

LAB 16+17

Thầy Mai Hoàng Đình
Trường đại học FPT

Người thực hiện

Đặng Hoàng Nguyên

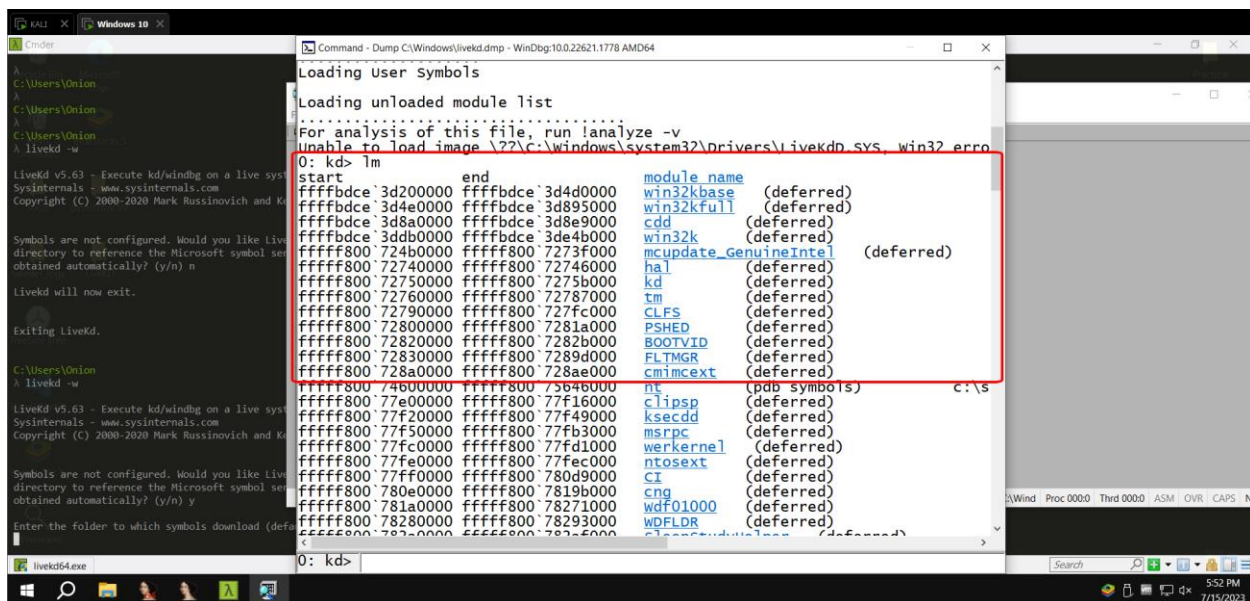
What You Need

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:



```
0: kd> lm
start      end                      module_name
ffffbdce`3d200000 fffffbdce`3d4d0000 win32kbase (deferred)
ffffbdce`3d4e0000 fffffbdce`3d895000 win32kfull (deferred)
ffffbdce`3d8a0000 fffffbdce`3d8e9000 cdd (deferred)
ffffbdce`3ddb0000 fffffbdce`3de4b000 win32k (deferred)
fffff800`72740000 fffff800`7273f000 mcupdate_GenuineIntel (deferred)
fffff800`72740000 fffff800`72746000 hal (deferred)
fffff800`72750000 fffff800`7275b000 kd (deferred)
fffff800`72760000 fffff800`72787000 tm (deferred)
fffff800`72790000 fffff800`727fc000 CLFS (deferred)
fffff800`72800000 fffff800`7281a000 PSHEd (deferred)
fffff800`72820000 fffff800`7282b000 BOOTVID (deferred)
fffff800`72830000 fffff800`7289d000 FLTMRG (deferred)
fffff800`728a0000 fffff800`728ae000 cmimcext (deferred)
fffff800`728b0000 fffff800`728b6000 nt (pdb symbols)
fffff800`728c0000 fffff800`728c6000 clipsp (deferred)
fffff800`728d0000 fffff800`728d6000 ksecdd (deferred)
fffff800`728e0000 fffff800`728e6000 msrpc (deferred)
fffff800`728f0000 fffff800`728f6000 werkernel (deferred)
fffff800`72900000 fffff800`72906000 ntosex (deferred)
fffff800`72910000 fffff800`72916000 CI (deferred)
fffff800`72920000 fffff800`72926000 cng (deferred)
fffff800`72930000 fffff800`72936000 wdf01000 (deferred)
fffff800`72940000 fffff800`72946000 WDFLDR (deferred)
```

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.

```
Command - Dump C:\Windows\livekd.dmp - WinDbg:10.0.22621.1778 AMD64
Unable to load image \??\C:\Windows\system32\Drivers\Livekd.sys, win32 error 0n2
0: kd> lm
start      end             module_name
fffffbdce`3d200000 fffffbdce`3d4d0000 win32kbase (deferred)
fffffbdce`3d4e0000 fffffbdce`3d895000 win32kfull (deferred)
fffffbdce`3d8a0000 fffffbdce`3d8e9000 cdd (deferred)
fffffbdce`3ddb0000 fffffbdce`3de4b000 win32k (deferred)
fffff800`724b0000 fffff800`7273f000 mcupdate_GenuineIntel (deferred)
fffff800`72740000 fffff800`72746000 hal (deferred)
fffff800`72750000 fffff800`7275b000 kd (deferred)
fffff800`72760000 fffff800`72787000 km (deferred)
fffff800`72790000 fffff800`727fc000 CLFS (deferred)
fffff800`72800000 fffff800`7281a000 PSHEd (deferred)
fffff800`72820000 fffff800`7282b000 BOOTVID (deferred)
fffff800`72830000 fffff800`7289d000 FLTMRGR (deferred)
fffff800`728a0000 fffff800`728ae000 cmimcext (deferred)
fffff800`74600000 fffff800`75646000 nt (pdb symbols) c:\symbols\ntkrnlmp.pdb\89284d00
fffff800`77e00000 fffff800`77f16000 clipsp (deferred)
fffff800`77f20000 fffff800`77f49000 ksecdd (deferred)
fffff800`77f50000 fffff800`77fb3000 msrpc (deferred)
fffff800`77fc0000 fffff800`77fd1000 werkernel (deferred)
fffff800`77fe0000 fffff800`77fec000 ntosex (deferred)
fffff800`77ff0000 fffff800`780d9000 CI (deferred)
fffff800`780e0000 fffff800`7819b000 cng (deferred)
fffff800`781a0000 fffff800`78271000 wdf01000 (deferred)
fffff800`78280000 fffff800`78293000 WDFLDR (deferred)
fffff800`782a0000 fffff800`782af000 SleepStudyHelper (deferred)
fffff800`782b0000 fffff800`782c1000 WppRecorder (deferred)
fffff800`782d0000 fffff800`782f6000 acpiex (deferred)
fffff800`78300000 fffff800`7830d000 msseccore (deferred)
fffff800`78310000 fffff800`7832a000 SgrmAgent (deferred)
fffff800`78330000 fffff800`783fc000 ACPT (deferred)
0: kd>
```

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.

```
0: kd> dd nt
fffff800`74600000 00905a4d 00000003 00000004 0000ffff
fffff800`74600010 000000b8 00000000 00000040 00000000
fffff800`74600020 00000000 00000000 00000000 00000000
fffff800`74600030 00000000 00000000 00000000 00000110
fffff800`74600040 0eba1f0e cd09b400 4c01b821 685421cd
fffff800`74600050 70207369 72676f72 63206d61 6f6e6e61
fffff800`74600060 65622074 6e757220 206e6920 20534f44
fffff800`74600070 65646f6d 0a0d0d2e 00000024 00000000
0: kd> da nt
fffff800`74600000 "MZ."
```

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.

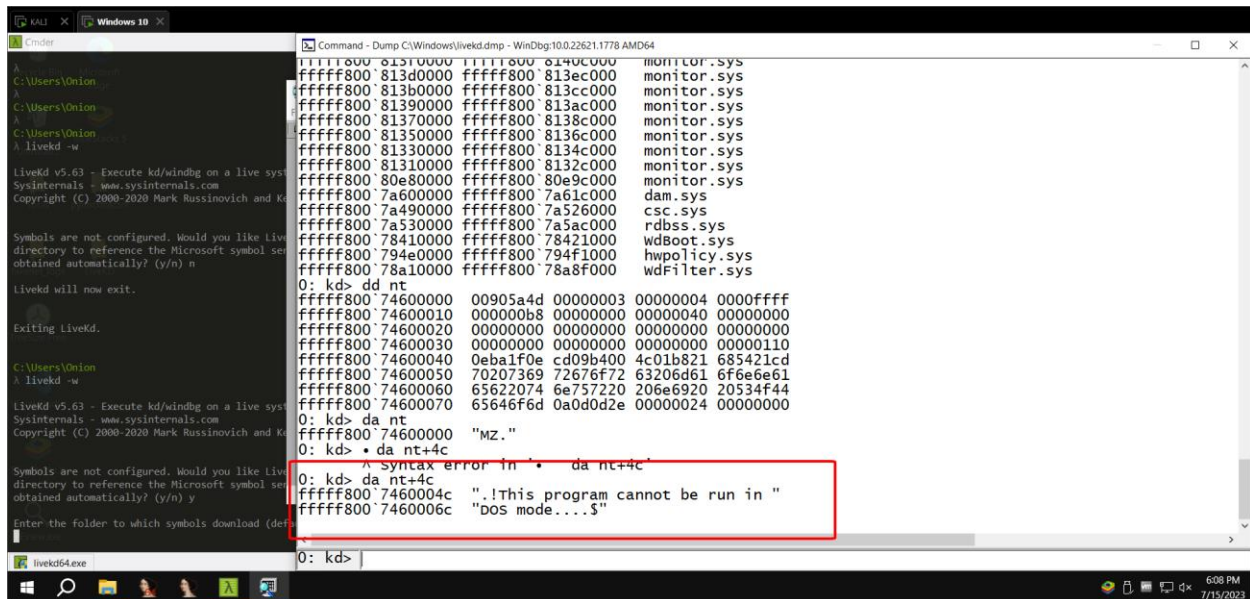
```
0: kd> dd nt
fffff800`74600000 00905a4d 00000003 00000004 0000ffff
fffff800`74600010 000000b8 00000000 00000040 00000000
fffff800`74600020 00000000 00000000 00000000 00000000
fffff800`74600030 00000000 00000000 00000000 00000110
fffff800`74600040 0eba1f0e cd09b400 4c01b821 685421cd
fffff800`74600050 70207369 72676f72 63206d61 6f6e6e61
fffff800`74600060 65622074 6e757220 206e6920 20534f44
fffff800`74600070 65646f6d 0a0d0d2e 00000024 00000000
0: kd> da nt
fffff800`74600000 "MZ."
```

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.



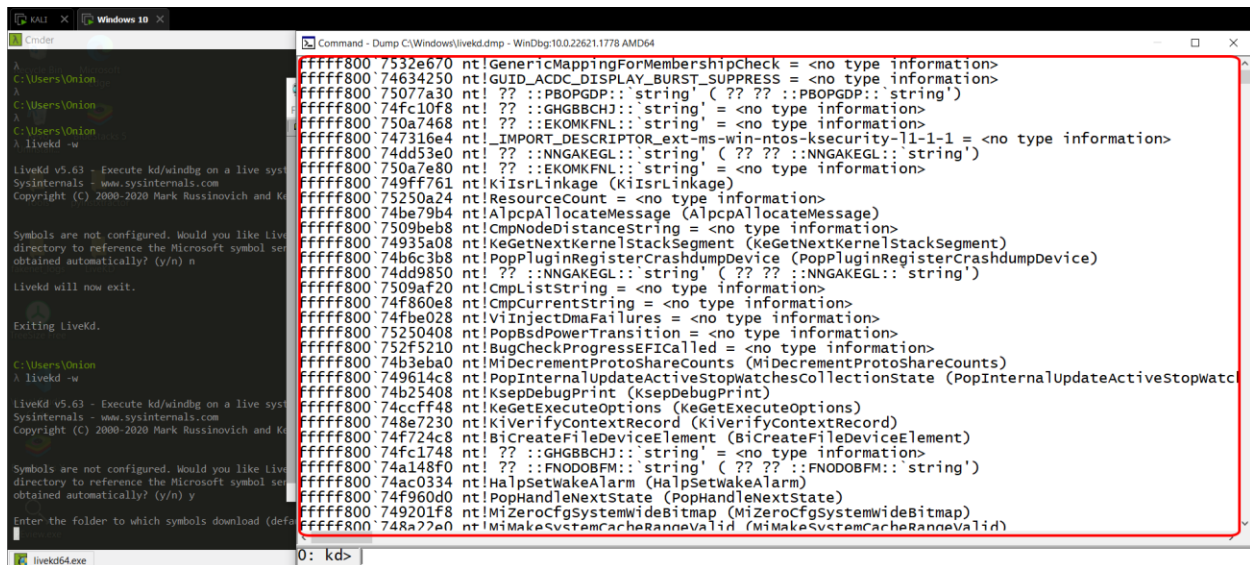
```
0: kd> dd nt
fffff800`74600000 00905a4d 00000003 00000004 0000ffff
fffff800`74600010 000000b8 00000000 00000040 00000000
fffff800`74600020 00000000 00000000 00000000 00000000
fffff800`74600030 00000000 00000000 00000000 00000110
fffff800`74600040 0eba1f0e cd09b400 4c01b821 685421cd
fffff800`74600050 70207369 72676f72 63206d61 6f6e6e61
fffff800`74600060 65622074 6e757220 206e6920 20534f44
fffff800`74600070 65646f6d 0a0d0d2e 00000024 00000000
0: kd> da nt
fffff800`74600000 "MZ."
0: kd> da nt+4c
0: kd> * syntax error in 'da nt+4c'
fffff800`7460004c ".!This program cannot be run in "
fffff800`7460006c "DOS mode...$"
```

Searching for Functions

In WinDbg, execute this command:

- `x nt!*`

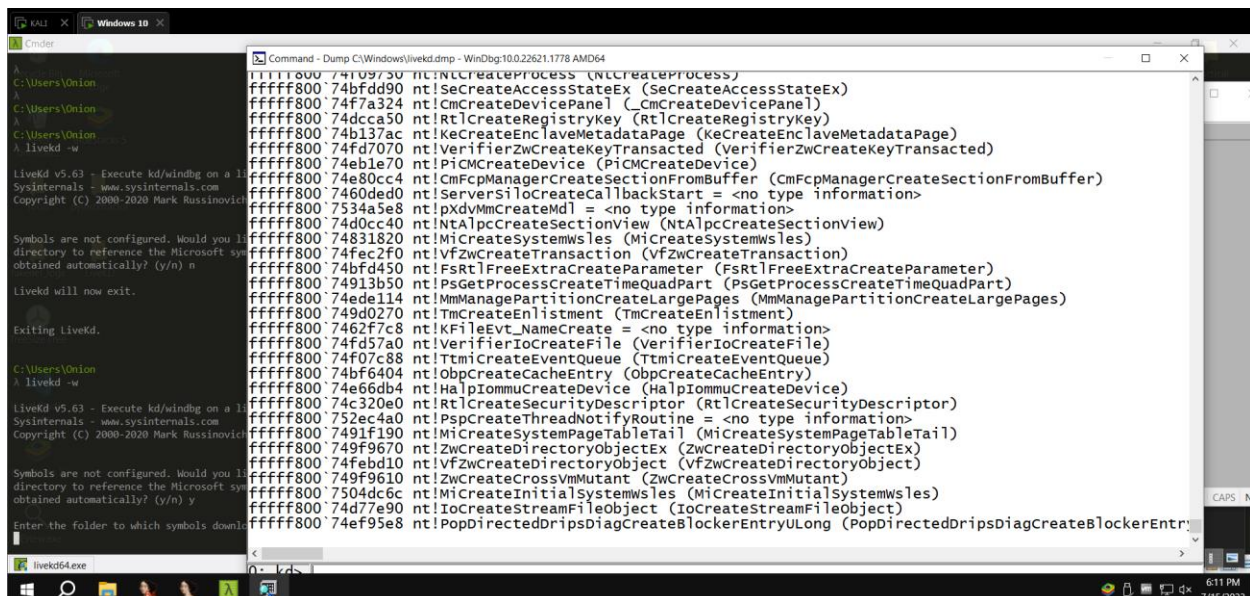
The `x nt!` command in WinDbg is used to display the contents of a memory region as a dump of NT structures, using the WinDbg symbol table to identify the structures and their fields. The `x nt!` command is a more powerful version of the `da nt` command, as it allows you to see the names of the structures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



In WinDbg, execute this command:

`x nt!*Create*`

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

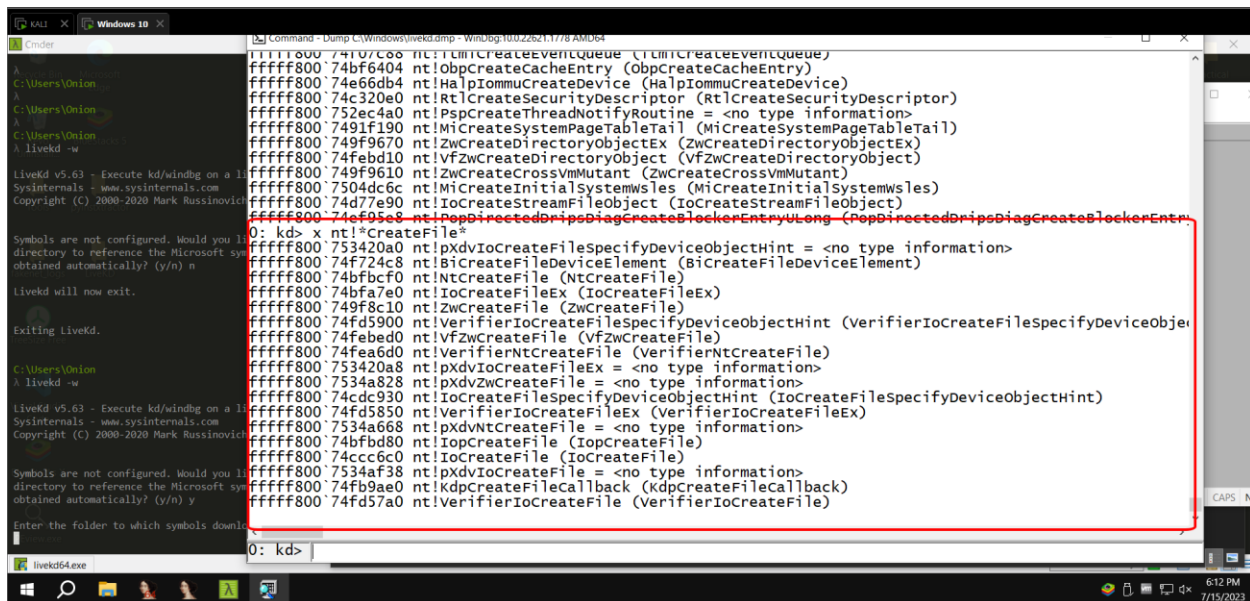


In WinDbg, execute this command:

- `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:



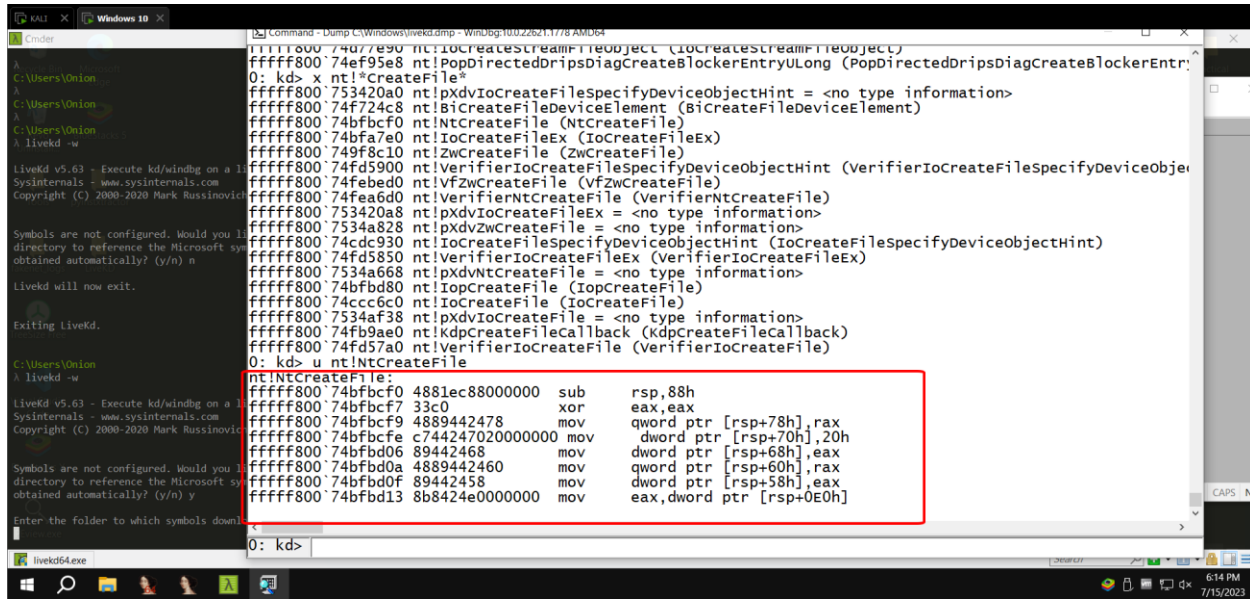
```
0: kd> u nt!NtCreateFile
fffff800`74b64000 nt!NtCreateFile (NtCreateFile)
fffff800`74b64004 nt!ObpCreateCacheEntry (ObpCreateCacheEntry)
fffff800`74b66db4 nt!HalpIoCreateDevice (HalpIoCreateDevice)
fffff800`74c320e0 nt!RtlCreateSecurityDescriptor (RtlCreateSecurityDescriptor)
fffff800`752ce4a0 nt!PspCreateThreadNotifyRoutine = <no type information>
fffff800`7491f190 nt!MiCreateSystemPageTableTail (MiCreateSystemPageTableTail)
fffff800`749f9670 nt!ZwCreateDirectoryObjectEx (ZwCreateDirectoryObjectEx)
fffff800`74febdl0 nt!VfZwCreateDirectoryObject (VfZwCreateDirectoryObject)
fffff800`749f9610 nt!ZwCreateCrossVmMutant (ZwCreateCrossVmMutant)
fffff800`7504dc6c nt!MiCreateInitialSystemWsls (MiCreateInitialSystemWsls)
fffff800`74d77e90 nt!IoCreateStreamFileObject (IoCreateStreamFileObject)
fffff800`74e95e8 nt!PopDirectedDripsDiagCreateBlockerEntryUlong (PopDirectedDripsDiagCreateBlockerEntryUlong)
0: kd> x nt!NtCreateFile
fffff800`753420a0 nt!pxdVioCreateFileSpecifyDeviceObjectHint = <no type information>
fffff800`74f724c8 nt!BiCreateFileDeviceElement (BiCreateFileDeviceElement)
fffff800`74bfbcf0 nt!NtCreateFile (NtCreateFile)
fffff800`74bfa7e0 nt!IoCreateFileEx (IoCreateFileEx)
fffff800`749f8c10 nt!ZwCreateFile (ZwCreateFile)
fffff800`74fd5900 nt!VerifierIoCreateFileSpecifyDeviceObjectHint (VerifierIoCreateFileSpecifyDeviceObjectHint)
fffff800`74febed0 nt!VfZwCreateFile (VfZwCreateFile)
fffff800`74fea6d0 nt!VerifierNtCreateFile (VerifierNtCreateFile)
fffff800`753420a8 nt!pxdVioCreateFileEx = <no type information>
fffff800`7534a828 nt!pxdVzCreateFile = <no type information>
fffff800`74cdc930 nt!IoCreateFileSpecifyDeviceObjectHint (IoCreateFileSpecifyDeviceObjectHint)
fffff800`74fd5850 nt!VerifierIoCreateFileEx (VerifierIoCreateFileEx)
fffff800`7534a668 nt!pxdVntCreateFile = <no type information>
fffff800`74bfbdb0 nt!IoCreateFile (IoCreateFile)
fffff800`74ccc6c0 nt!IoCreateFile (IoCreateFile)
fffff800`7534af38 nt!pxdVioCreateFile = <no type information>
fffff800`74fb9ae0 nt!KdpCreateFileCallback (KdpCreateFileCallback)
fffff800`74fd57a0 nt!VerifierIoCreateFile (VerifierIoCreateFile)
```

Unassembling a Function

In WinDbg, execute this command:

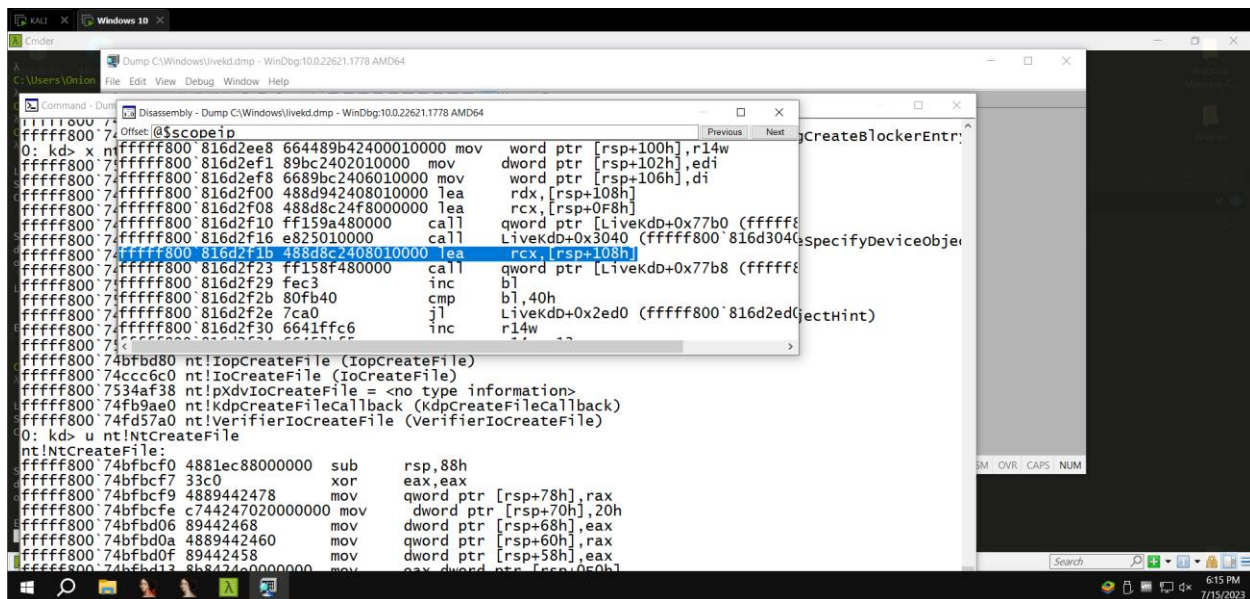
- `u nt!NtCreateFile`

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



```
0: kd> u nt!NtCreateFile
nt!NtCreateFile:
fffff800`74bfbcf0 4881ec88000000 sub     rsp,88h
fffff800`74bfbcf7 33c0 xor     eax,eax
fffff800`74bfbcf9 4889442478 mov     qword ptr [rsp+78h],rax
fffff800`74bfbcf9 c744247020000000 mov     dword ptr [rsp+70h],20h
fffff800`74bfbdb0 89442468 mov     dword ptr [rsp+68h],eax
fffff800`74bfbdb0 88942460 mov     qword ptr [rsp+60h],rax
fffff800`74bfbdb0 89442458 mov     dword ptr [rsp+58h],eax
fffff800`74bfbdb1 8b8424e0000000 mov     eax,dword ptr [rsp+0E0h]
```

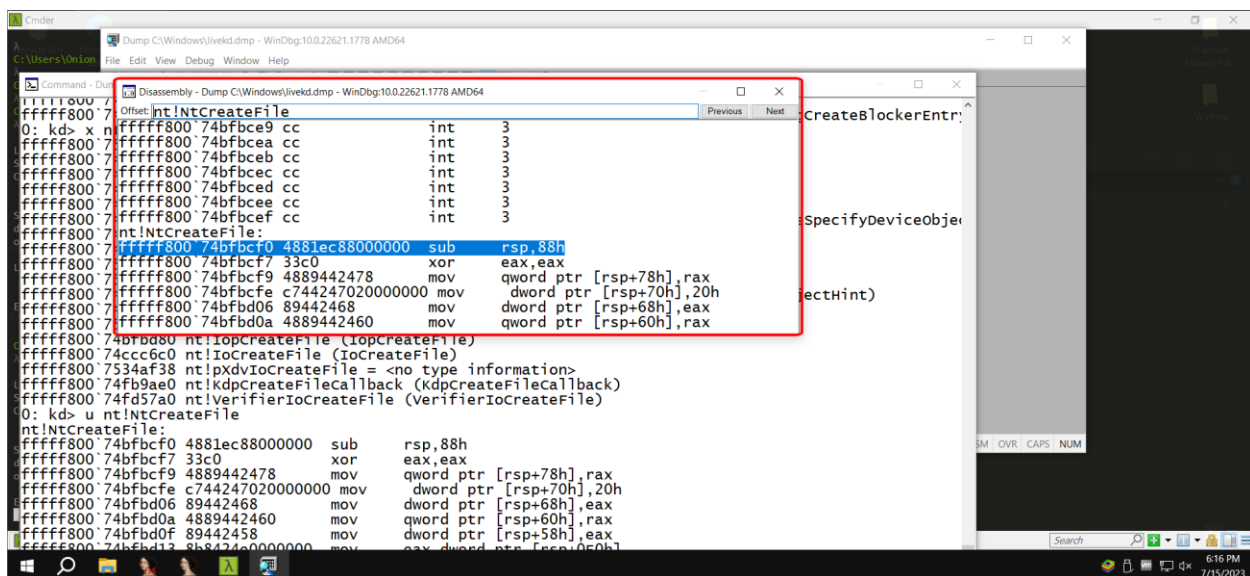
From the WinDbg menu bar, click View, Disassembly.



In the Offset bar at the top, enter

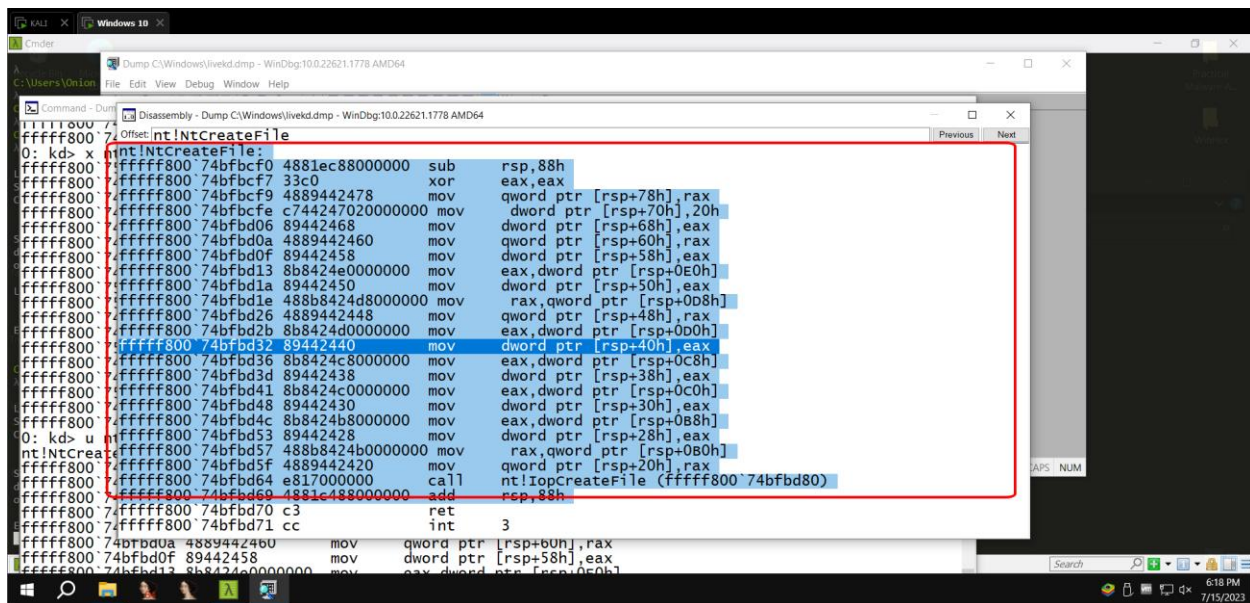
- **nt!NtCreateFile**

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

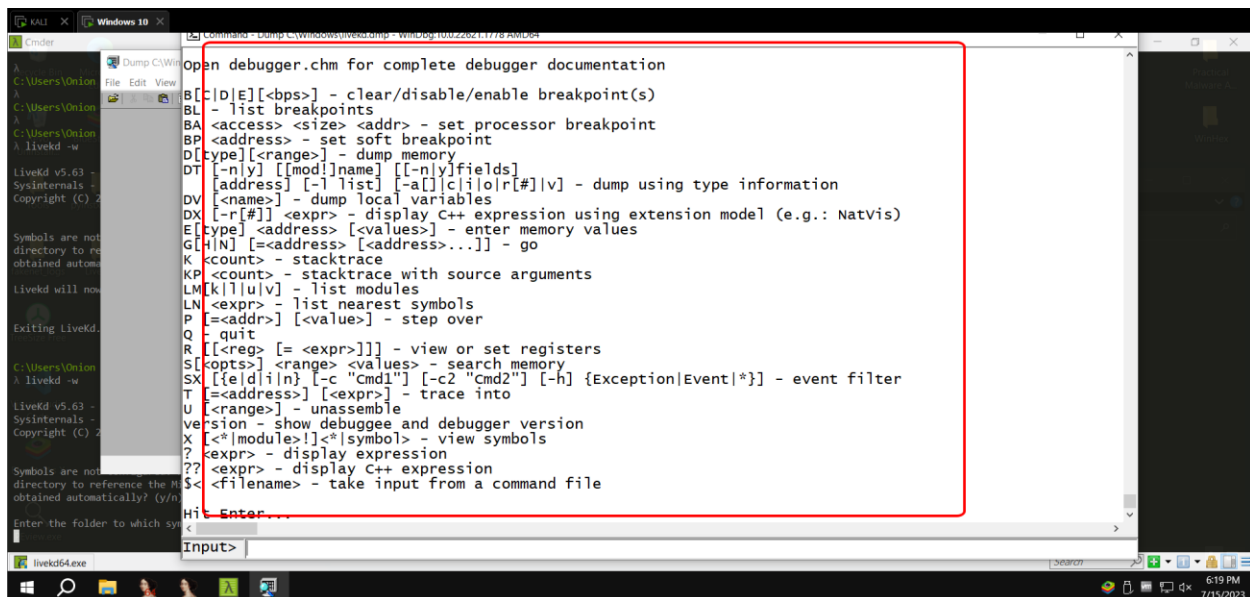
- **nt!NtCreateFile+16**



Online Help

In WinDbg, execute this command:

- ?

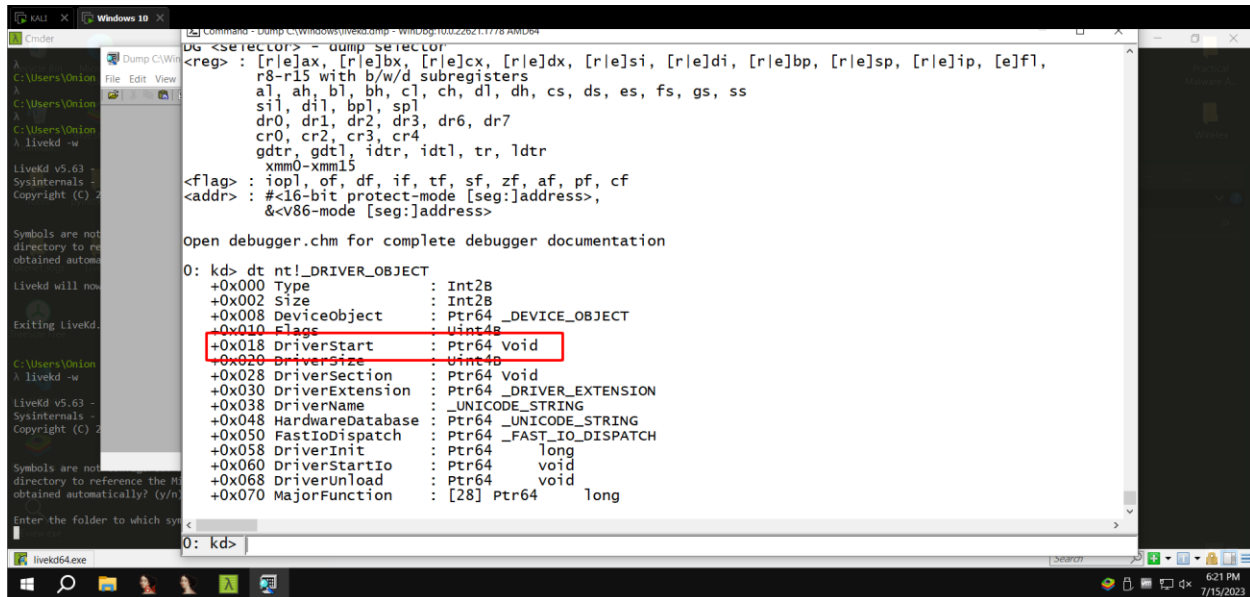


Viewing Type Information for a Structure

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.



```
0: kd> dt nt!_DRIVER_OBJECT
+0x000 Type           : Int2B
+0x002 Size           : Int2B
+0x008 DeviceObject   : Ptr64 _DEVICE_OBJECT
+0x010 Flags          : Uint4B
+0x018 DriverStart    : Ptr64 Void
+0x020 DriverSize     : Uint4B
+0x028 DriverSection  : Ptr64 Void
+0x030 DriverExtension : Ptr64 _DRIVER_EXTENSION
+0x038 DriverName      : _UNICODE_STRING
+0x048 HardwareDatabase : Ptr64 _UNICODE_STRING
+0x050 FastIoDispatch  : Ptr64 _FAST_IO_DISPATCH
+0x058 DriverInit      : Ptr64 long
+0x060 DriverStartIo   : Ptr64 void
+0x068 DriverUnload    : Ptr64 void
+0x070 MajorFunction   : [28] Ptr64 long
```

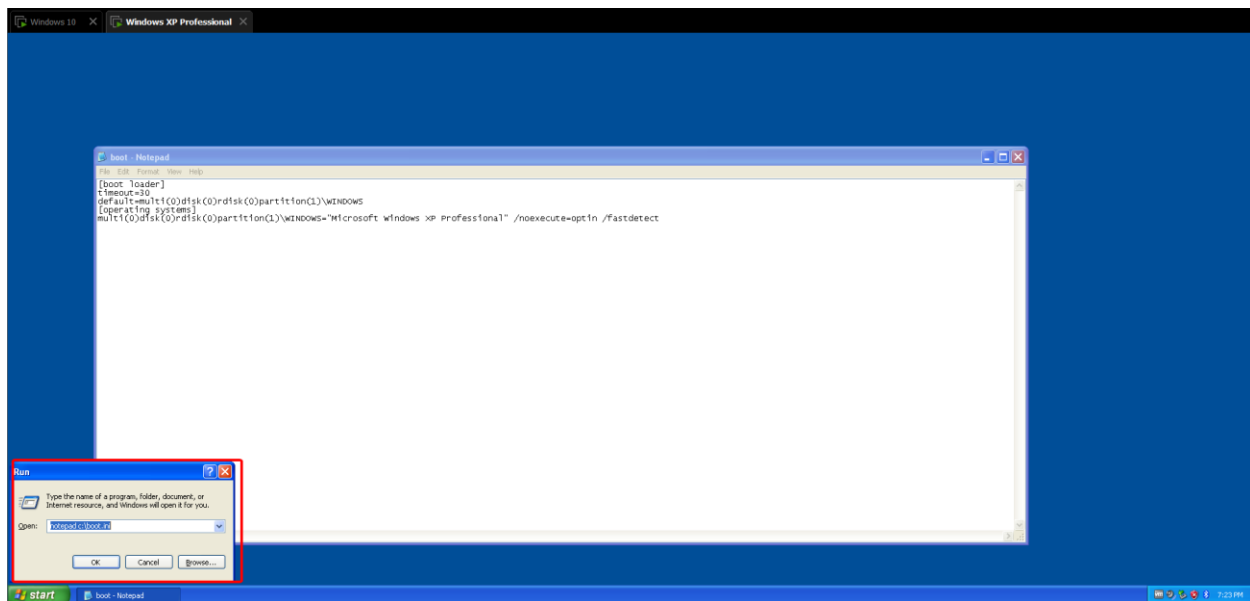
Adding a Boot Menu Item to the TARGET machine

Start the Windows virtual machine.

Click Start, Run.

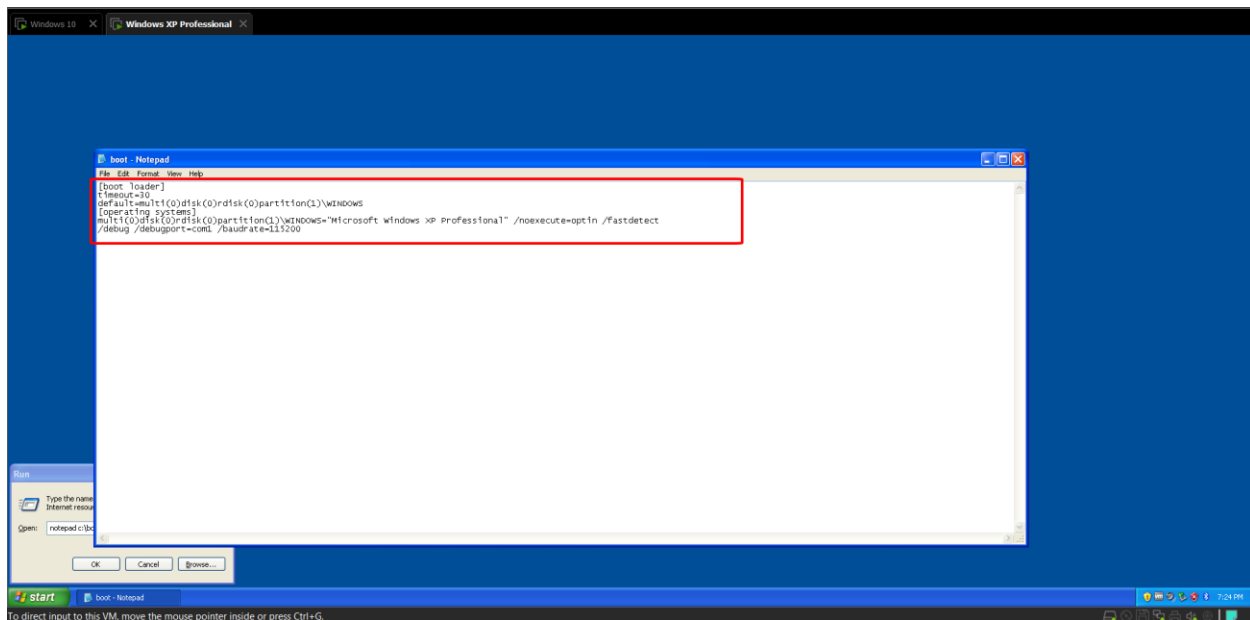
In the Run box, execute this command:

- **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.

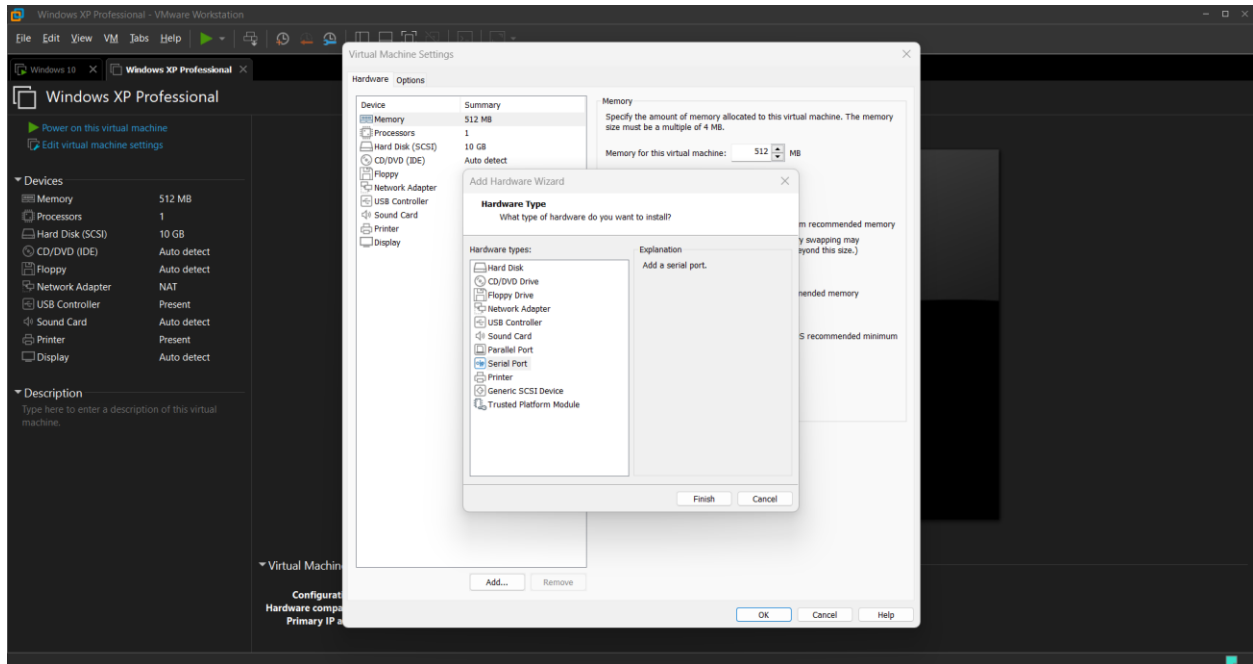
In the left pane of VMware Player, click your **TARGET** machine.

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

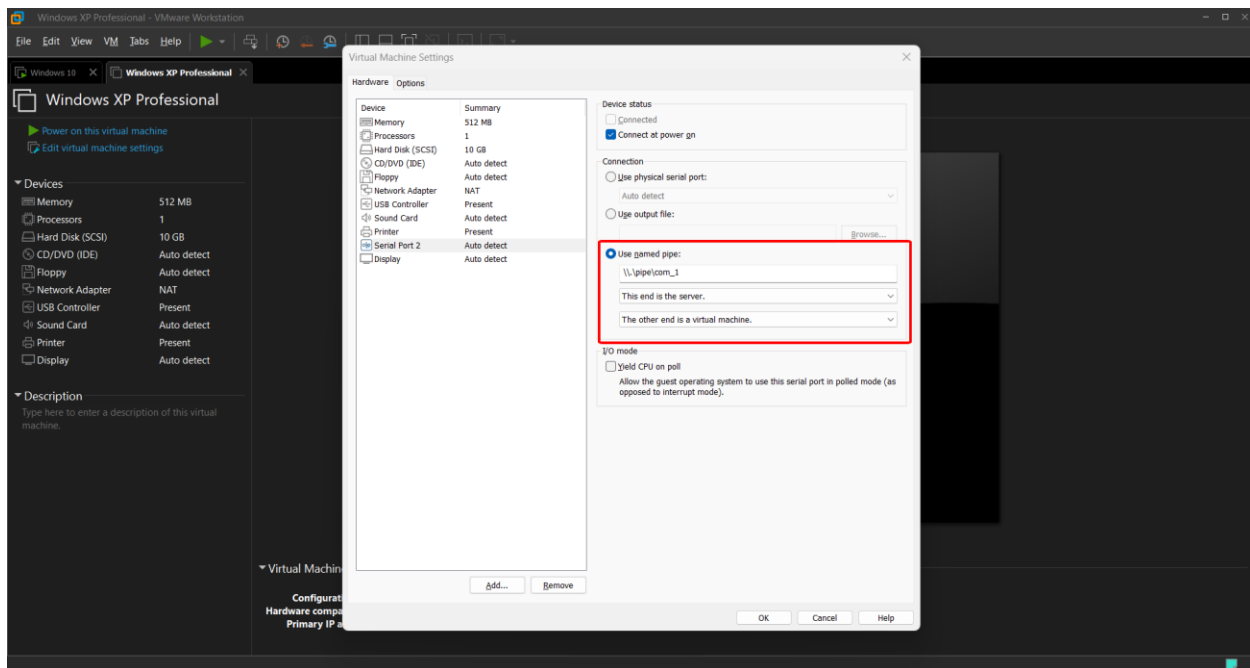
In the "Hardware Type" box, select " **Serial Port**" ", as shown below.



Click **Next**

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

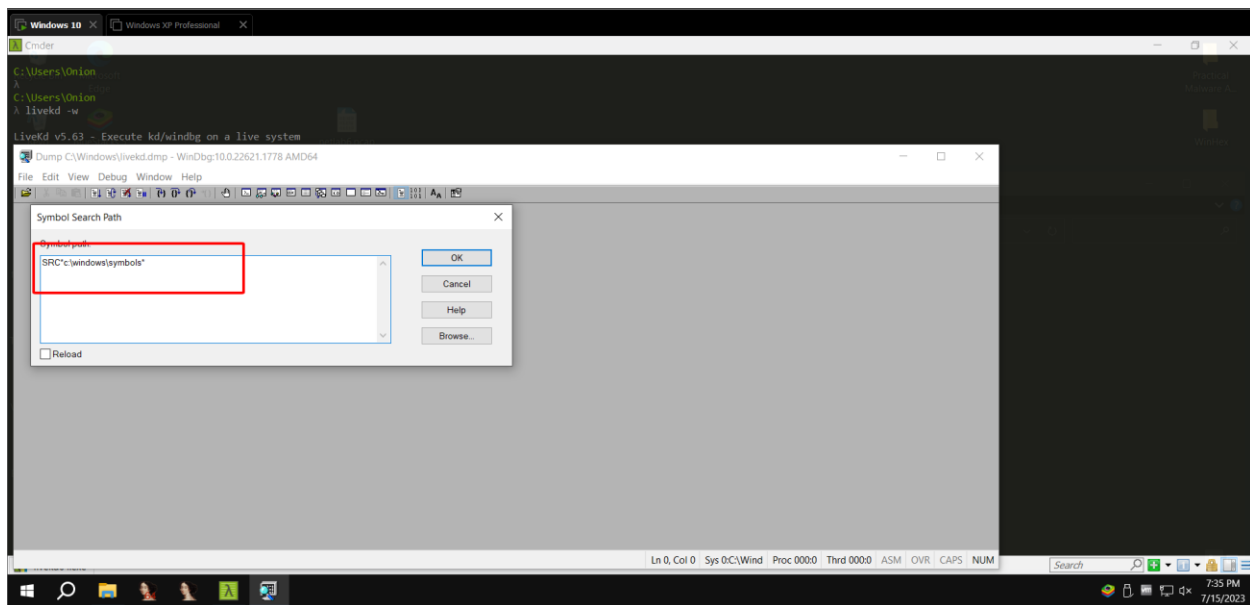
- **\\.\pipe\com_1**



Configuring Symbols in WinDbg

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

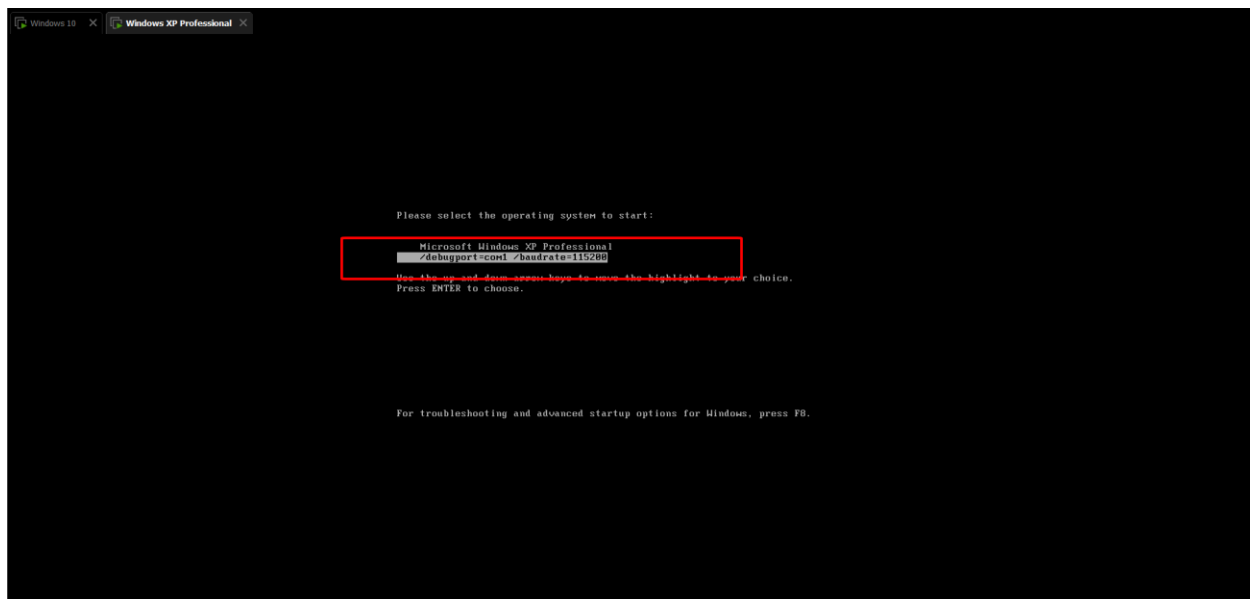
- **SRC*c:\windows\symbols***



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.



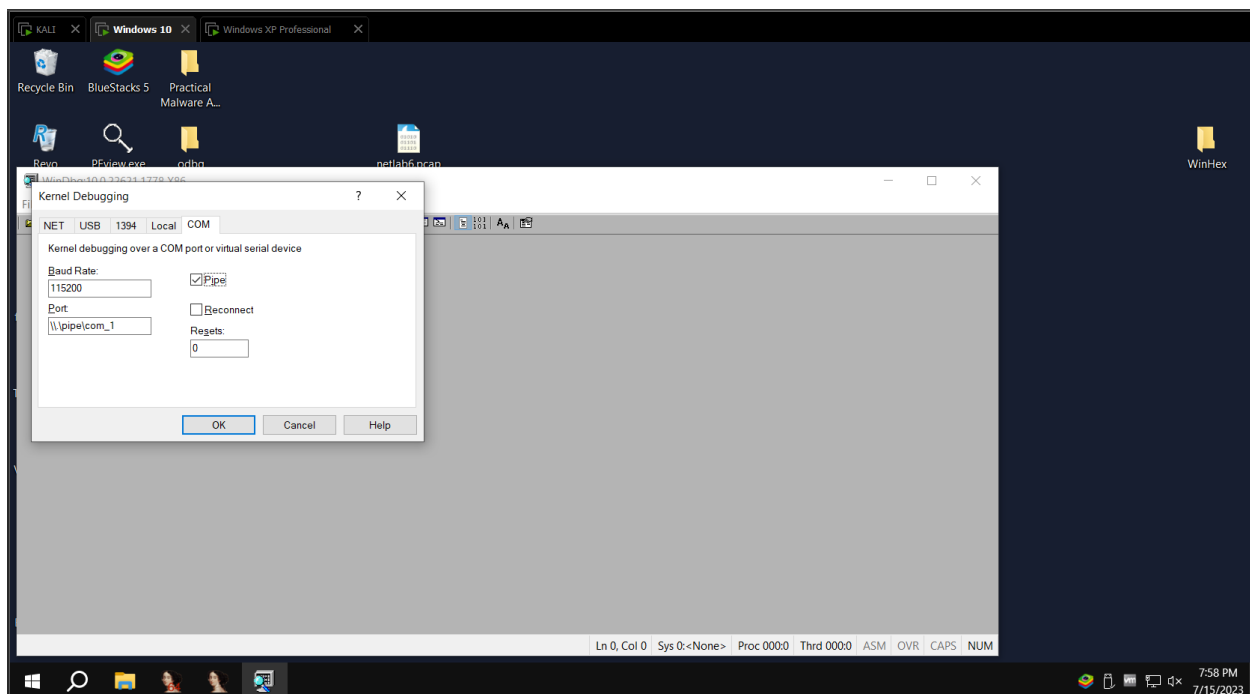
Starting Kernel Debugging

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**"