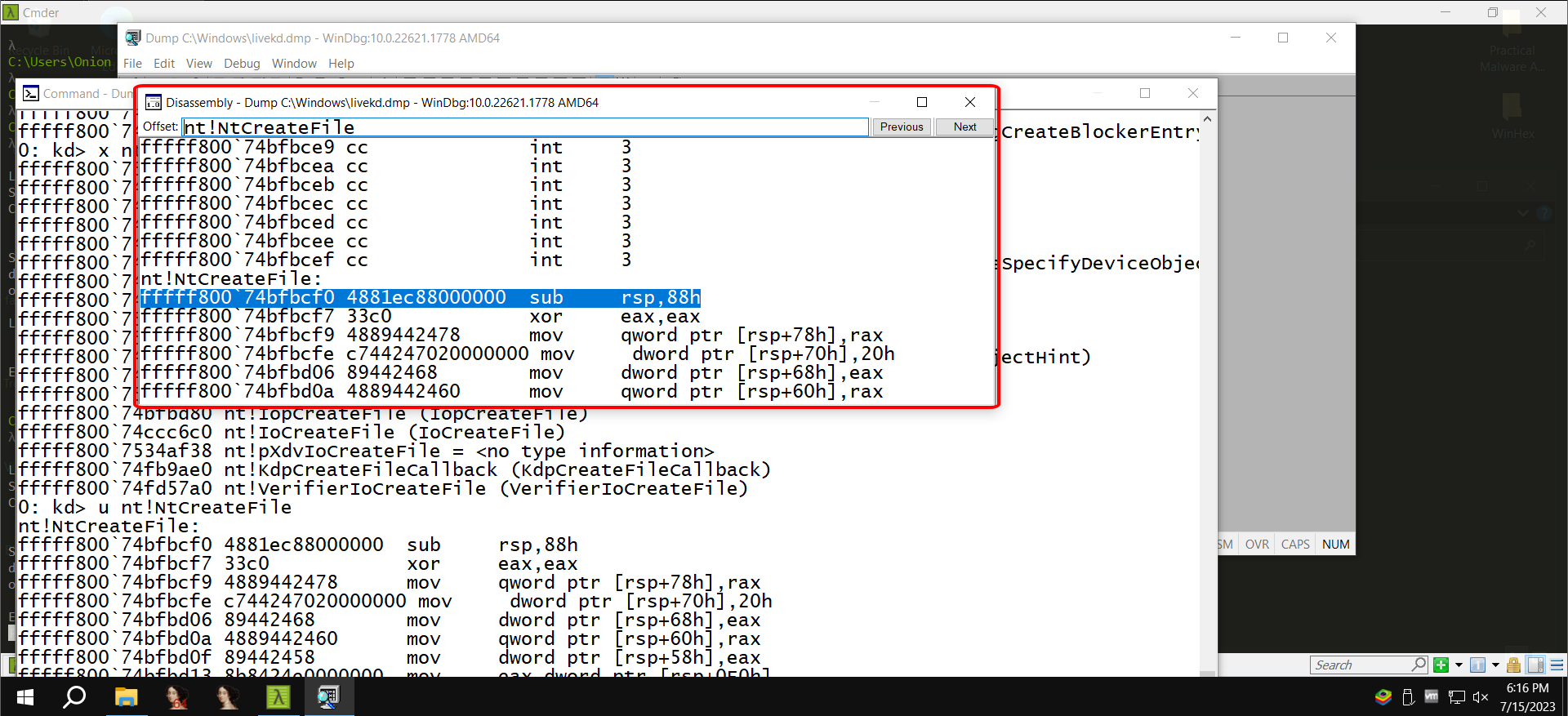
From the WinDbg menu bar, click View, Disassembly.



In the Offset bar at the top, enter

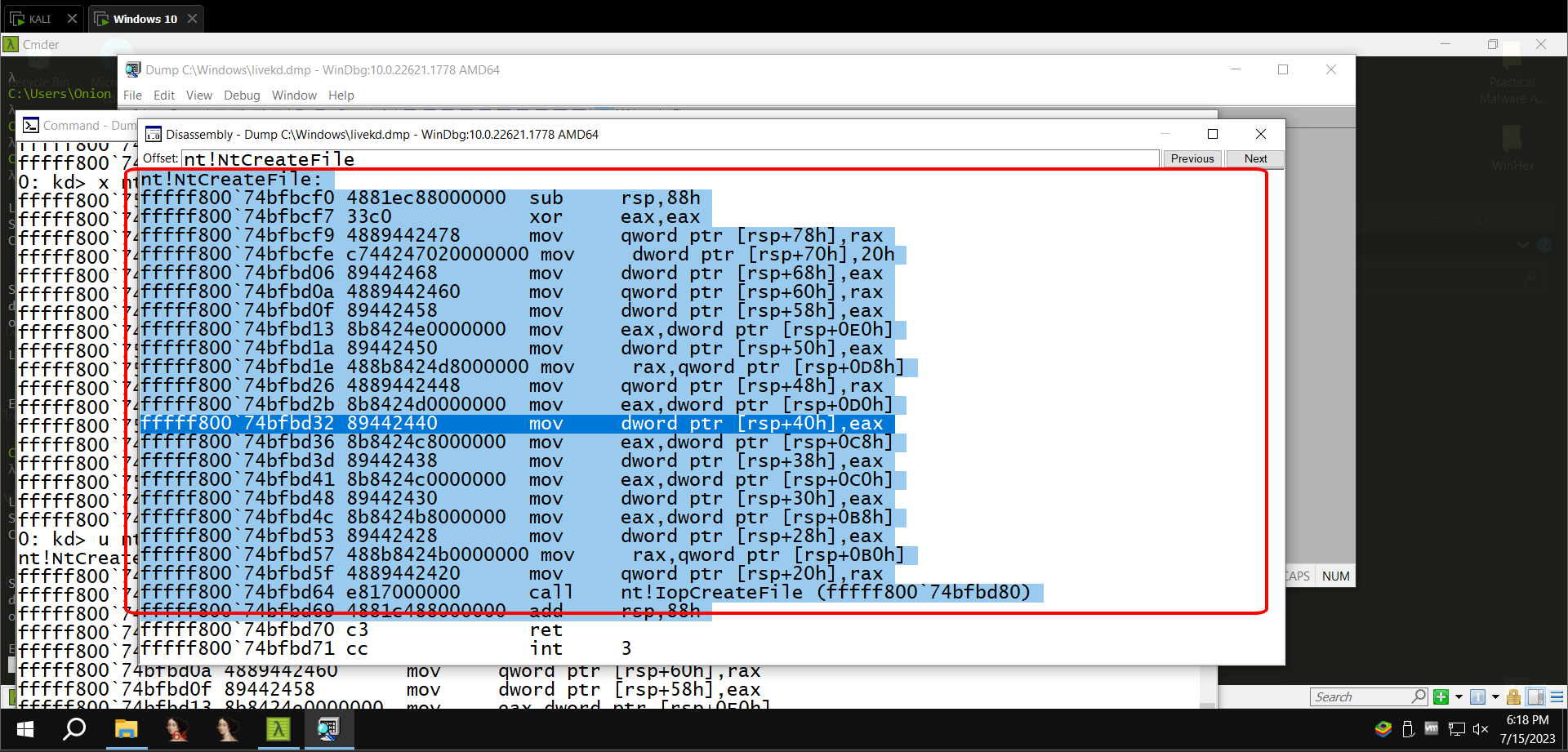
* **nt!NtCreateFile**

This shows the assembly code before and after the start of the NtCreateFile function, as shown below:



In the Offset bar at the top, enter

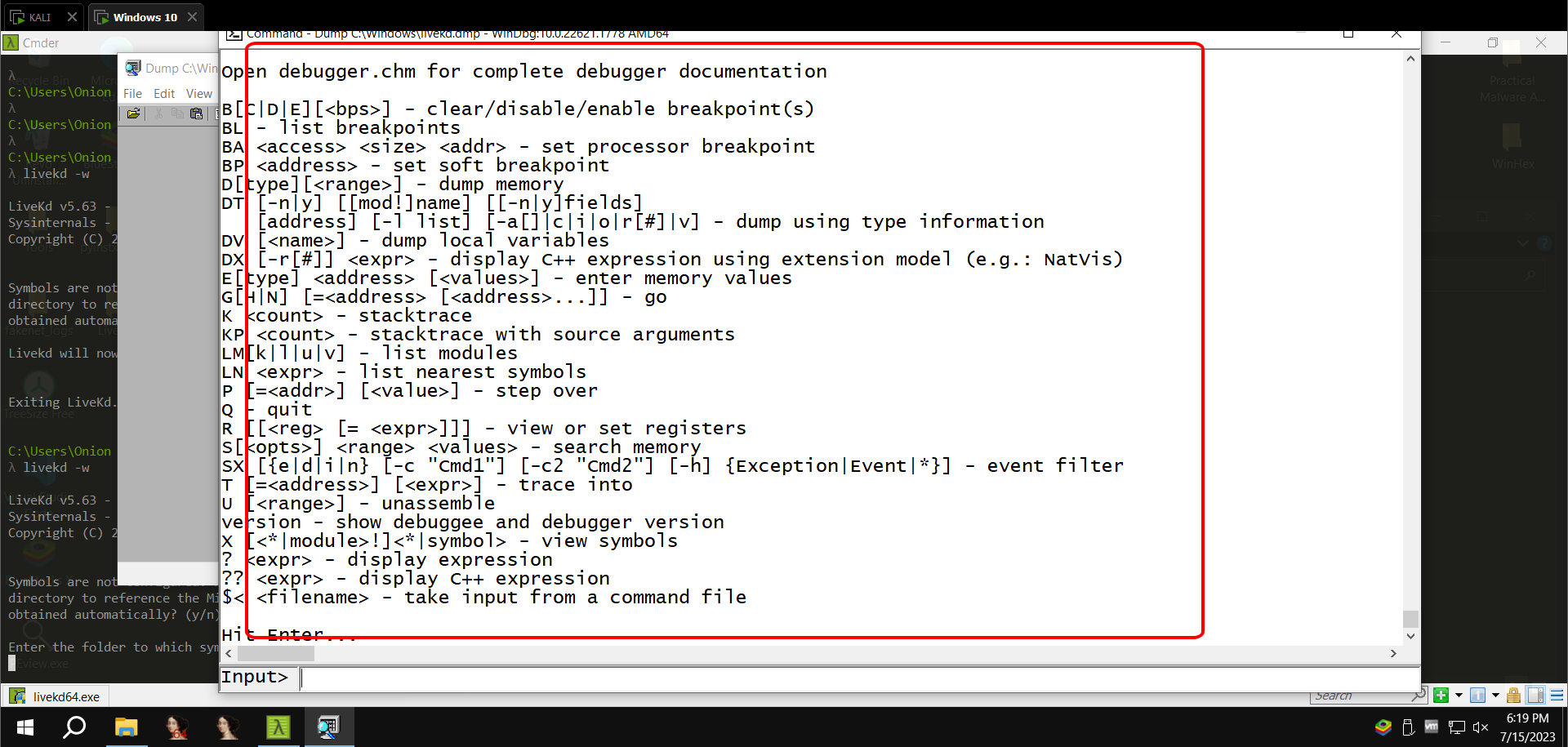
* **nt!NtCreateFile+16**



**Online Help**

In WinDbg, execute this command:

* ?

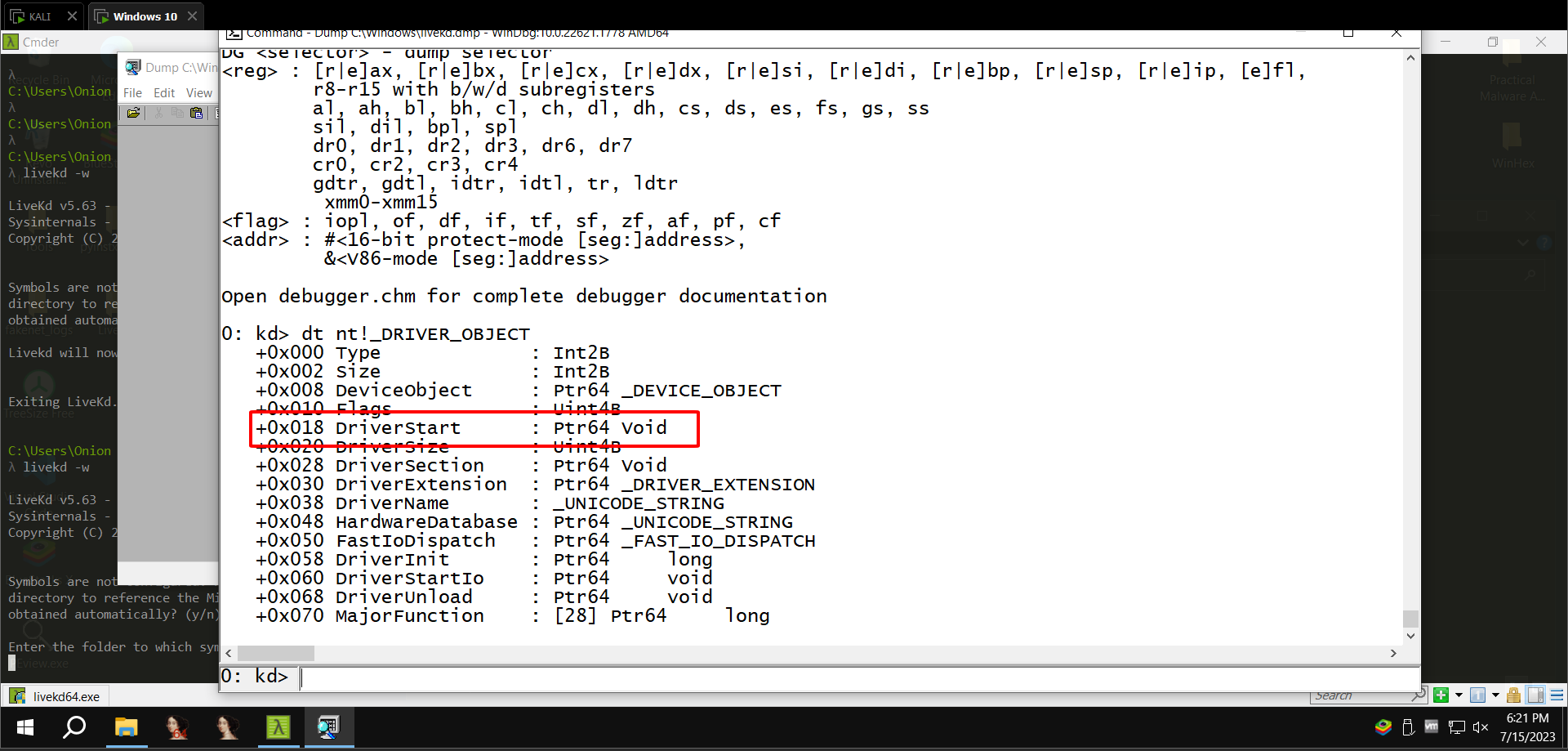


**Viewing Type Information for a Structure**

In WinDbg, execute this command:

* **dt nt!\_DRIVER\_OBJECT**

The **dt nt!\_DRIVER\_OBJECT** command in WinDbg is used to display the contents of a **DRIVER\_OBJECT** structure. The **DRIVER\_OBJECT** structure is a kernel-mode data structure that contains information about a driver.



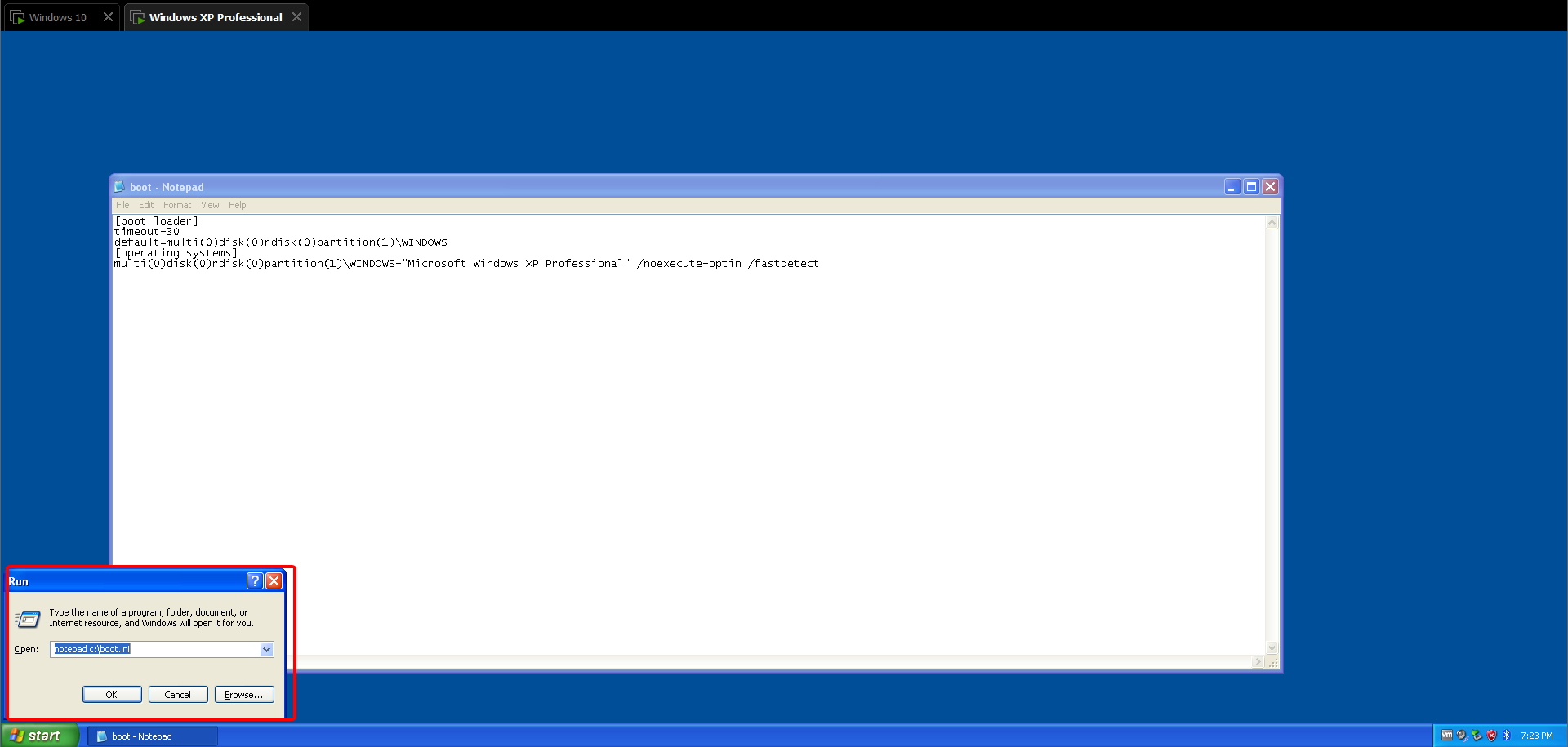
**Adding a Boot Menu Item to the TARGET machine**

Start the Windows virtual machine.

Click Start, Run.

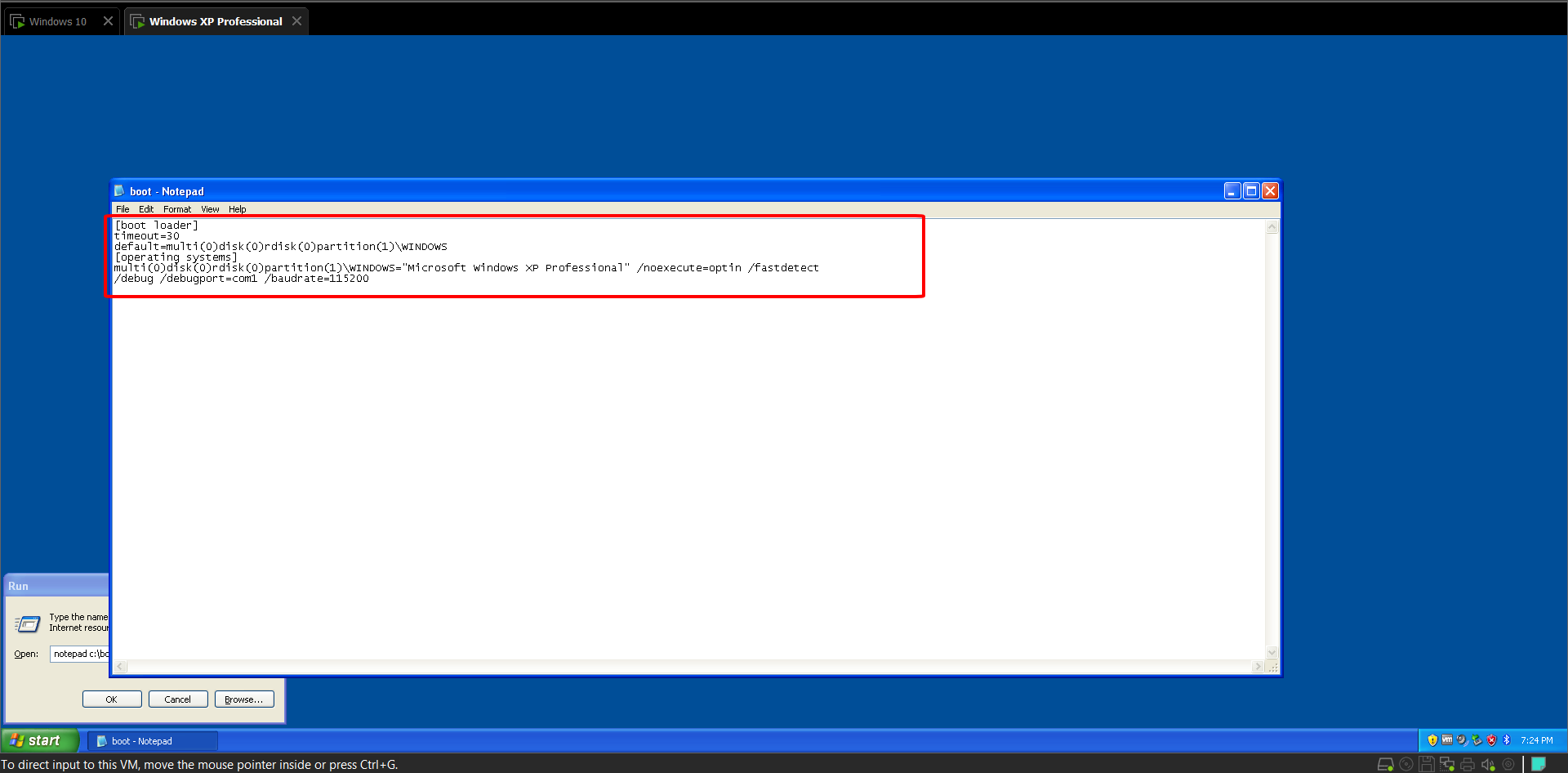
In the Run box, execute this command:

* **notepad c:\boot.ini**



In Notepad, copy the existing boot line, paste it at the end of the file, and add these switches to the end of the line, as shown below:

* **/debug /debugport=com1 /baudrate=115200**



Save the file.

**Adding a Virtual Serial Adapter**

**Power off the TARGET virtual machine**

On the WINDBG machine, start VMware Player.

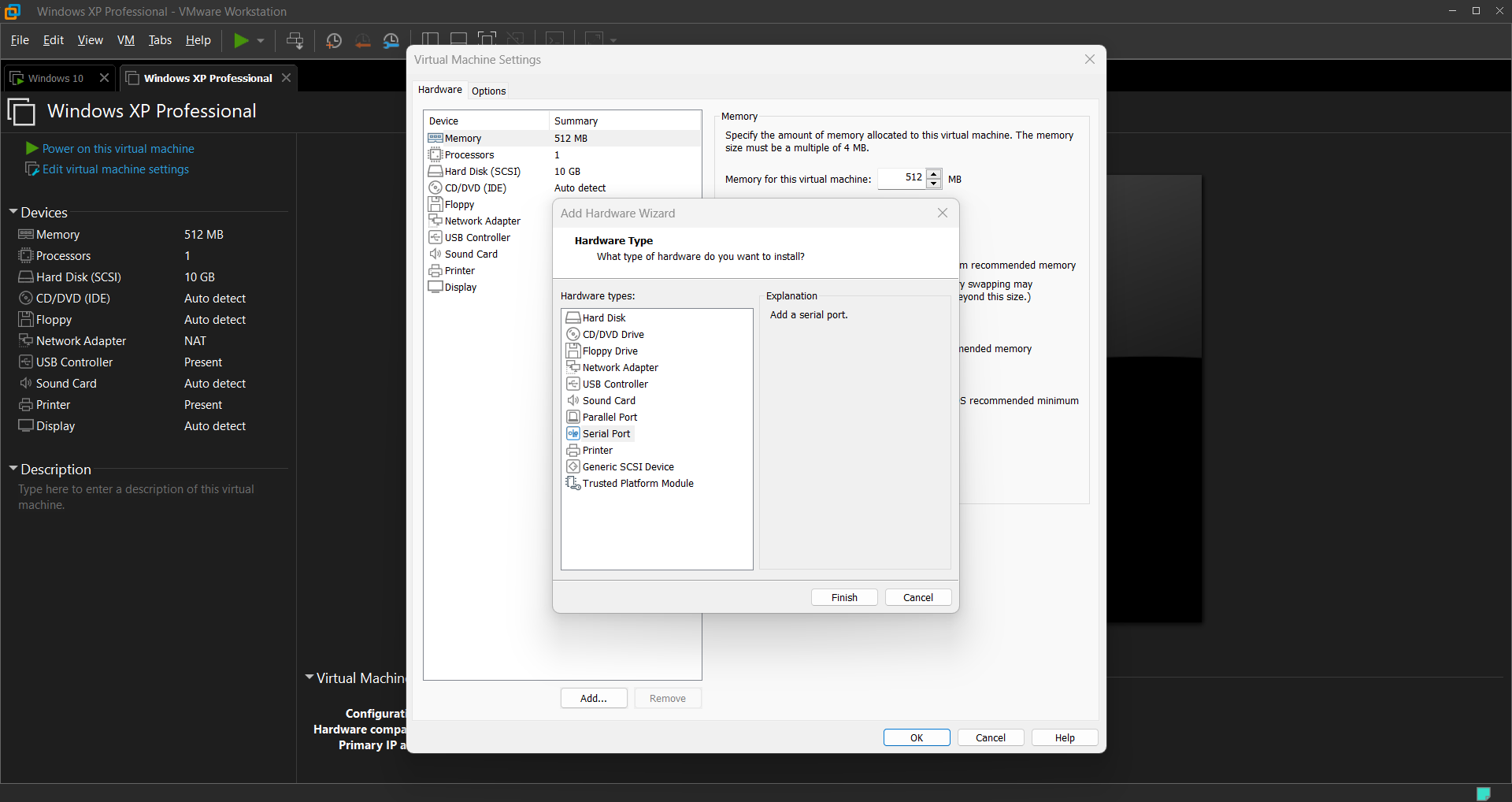
In the left pane of VMware Player, click your TARGET machine.

At the lower right of VMware Player, click " **Edit virtual machine settings”**

In the left side of the "Virtual Machine Settings" box, click the

**Add...** button

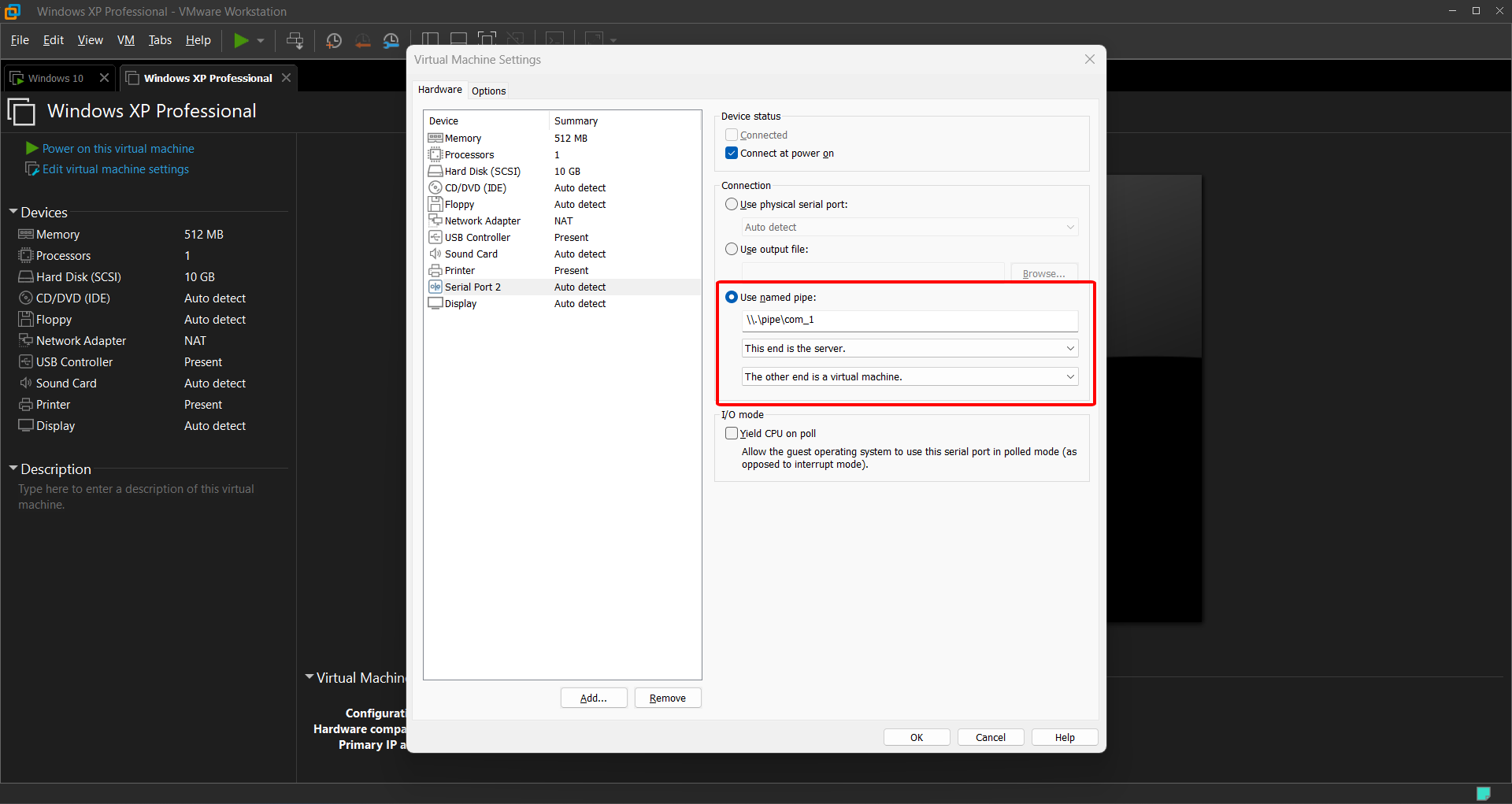
In the "Hardware Type" box, select " **Serial Port”** ", as shown below.



Click Next

In the "Serial Port Type" box, click "**Output to named pipe**”

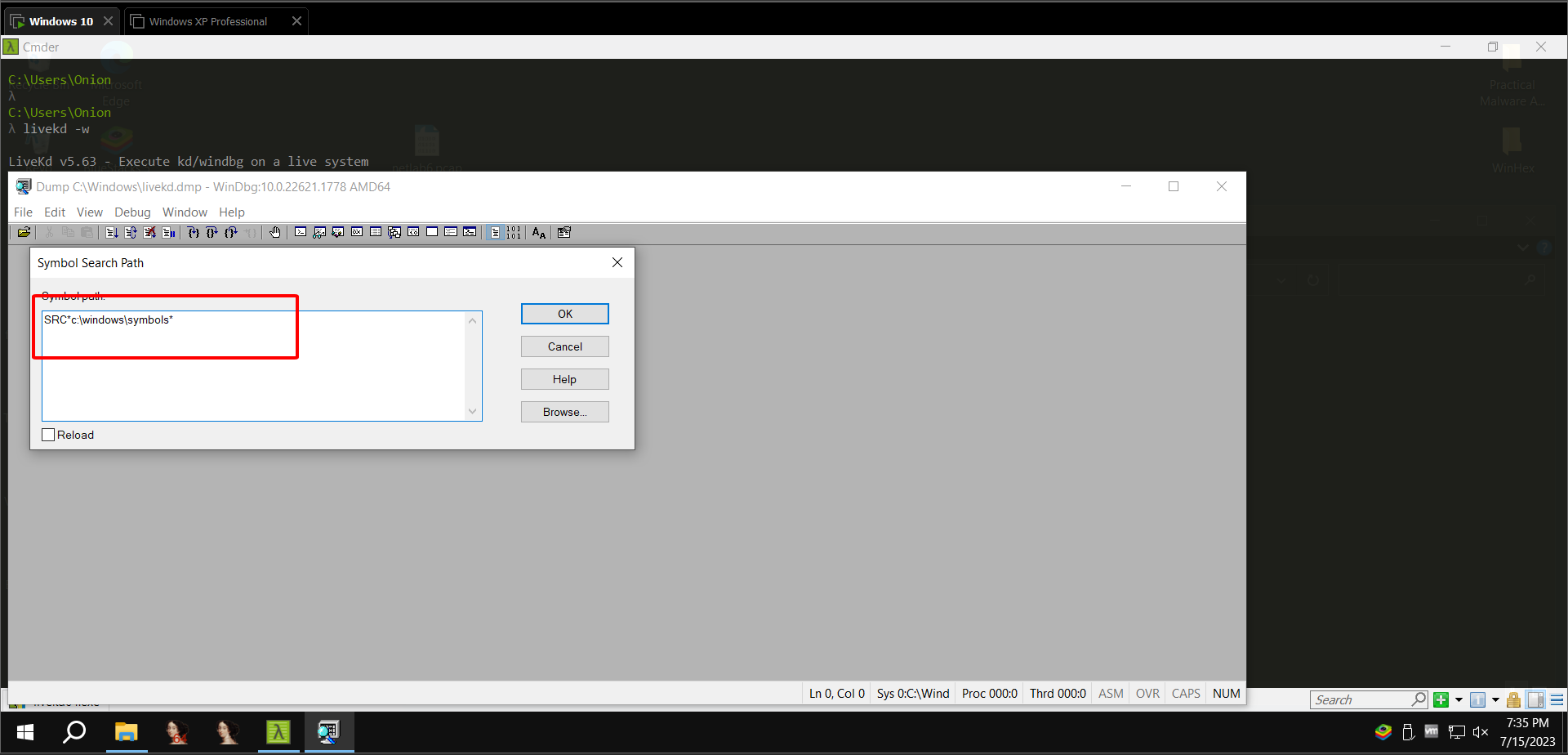
* **\\.\pipe\com\_1**



**Configuring Symbols in WinDbg**

In WinDbg, click **File, Symbol File Path.** ". Enter this line, as shown below:

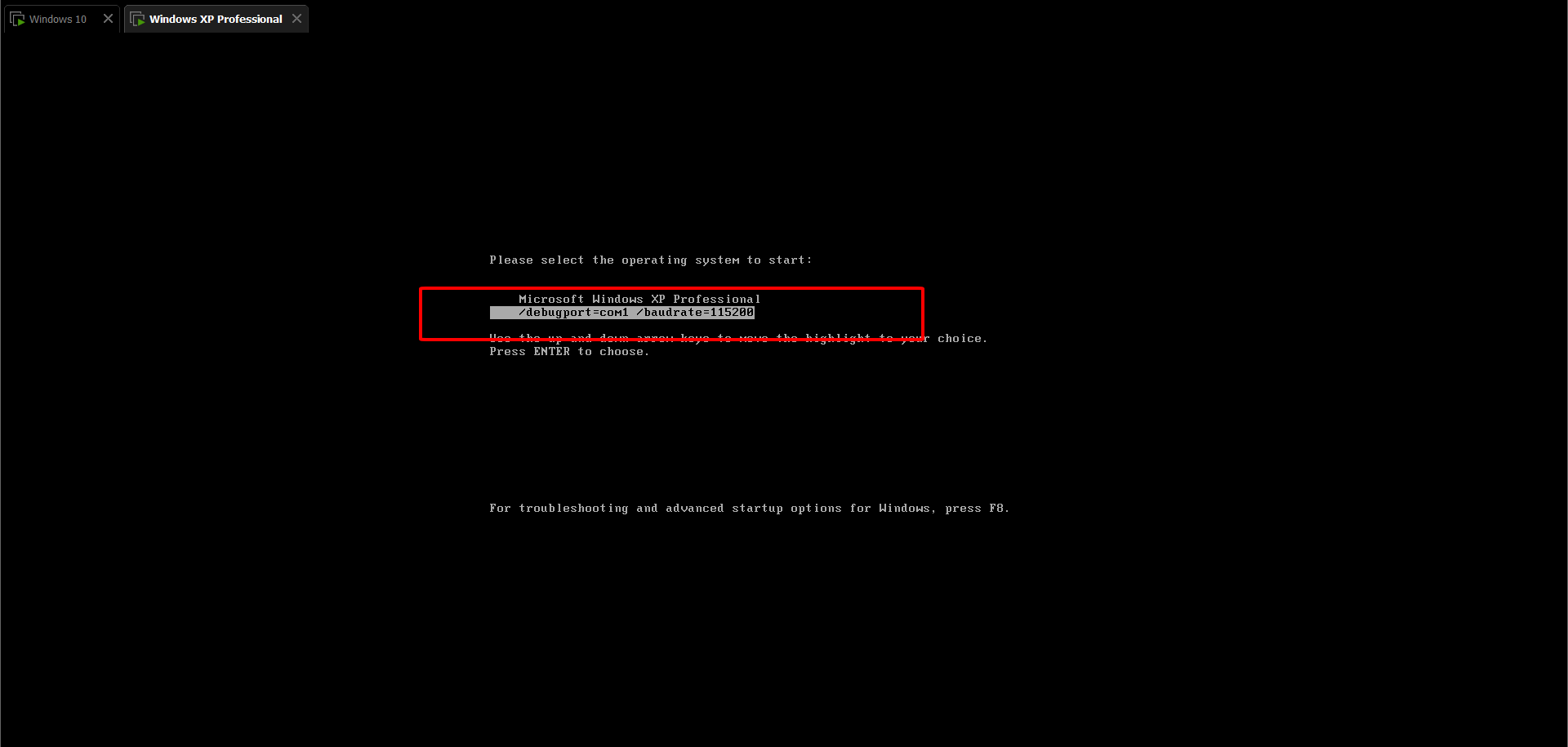
* **SRC\*c:\windows\symbols\***



**Starting the TARGET machine**

Start the TARGET virtual machine.

When you should see two boot-menu options, choose the second one, "Microsoft Windows XP Professional with debugger enabled", as shown below.



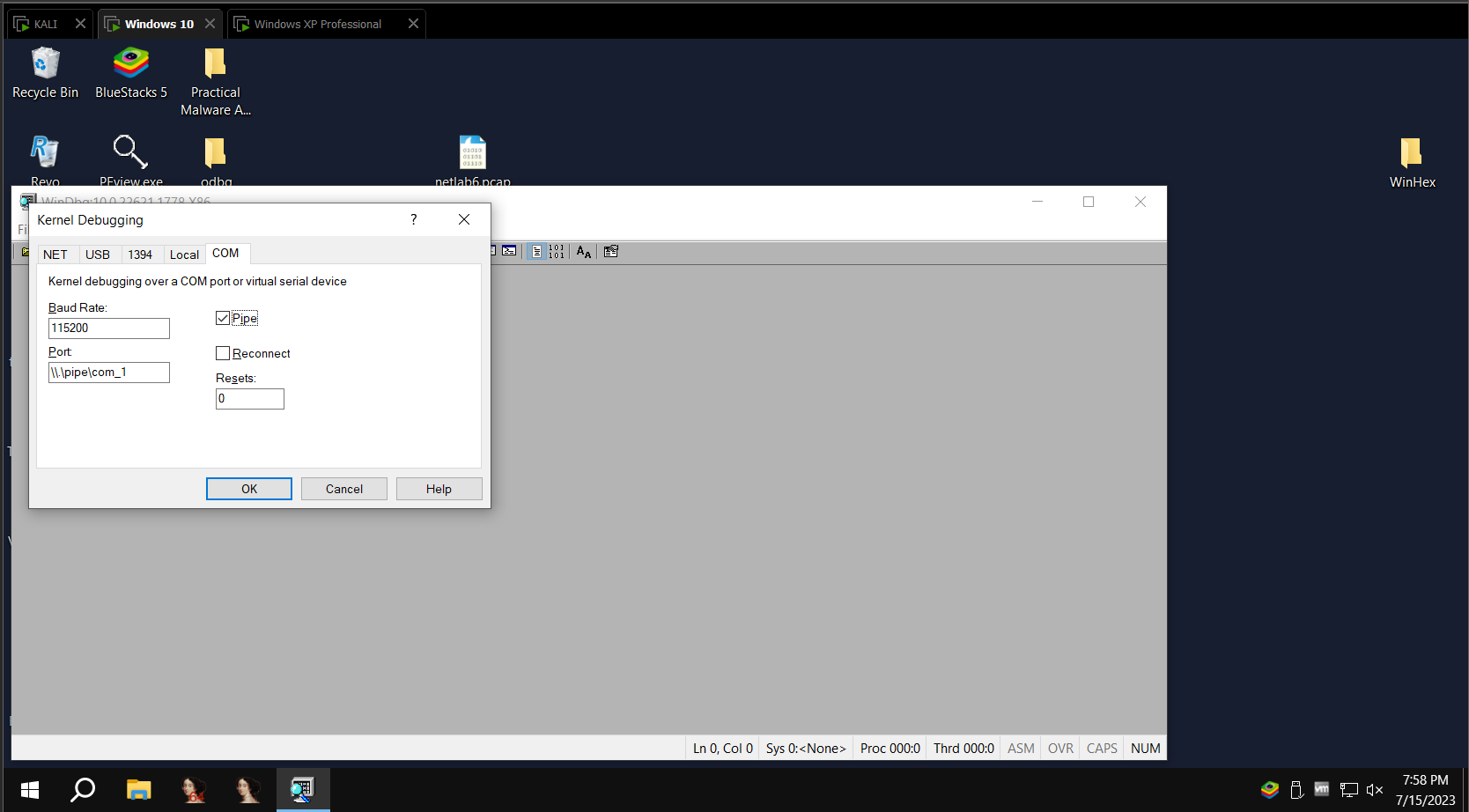
**Starting Kernel Debugging**

In WinDbg, click **File, “Kernel Debug”**

In the "Kernel Debugging" box, click the **COM** tab

Change the Port to

* **\\.\pipe\com\_1**

and check the Pipe box, as shown below. Then click OK

Your WINDBG machine should now show the message " **Connected to Windows XP”**