NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302

LAB 17: Using Kernel Debugging Commands with WinDbg

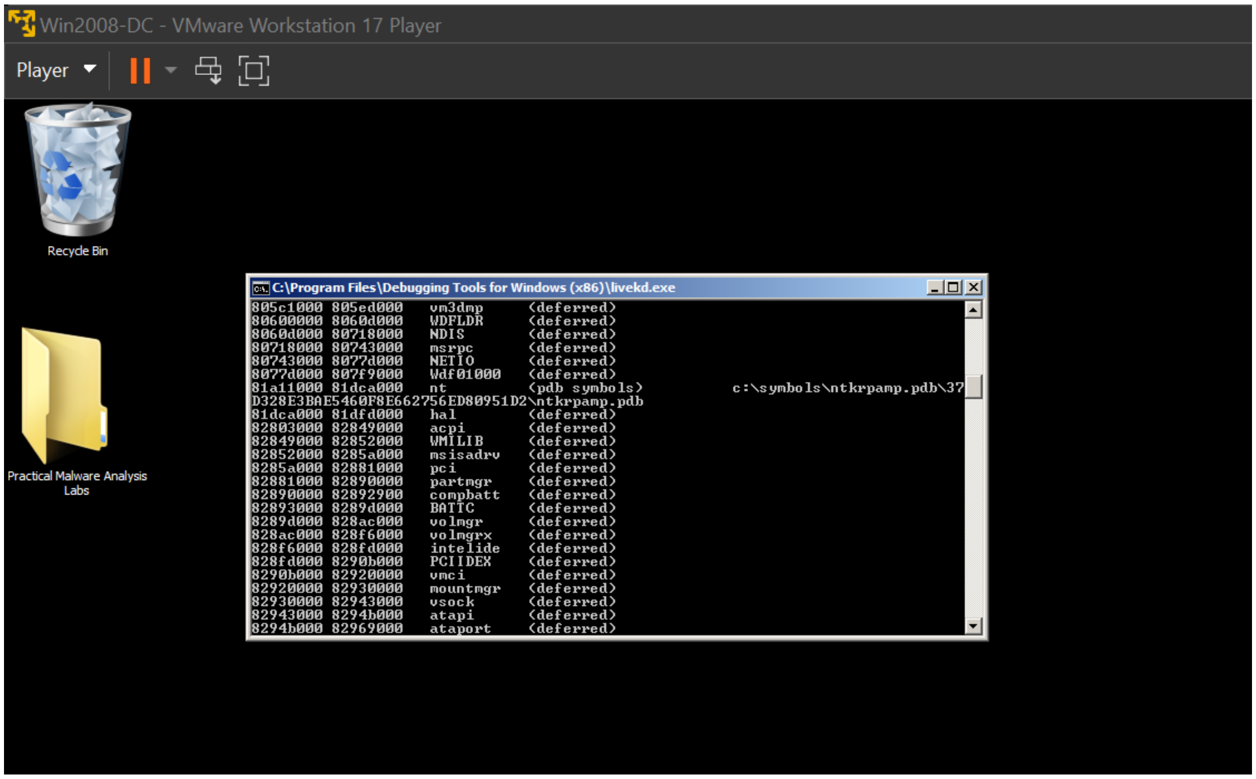
Listing Modules with lm

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

A long list of loaded modules scrolls by. Scroll back to see the lm command you entered, and the first few loaded kernel modules, as shown below.

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

This is Ntoskrnl, the main kernel module.


Win2008-DC - VMware Workstation 17 Player
Player
Recycle Bin
Practical Malware Analysis Labs
C:\Program Files\Debugging Tools for Windows (x86)\livekd.exe

```
805c1000 805ed000   vm3dmp     (deferred)
80600000 8060d000   WDFLDR     (deferred)
8060d000 80718000   NDIS       (deferred)
80718000 80743000   msrpc      (deferred)
80743000 8077d000   NETIO      (deferred)
8077d000 807f9000   Wdf01000   (deferred)
81a11000 81dca000   nt         (pdb symbols)          c:\symbols\ntkrpamp.pdb\37
D328E3BAE5460F8E662756ED80951D2\ntkrpamp.pdb
81dca000 81dfd000   hal        (deferred)
82803000 82849000   acpi       (deferred)
82849000 82852000   WMILIB     (deferred)
82852000 8285a000   msisadrv   (deferred)
8285a000 82881000   pci        (deferred)
82881000 82890000   partmgr    (deferred)
82890000 82892900   compbatt   (deferred)
82893000 8289d000   BATTC      (deferred)
8289d000 828ac000   volmgr     (deferred)
828ac000 828f6000   volmgrx    (deferred)
828f6000 828fd000   intelide   (deferred)
828fd000 8290b000   PCIIDEX    (deferred)
8290b000 82920000   vmci       (deferred)
82920000 82930000   mountmgr   (deferred)
82930000 82943000   vsock      (deferred)
82943000 8294b000   atapi      (deferred)
8294b000 82969000   ataport    (deferred)
```

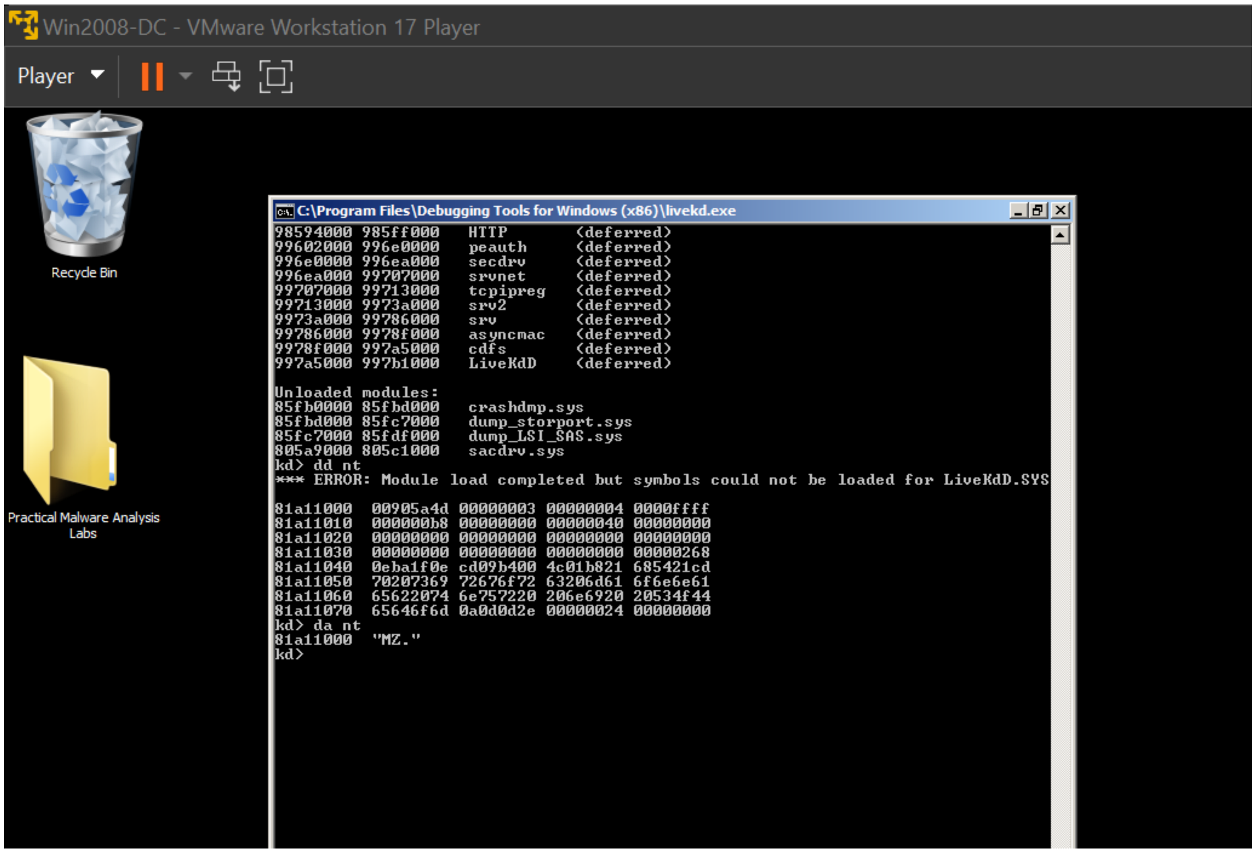
Viewing Memory

In WinDbg, execute this command: **dd nt**

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**

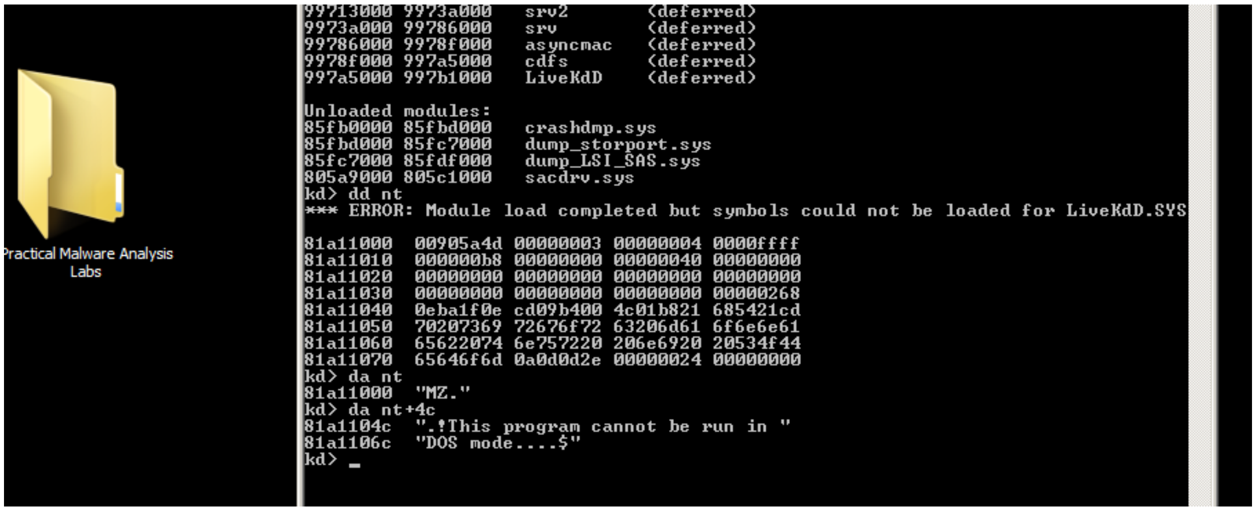
You see the characters "MZ" --they are at the start of every EXE file.


Win2008-DC - VMware Workstation 17 Player
Player
Recycle Bin
Practical Malware Analysis Labs
C:\Program Files\Debugging Tools for Windows (x86)\livekd.exe

```
98594000 985ff000   HTTP       (deferred)
99602000 996e0000   peauth     (deferred)
996e0000 996ea000   secdrv     (deferred)
996ea000 99707000   srvnet     (deferred)
99707000 99713000   tcpipreg   (deferred)
99713000 9973a000   srv2       (deferred)
9973a000 99786000   srv        (deferred)
99786000 9978f000   asyncmac   (deferred)
9978f000 997a5000   cdfs       (deferred)
997a5000 997b1000   LiveKdD    (deferred)

Unloaded modules:
85fb0000 85fbd000   crashdmp.sys
85fbd000 85fc7000   dump_storport.sys
85fc7000 85fdf000   dump_LSI_SAS.sys
805a9000 805c1000   sacdrv.sys
kd> dd nt
*** ERROR: Module load completed but symbols could not be loaded for LiveKdD.SYS

81a11000  00905a4d 00000003 00000004 0000ffff
81a11010  000000b8 00000000 00000040 00000000
81a11020  00000000 00000000 00000000 00000000
81a11030  00000000 00000000 00000000 00000268
81a11040  0eba1f0e cd09b400 4c01b821 685421cd
81a11050  70207369 72676f72 63206d61 6f6e6e61
81a11060  65622074 6e757220 206e6920 20534f44
81a11070  65646f6d 0a0d0d2e 00000024 00000000
kd> da nt
81a11000  "MZ."
kd>
```

In WinDbg, execute this command: **da nt+4c**

You see the message "This program cannot be run in DOS mode", as shown below:

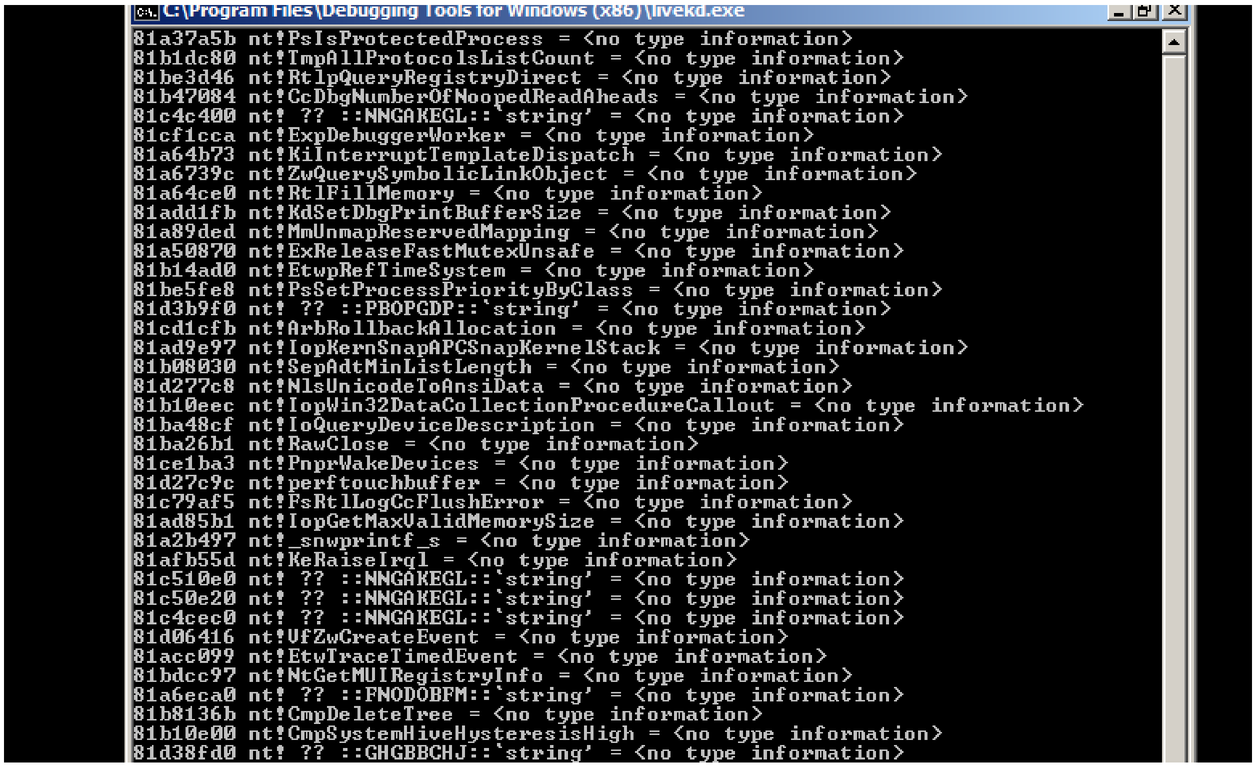


```
99713000 9973a000   srv2       (deferred)
9973a000 99786000   srv        (deferred)
99786000 9978f000   asyncmac   (deferred)
9978f000 997a5000   cdfs       (deferred)
997a5000 997b1000   LiveKdD    (deferred)

Unloaded modules:
85fb0000 85fbd000   crashdmp.sys
85fbd000 85fc7000   dump_storport.sys
85fc7000 85fdf000   dump_LSI_SAS.sys
805a9000 805c1000   sacdrv.sys
kd> dd nt
*** ERROR: Module load completed but symbols could not be loaded for LiveKdD.SYS

81a11000  00905a4d 00000003 00000004 0000ffff
81a11010  000000b8 00000000 00000040 00000000
81a11020  00000000 00000000 00000000 00000000
81a11030  00000000 00000000 00000000 00000268
81a11040  0eba1f0e cd09b400 4c01b821 685421cd
81a11050  70207369 72676f72 63206d61 6f6e6e61
81a11060  65622074 6e757220 206e6920 20534f44
81a11070  65646f6d 0a0d0d2e 00000024 00000000
kd> da nt
81a11000  "MZ."
kd> da nt+4c
81a1104c  ".!This program cannot be run in "
81a1106c  "DOS mode....$"
kd> _
```

Searching for Functions

In WinDbg, execute this command: **x nt!***

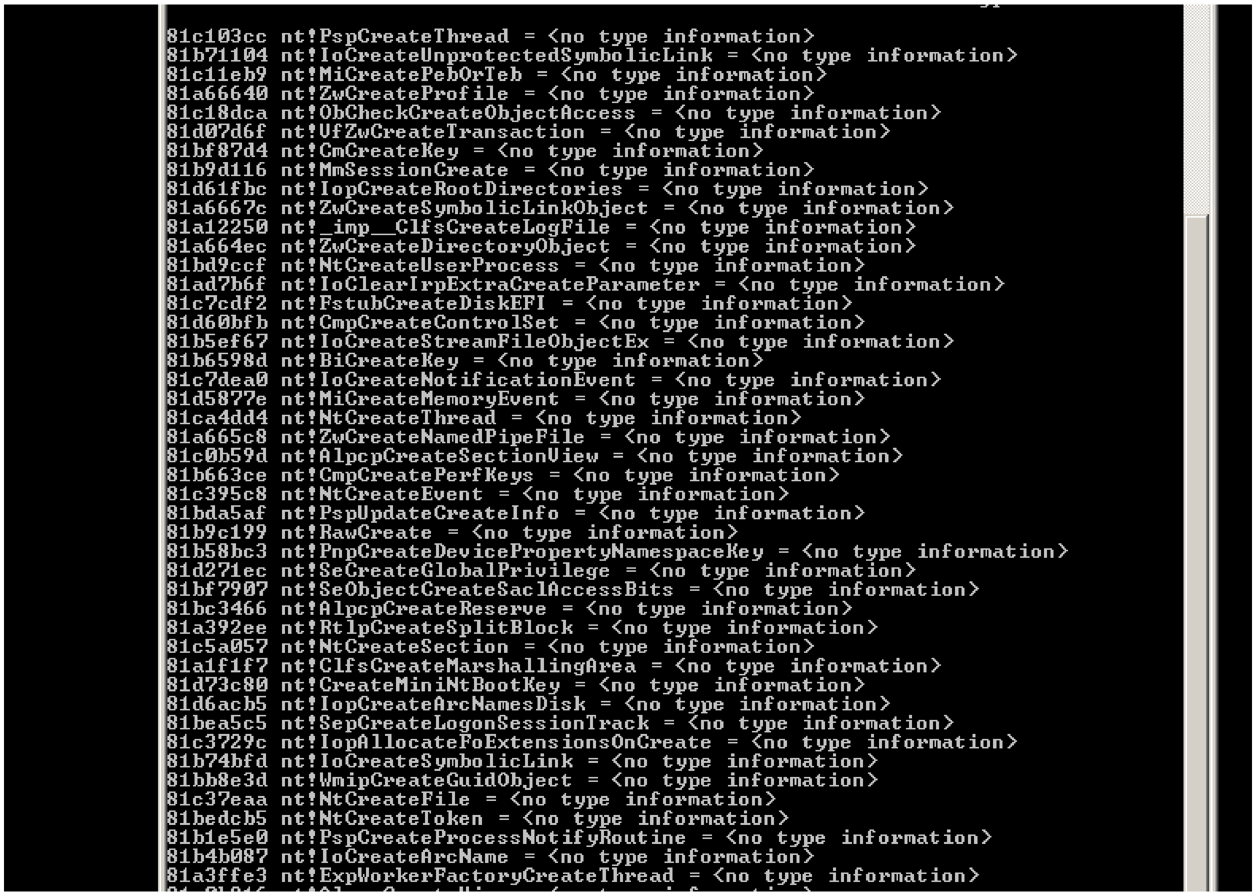
This finds all the functions in Ntoskrnl. There are a lot of them, as shown below:


C:\Program Files\Debugging Tools for Windows (x86)\livekd.exe

```
81a37a5b nt!PsIsProtectedProcess = <no type information>
81b1dc80 nt!TmpAllProtocolsListCount = <no type information>
81be3d46 nt!RtlpQueryRegistryDirect = <no type information>
81b47084 nt!CcDbgNumberOfNoopedReadAheads = <no type information>
81c4c400 nt! ?? ::NNGAKEGL::`string' = <no type information>
81cf1cca nt!ExpDebuggerWorker = <no type information>
81a64b73 nt!KiInterruptTemplateDispatch = <no type information>
81a6739c nt!ZwQuerySymbolicLinkObject = <no type information>
81a64ce0 nt!RtlFillMemory = <no type information>
81add1fb nt!KdSetDbgPrintBufferSize = <no type information>
81a89ded nt!MmUnmapReservedMapping = <no type information>
81a50870 nt!ExReleaseFastMutexUnsafe = <no type information>
81b14ad0 nt!EtwpRefTimeSystem = <no type information>
81be5fe8 nt!PsSetProcessPriorityByClass = <no type information>
81d3b9f0 nt! ?? ::PBOPGDP::`string' = <no type information>
81cd1cfb nt!ArbRollbackAllocation = <no type information>
81ad9e97 nt!IopKernSnapAPCSnapKernelStack = <no type information>
81b08030 nt!SepAdtMinListLength = <no type information>
81d277c8 nt!NlsUnicodeToAnsiData = <no type information>
81b10eec nt!IopWin32DataCollectionProcedureCallout = <no type information>
81ba48cf nt!IoQueryDeviceDescription = <no type information>
81ba26b1 nt!RawClose = <no type information>
81ce1ba3 nt!PnprWakeDevices = <no type information>
81d27c9c nt!perftouchbuffer = <no type information>
81c79af5 nt!FsRtlLogCcFlushError = <no type information>
81ad85b1 nt!IopGetMaxValidMemorySize = <no type information>
81a2b497 nt!_snwprintf_s = <no type information>
81afb55d nt!KeRaiseIrql = <no type information>
81c510e0 nt! ?? ::NNGAKEGL::`string' = <no type information>
81c50e20 nt! ?? ::NNGAKEGL::`string' = <no type information>
81c4cec0 nt! ?? ::NNGAKEGL::`string' = <no type information>
81d06416 nt!VfZwCreateEvent = <no type information>
81acc099 nt!EtwTraceTimedEvent = <no type information>
81bdcc97 nt!NtGetMUIRegistryInfo = <no type information>
81a6eca0 nt! ?? ::FNODOBFM::`string' = <no type information>
81b8136b nt!CmpDeleteTree = <no type information>
81b10e00 nt!CmpSystemHiveHysteresisHigh = <no type information>
81d38fd0 nt! ?? ::GHGBBCHJ::`string' = <no type information>
```

In WinDbg, execute this command: **x nt!*Create***

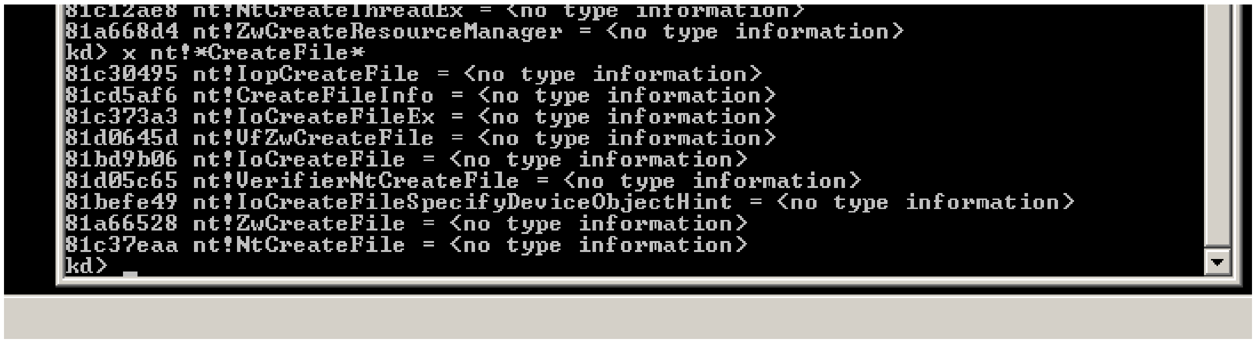
This finds all the functions in Ntoskrnl that contain the word "Create". There are a lot of them, too.



```
81c103cc nt!PspCreateThread = <no type information>
81b71104 nt!IoCreateUnprotectedSymbolicLink = <no type information>
81c11eb9 nt!MiCreatePebOrTeb = <no type information>
81a66640 nt!ZwCreateProfile = <no type information>
81c18dca nt!ObCheckCreateObjectAccess = <no type information>
81d07d6f nt!VfZwCreateTransaction = <no type information>
81bf87d4 nt!CmCreateKey = <no type information>
81b9d116 nt!MmSessionCreate = <no type information>
81d61fbc nt!IopCreateRootDirectories = <no type information>
81a6667c nt!ZwCreateSymbolicLinkObject = <no type information>
81a12250 nt!_imp__ClfsCreateLogFile = <no type information>
81a664ec nt!ZwCreateDirectoryObject = <no type information>
81bd9ccf nt!NtCreateUserProcess = <no type information>
81ad7b6f nt!IoClearIrpExtraCreateParameter = <no type information>
81c7cdf2 nt!FstubCreateDiskEFI = <no type information>
81d60bfb nt!CmpCreateControlSet = <no type information>
81b5ef67 nt!IoCreateStreamFileObjectEx = <no type information>
81b6598d nt!BiCreateKey = <no type information>
81c7dea0 nt!IoCreateNotificationEvent = <no type information>
81d5877e nt!MiCreateMemoryEvent = <no type information>
81ca4dd4 nt!NtCreateThread = <no type information>
81a665c8 nt!ZwCreateNamedPipeFile = <no type information>
81c0b59d nt!AlpcpCreateSectionView = <no type information>
81b663ce nt!CmpCreatePerfKeys = <no type information>
81c395c8 nt!NtCreateEvent = <no type information>
81bda5af nt!PspUpdateCreateInfo = <no type information>
81b9c199 nt!RawCreate = <no type information>
81b58bc3 nt!PnpCreateDevicePropertyNamespaceKey = <no type information>
81d271ec nt!SeCreateGlobalPrivilege = <no type information>
81bf7907 nt!SeObjectCreateSaclAccessBits = <no type information>
81bc3466 nt!AlpcpCreateReserve = <no type information>
81a392ee nt!RtlpCreateSplitBlock = <no type information>
81c5a057 nt!NtCreateSection = <no type information>
81a1f1f7 nt!ClfsCreateMarshallingArea = <no type information>
81d73c80 nt!CreateMiniNtBootKey = <no type information>
81d6acb5 nt!IopCreateArcNamesDisk = <no type information>
81bea5c5 nt!SepCreateLogonSessionTrack = <no type information>
81c3729c nt!IopAllocateFoExtensionsOnCreate = <no type information>
81b74bfd nt!IoCreateSymbolicLink = <no type information>
81bb8e3d nt!WmipCreateGuidObject = <no type information>
81c37eaa nt!NtCreateFile = <no type information>
81bedcb5 nt!NtCreateToken = <no type information>
81b1e5e0 nt!PspCreateProcessNotifyRoutine = <no type information>
81b4b087 nt!IoCreateArcName = <no type information>
81a3ffe3 nt!ExpWorkerFactoryCreateThread = <no type information>
```

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:



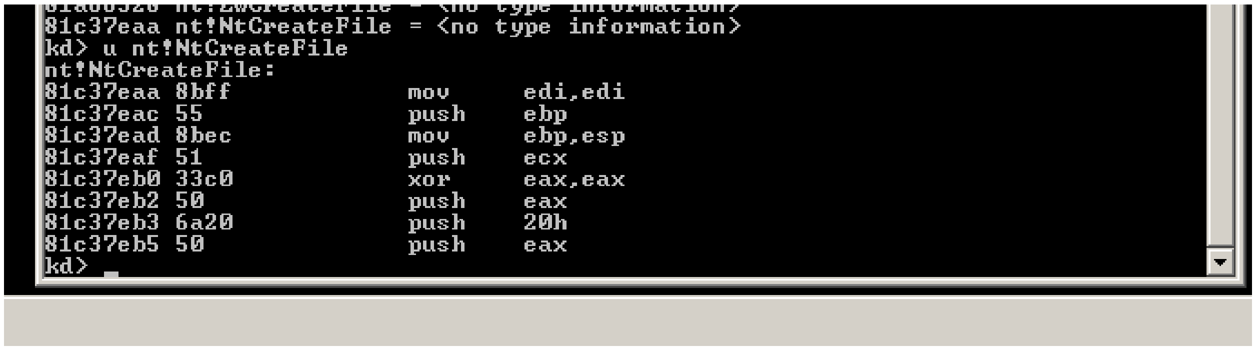
```
81c12ae8 nt!NtCreateThreadEx = <no type information>
81a668d4 nt!ZwCreateResourceManager = <no type information>
kd> x nt!*CreateFile*
81c30495 nt!IopCreateFile = <no type information>
81cd5af6 nt!CreateFileInfo = <no type information>
81c373a3 nt!IoCreateFileEx = <no type information>
81d0645d nt!VfZwCreateFile = <no type information>
81bd9b06 nt!IoCreateFile = <no type information>
81d05c65 nt!VerifierNtCreateFile = <no type information>
81befe49 nt!IoCreateFileSpecifyDeviceObjectHint = <no type information>
81a66528 nt!ZwCreateFile = <no type information>
81c37eaa nt!NtCreateFile = <no type information>
kd> _
```

Unassembling a Function

In WinDbg, execute this command: **u nt!NtCreateFile**

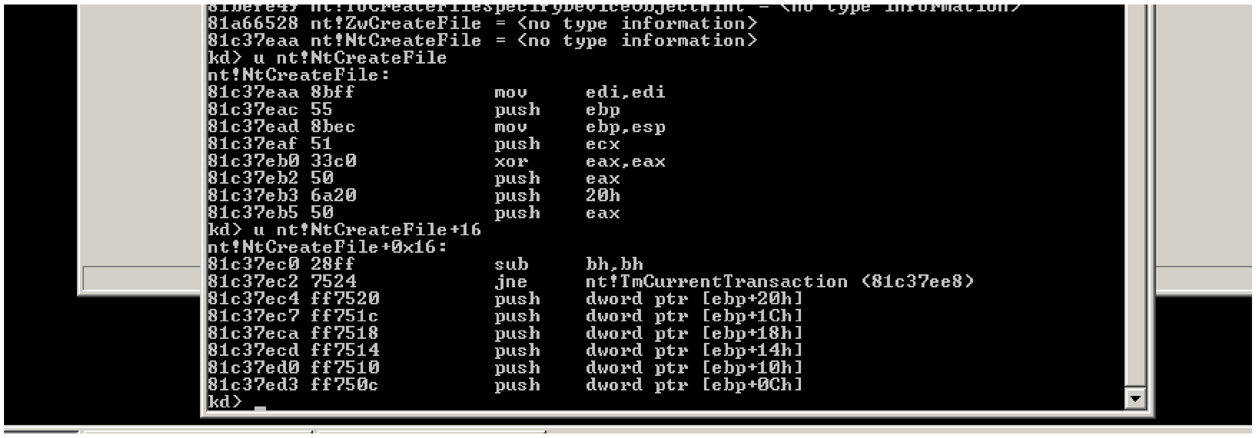
NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302

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



```
81c37eaa nt!NtCreateFile = <no type information>
kd> u nt!NtCreateFile
nt!NtCreateFile:
81c37eaa 8bff            mov     edi,edi
81c37eac 55              push    ebp
81c37ead 8bec            mov     ebp,esp
81c37eaf 51              push    ecx
81c37eb0 33c0            xor     eax,eax
81c37eb2 50              push    eax
81c37eb3 6a20            push    20h
81c37eb5 50              push    eax
kd> _
```

nt!NtCreateFile+16

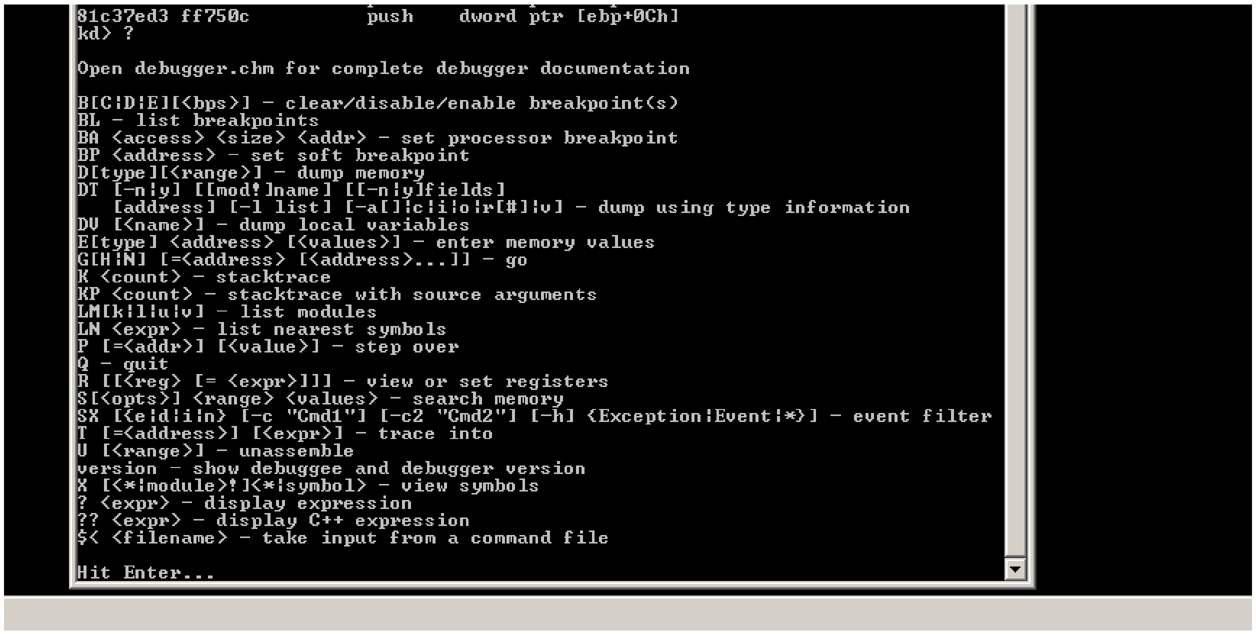


```
81a66528 nt!ZwCreateFile = <no type information>
81c37eaa nt!NtCreateFile = <no type information>
kd> u nt!NtCreateFile
nt!NtCreateFile:
81c37eaa 8bff            mov     edi,edi
81c37eac 55              push    ebp
81c37ead 8bec            mov     ebp,esp
81c37eaf 51              push    ecx
81c37eb0 33c0            xor     eax,eax
81c37eb2 50              push    eax
81c37eb3 6a20            push    20h
81c37eb5 50              push    eax
kd> u nt!NtCreateFile+16
nt!NtCreateFile+0x16:
81c37ec0 28ff            sub     bh,bh
81c37ec2 7524            jne     nt!TmCurrentTransaction (81c37ee8)
81c37ec4 ff7520          push    dword ptr [ebp+20h]
81c37ec7 ff751c          push    dword ptr [ebp+1Ch]
81c37eca ff7518          push    dword ptr [ebp+18h]
81c37ecd ff7514          push    dword ptr [ebp+14h]
81c37ed0 ff7510          push    dword ptr [ebp+10h]
81c37ed3 ff750c          push    dword ptr [ebp+0Ch]
kd> _
```

Online Help

Close the Disassembly window. In WinDbg, execute this command: **?**

You see the first page of the online help, as shown below:

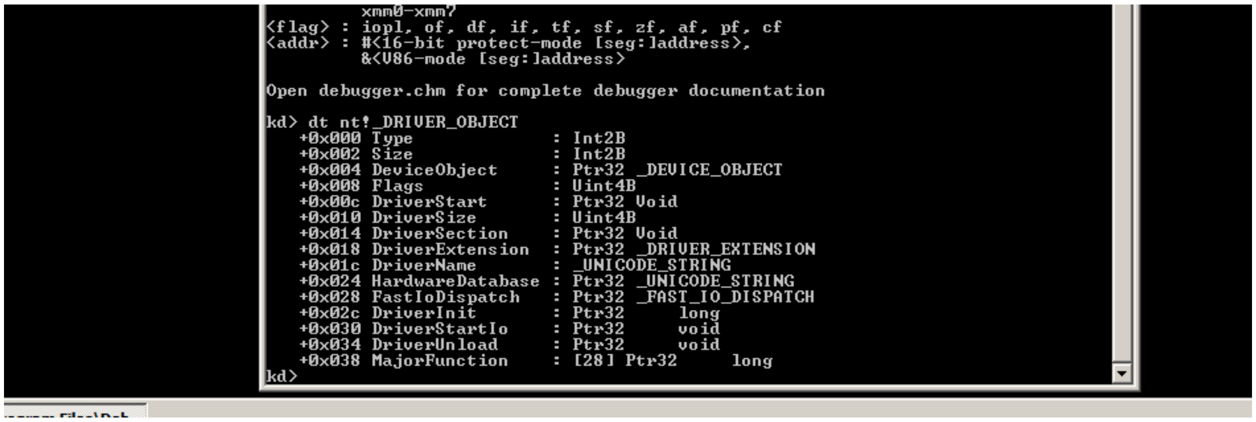


```
81c37ed3 ff750c          push    dword ptr [ebp+0Ch]
kd> ?

Open debugger.chm for complete debugger documentation

B[C|D|E][<bps>] - clear/disable/enable breakpoint(s)
BL - list breakpoints
BA <access> <size> <addr> - set processor breakpoint
BP <address> - set soft breakpoint
D[type][<range>] - dump memory
DT [-n|y] [[mod!]name] [[-n|y]fields]
   [address] [-l list] [-a[]|c|i|o|r[#]|v] - dump using type information
DV [<name>] - dump local variables
E[type] <address> [<values>] - enter memory values
G[H|N] [=<address> [<address>...]] - go
K <count> - stacktrace
KP <count> - stacktrace with source arguments
LM[k|l|u|v] - list modules
LN <expr> - list nearest symbols
P [=<addr>] [<value>] - step over
Q - quit
R [[<reg> [= <expr>]]] - view or set registers
S[<opts>] <range> <values> - search memory
SX [{e|d|i|n} [-c "Cmd1"] [-c2 "Cmd2"] [-h] {Exception|Event|*}] - event filter
T [=<address>] [<expr>] - trace into
U [<range>] - unassemble
version - show debuggee and debugger version
X [<*|module>!]<*|symbol> - view symbols
? <expr> - display expression
?? <expr> - display C++ expression
$< <filename> - take input from a command file

Hit Enter...
```

Viewing Type Information for a Structure

In WinDbg, execute this command: **dt nt!_DRIVER_OBJECT**

This shows the first few lines of a driver object structure, which stores information about a kernel driver, as shown below. Notice the DriverStart pointer--this contains the location of the driver in memory.



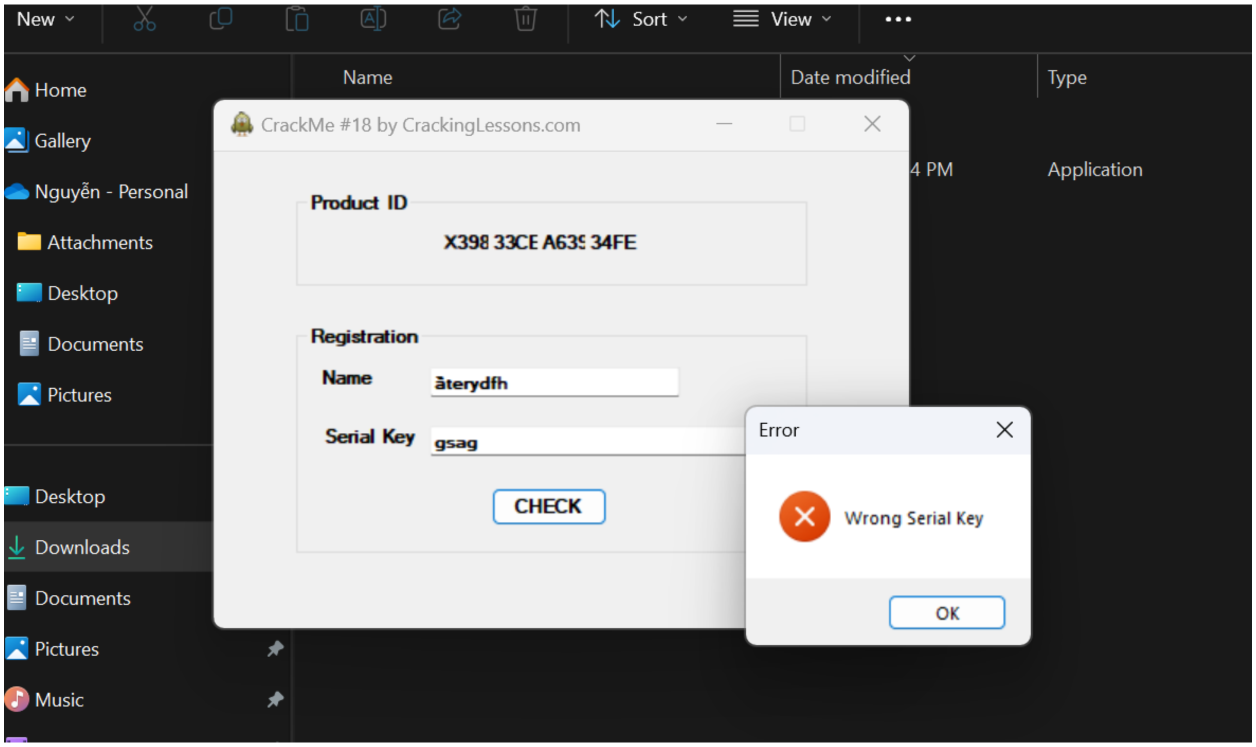
```
          xmm0-xmm7
<flag> : iopl, of, df, if, tf, sf, zf, af, pf, cf
<addr> : #<16-bit protect-mode [seg:]address>,
         &<V86-mode [seg:]address>

Open debugger.chm for complete debugger documentation

kd> dt nt!_DRIVER_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x004 DeviceObject     : Ptr32 _DEVICE_OBJECT
   +0x008 Flags            : Uint4B
   +0x00c DriverStart      : Ptr32 Void
   +0x010 DriverSize       : Uint4B
   +0x014 DriverSection    : Ptr32 Void
   +0x018 DriverExtension  : Ptr32 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING
   +0x024 HardwareDatabase : Ptr32 _UNICODE_STRING
   +0x028 FastIoDispatch   : Ptr32 _FAST_IO_DISPATCH
   +0x02c DriverInit       : Ptr32     long
   +0x030 DriverStartIo    : Ptr32     void
   +0x034 DriverUnload     : Ptr32     void
   +0x038 MajorFunction    : [28] Ptr32     long
kd>
```

NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302

CRACK ME 18

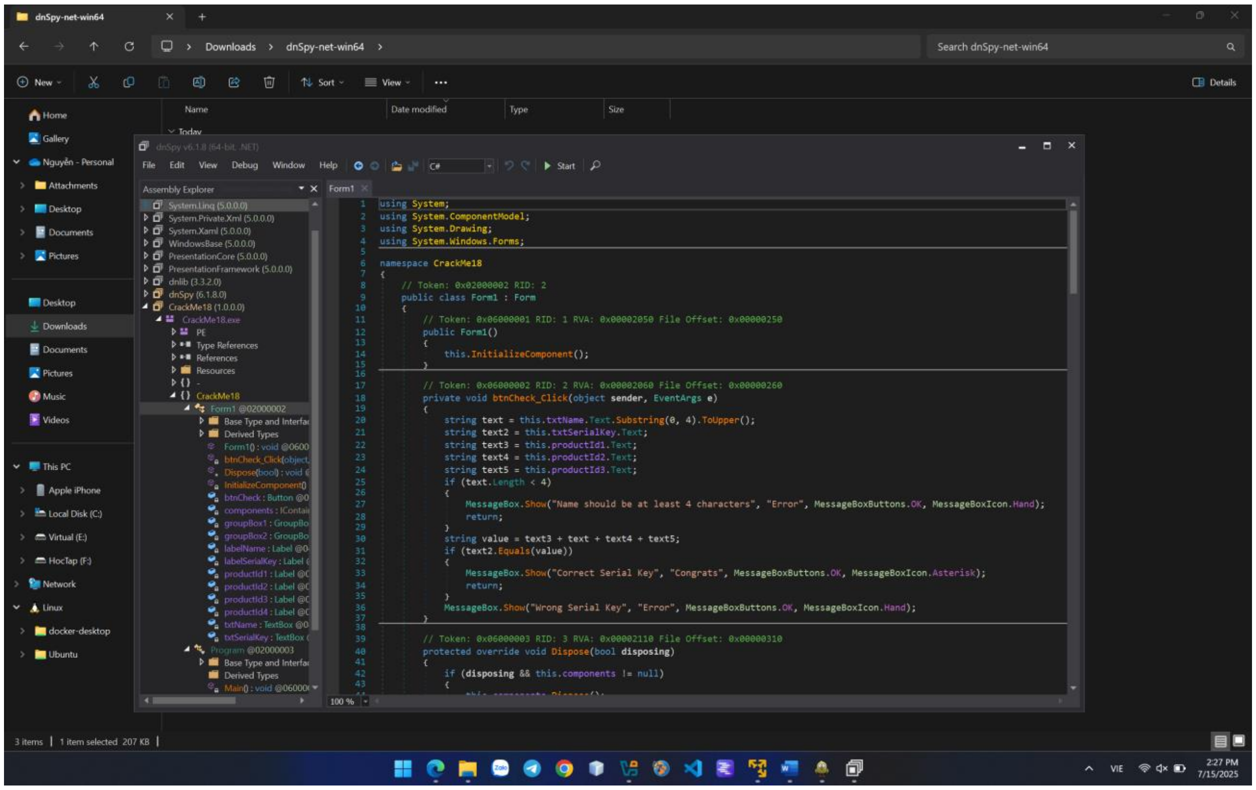
There are 3 tasks:

1. Patch it to always succeed no matter what name and serial key you enter.
2. Do serial fishing to extract the serial key based on a given name of your choice.
3. Create a keygen



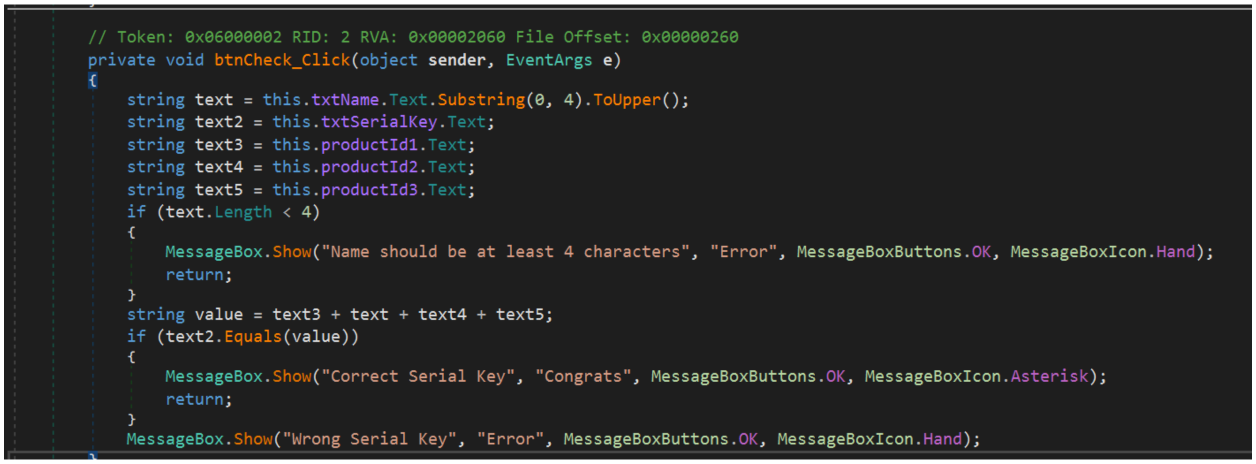
NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302

Because this program using .Net C#, so I using this dnSpy tool to debugging this file .exe



```csharp
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace CrackMe18
{
    // Token: 0x02000002 RID: 2
    public class Form1 : Form
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        public Form1()
        {
            this.InitializeComponent();
        }

        // Token: 0x06000002 RID: 2 RVA: 0x00002060 File Offset: 0x00000260
        private void btnCheck_Click(object sender, EventArgs e)
        {
            string text = this.txtName.Text.Substring(0, 4).ToUpper();
            string text2 = this.txtSerialKey.Text;
            string text3 = this.productId1.Text;
            string text4 = this.productId2.Text;
            string text5 = this.productId3.Text;
            if (text.Length < 4)
            {
                MessageBox.Show("Name should be at least 4 characters", "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
                return;
            }
            string value = text3 + text + text4 + text5;
            if (text2.Equals(value))
            {
                MessageBox.Show("Correct Serial Key", "Congrats", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
                return;
            }
            MessageBox.Show("Wrong Serial Key", "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
        }

        // Token: 0x06000003 RID: 3 RVA: 0x00002110 File Offset: 0x00000310
        protected override void Dispose(bool disposing)
        {
            if (disposing && this.components != null)
            {
```

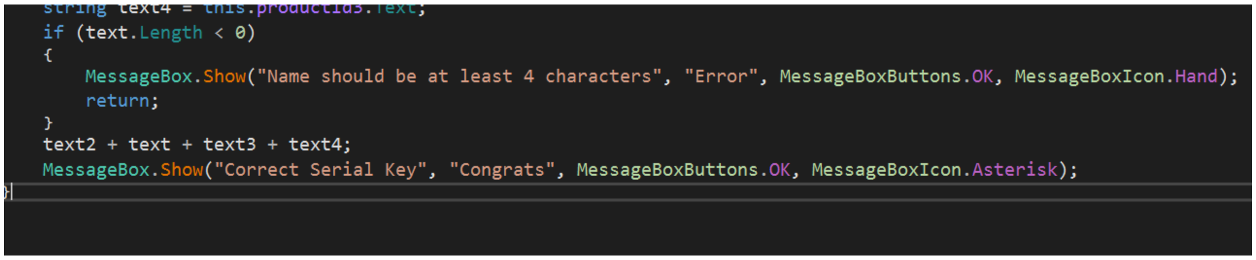
This is the code part of this check program



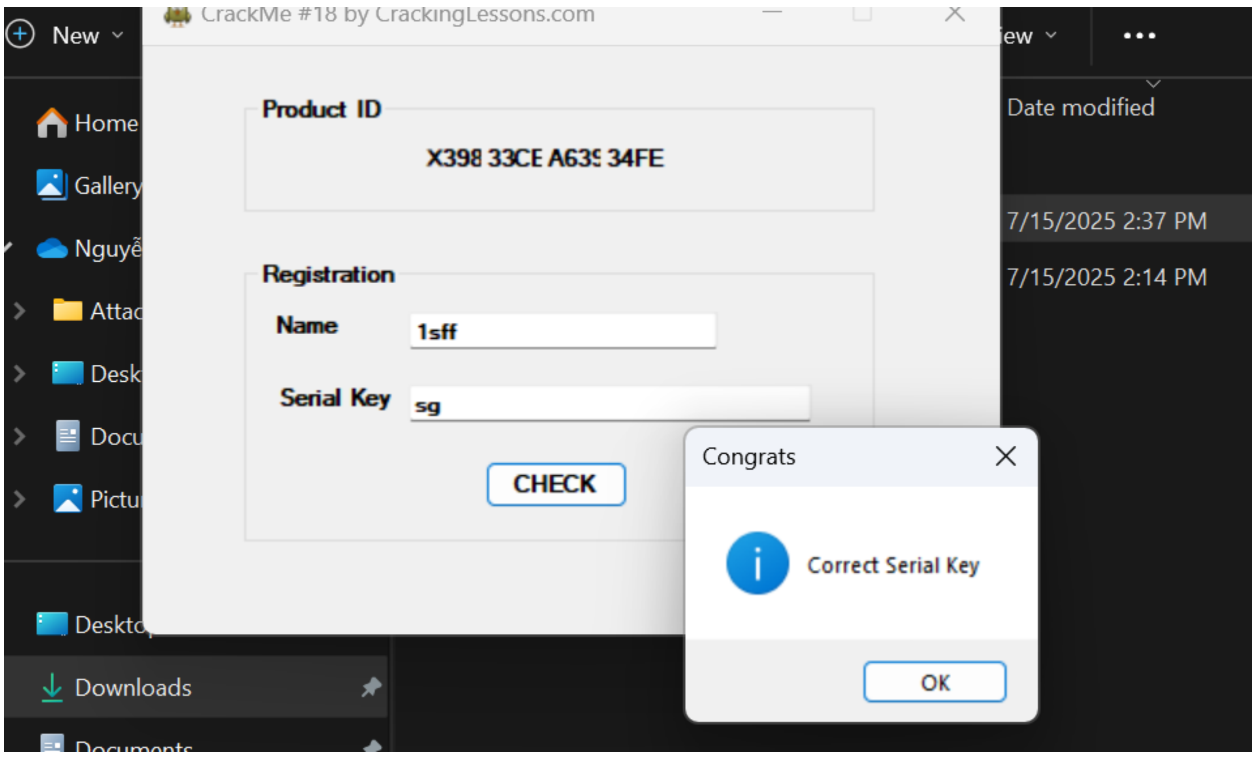
```csharp
// Token: 0x06000002 RID: 2 RVA: 0x00002060 File Offset: 0x00000260
private void btnCheck_Click(object sender, EventArgs e)
{
    string text = this.txtName.Text.Substring(0, 4).ToUpper();
    string text2 = this.txtSerialKey.Text;
    string text3 = this.productId1.Text;
    string text4 = this.productId2.Text;
    string text5 = this.productId3.Text;
    if (text.Length < 4)
    {
        MessageBox.Show("Name should be at least 4 characters", "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
        return;
    }
    string value = text3 + text + text4 + text5;
    if (text2.Equals(value))
    {
        MessageBox.Show("Correct Serial Key", "Congrats", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        return;
    }
    MessageBox.Show("Wrong Serial Key", "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
```

I change the text.Length < 0 and delete the IF block of Correct status with Incorrect satus like this and patch to check:

NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302



```
    string text4 = this.productId3.Text;
    if (text.Length < 0)
    {
        MessageBox.Show("Name should be at least 4 characters", "Error", MessageBoxButtons.OK, MessageBoxIcon.Hand);
        return;
    }
    text2 + text + text3 + text4;
    MessageBox.Show("Correct Serial Key", "Congrats", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
}
```



CrackMe #18 by CrackingLessons.com

Product ID

X398 33CE A639 34FE

Registration

Name: 1sff

Serial Key: sg

CHECK

Congrats

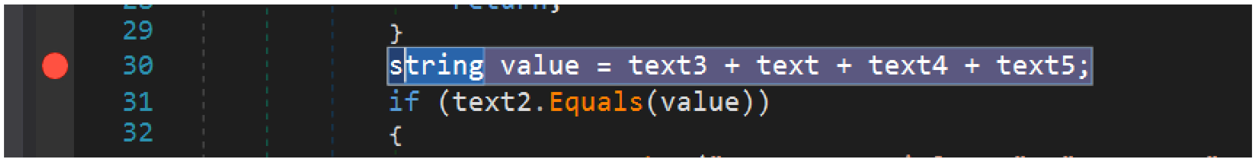
Correct Serial Key

OK

Done the mission 1.

Next to challenge 2: serial fishing

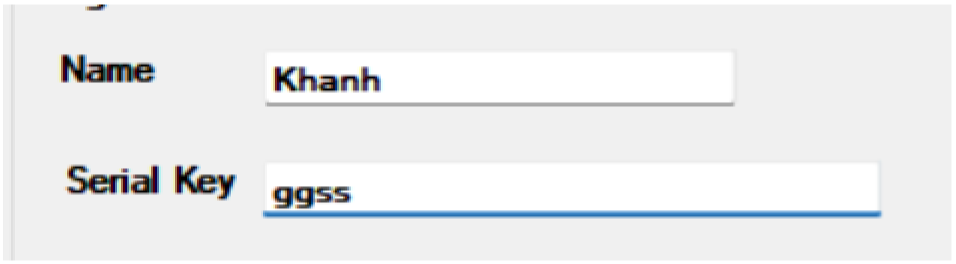
Add breakpoint to this line to view the change of String compare after this.



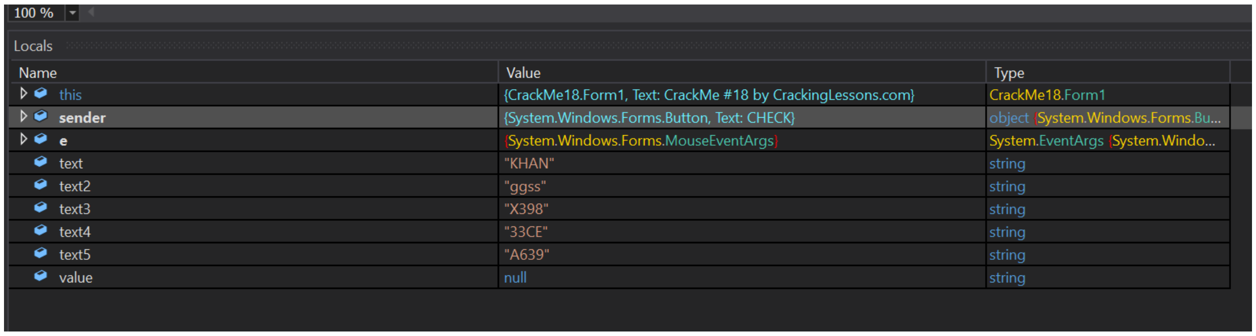
```
}
string value = text3 + text + text4 + text5;
if (text2.Equals(value))
{
```

Run and see that:

NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302



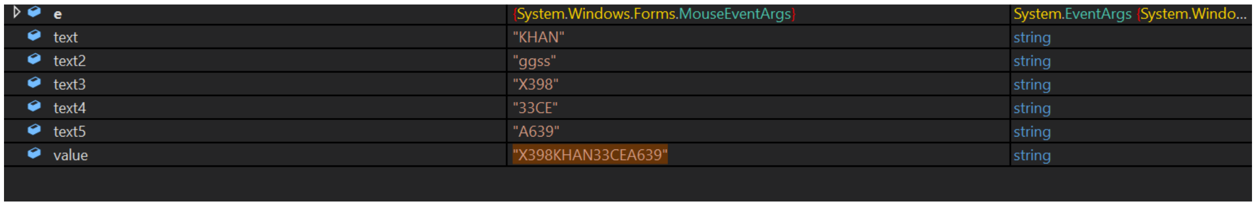
Name	Khanh
Serial Key	ggss



Locals

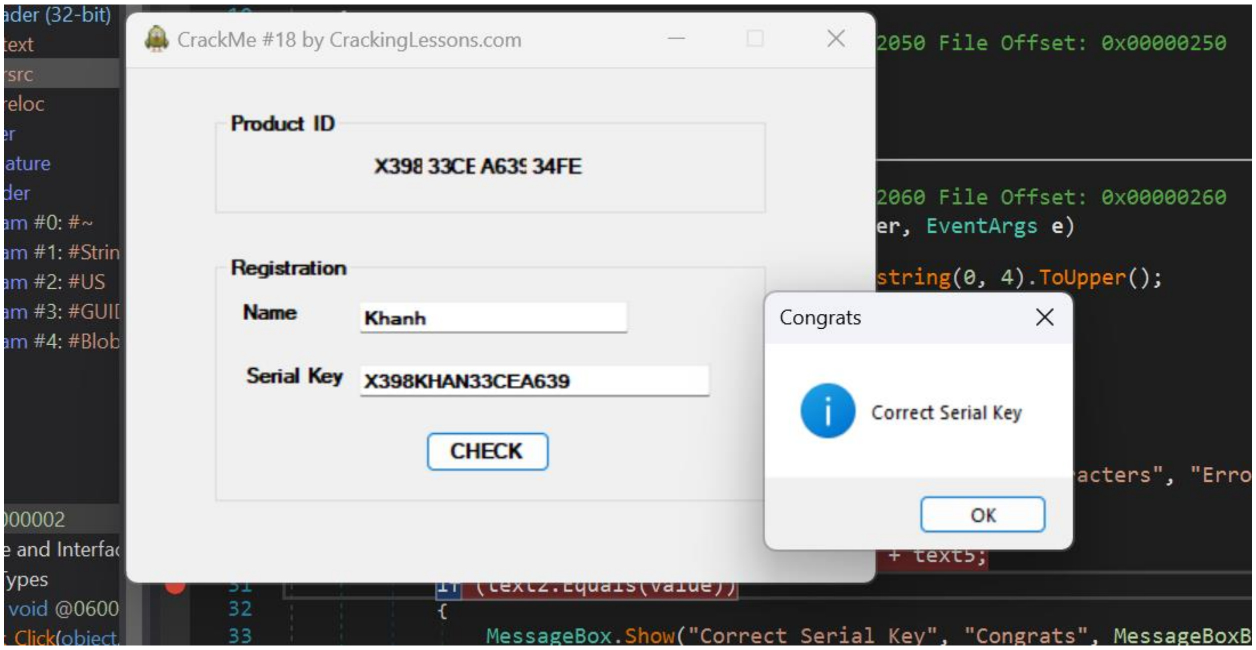
Name	Value	Type
this	{CrackMe18.Form1, Text: CrackMe #18 by CrackingLessons.com}	CrackMe18.Form1
sender	{System.Windows.Forms.Button, Text: CHECK}	object {System.Windows.Forms.Bu...
e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs {System.Windo...
text	"KHAN"	string
text2	"ggss"	string
text3	"X398"	string
text4	"33CE"	string
text5	"A639"	string
value	null	string

Run to the IF compare to see the Value variable. This is the exact fishsing key we need:



e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs {System.Windo...
text	"KHAN"	string
text2	"ggss"	string
text3	"X398"	string
text4	"33CE"	string
text5	"A639"	string
value	"X398KHAN33CEA639"	string

That is the fishing code:



CrackMe #18 by CrackingLessons.com

Product ID

X398 33CE A639 34FE

Registration

Name: Khanh

Serial Key: X398KHAN33CEA639

CHECK

Congrats

Correct Serial Key

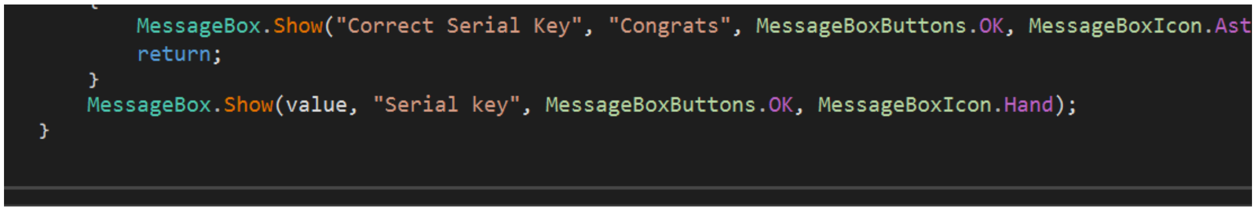
OK

```
2050 File Offset: 0x00000250
2060 File Offset: 0x00000260
er, EventArgs e)
string(0, 4).ToUpper();
acters", "Erro
+ text5;
if (text2.Equals(value))
{
    MessageBox.Show("Correct Serial Key", "Congrats", MessageBoxB
```

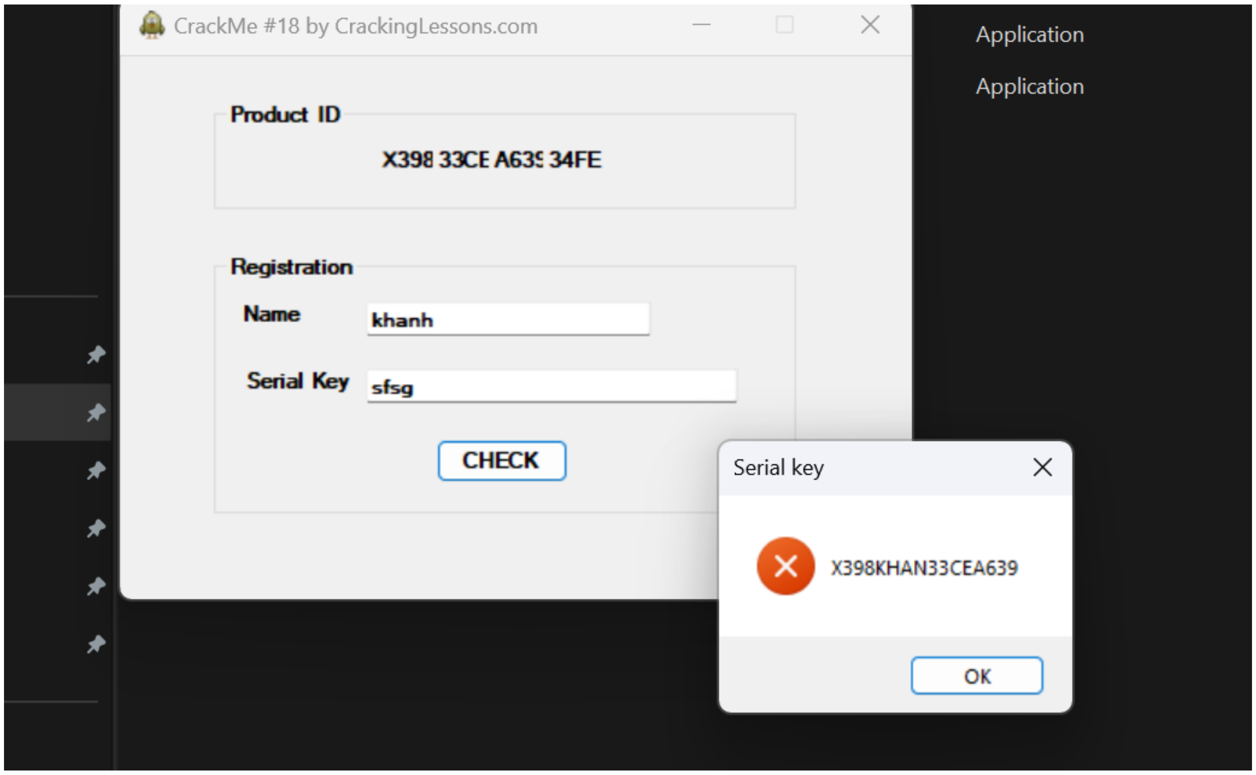
Next the final challenge:

NGUYEN NAM KHANH – HE1159 – IA1902 – IAM302

Create keygen means display keygen to the box, so i change the box to display keygen like this:



```
            MessageBox.Show("Correct Serial Key", "Congrats", MessageBoxButtons.OK, MessageBoxIcon.Ast
            return;
        }
        MessageBox.Show(value, "Serial key", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    }
```



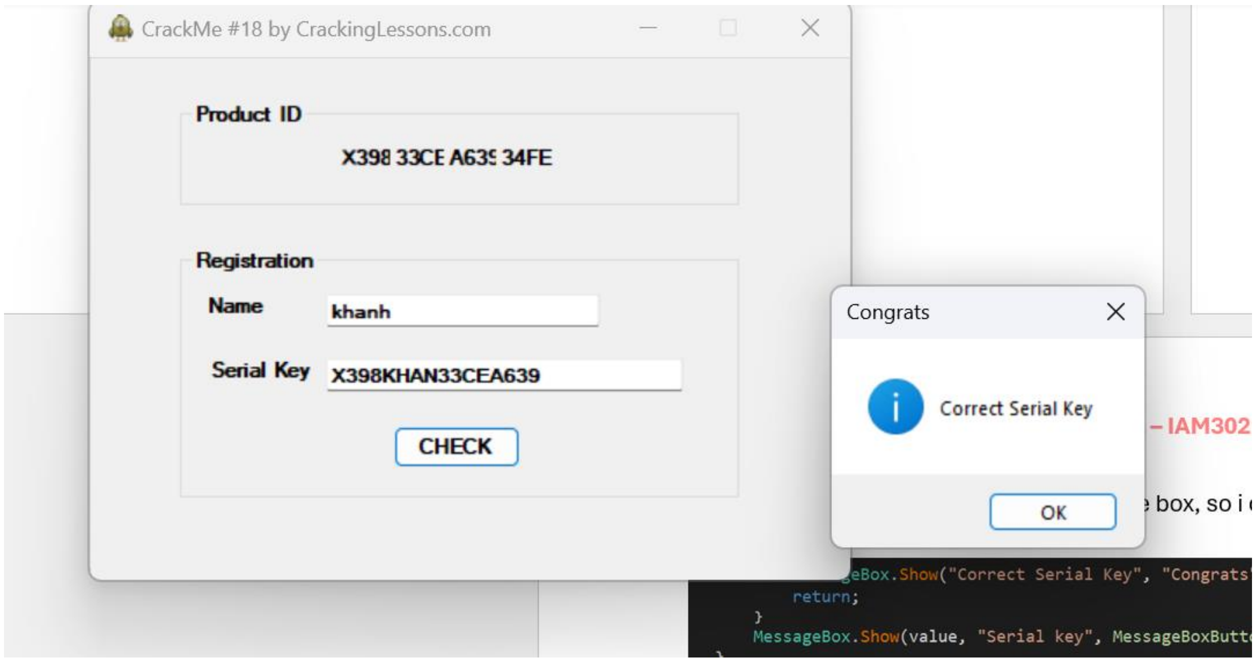
CrackMe #18 by CrackingLessons.com

Product ID: X398 33CE A639 34FE

Registration
Name: khanh
Serial Key: sfsg

Serial key

X398KHAN33CEA639



CrackMe #18 by CrackingLessons.com

Product ID: X398 33CE A639 34FE

Registration
Name: khanh
Serial Key: X398KHAN33CEA639

Congrats

Correct Serial Key