

How to recover injured ESE files

Table of Contents

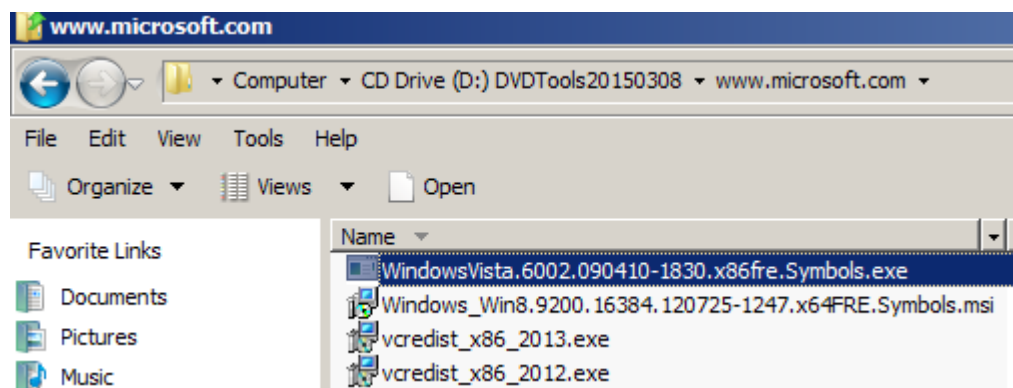
How to recover injured ESE files.....	1
Required applications.....	2
Install the symbol files.....	2
Active directory.....	4
Simulate the injure of the first 128 bytes of the ntds.dit file.....	4
Simulate the injure of the first 8192 bytes of the ntds.dit file.....	9
Simulate the injure of the first 16384 bytes of the ntds.dit file.....	16
Find the Highest DB time.....	22
Find the Checksum.....	23

Required applications

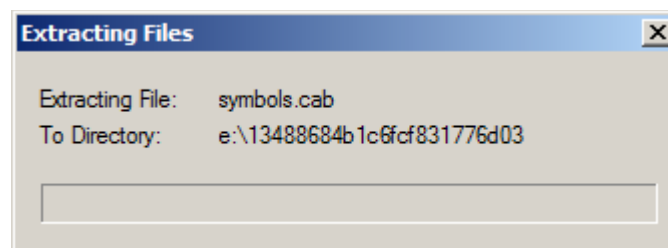
- Domain controller
- Hex editor
- olly debugger
- api monitor
- symbol files for the operating system
- python interpreter

Install the symbol files

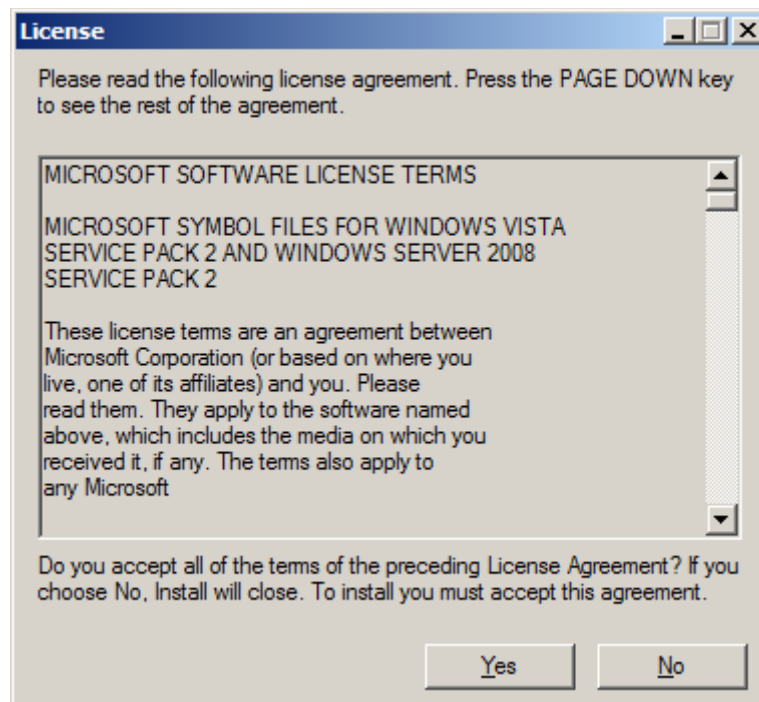
Start the symbol file installer



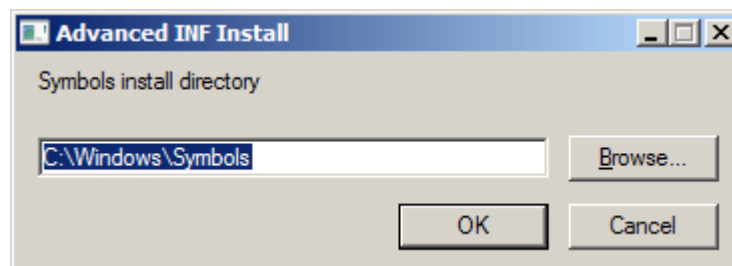
Wait until it extracts the setup files:



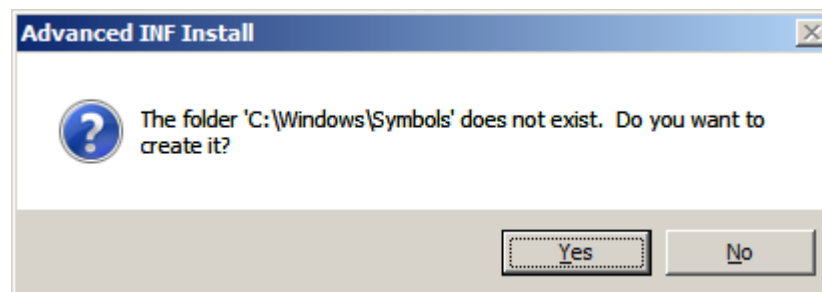
Accept the licence agreement:



Define the destination directory, the default is c:\Windows\Symbols and click to the OK button



Click yes, to create the required directory:



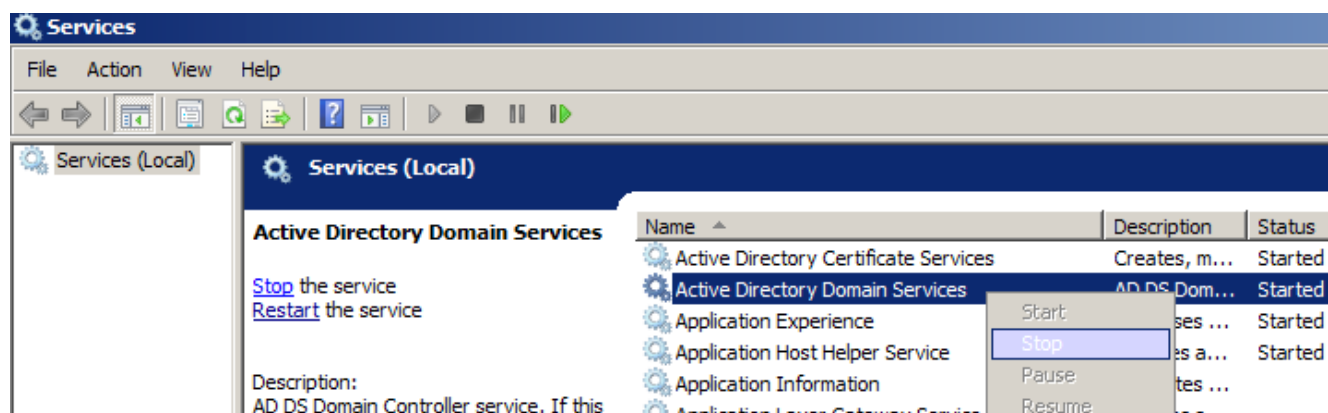
The ESE (Enhanced Storage Engine) file format (you may call it as JET database) used at many places in Microsoft windows like active directory, exchange, search engine, and so on. Because its wildely used important to know, how to restore it in case of a data loss event (OK, of course everyone has an up to date and working backup, but just for the time you may not).

Let us start with the Active directory file.

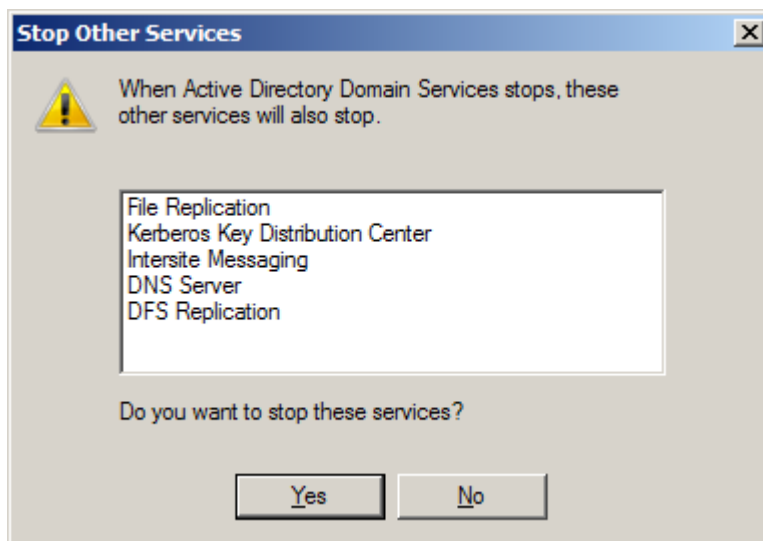
Active directory

Simulate the injure of the first 128 bytes of the ntds.dit file

To be able to manypulate the files of the active directory first you must stop the active directory service. To do it start the **administrative tools \ Services**, **right click to the Active Directory Domain Services** and from the popup menu select the **Stop** command.



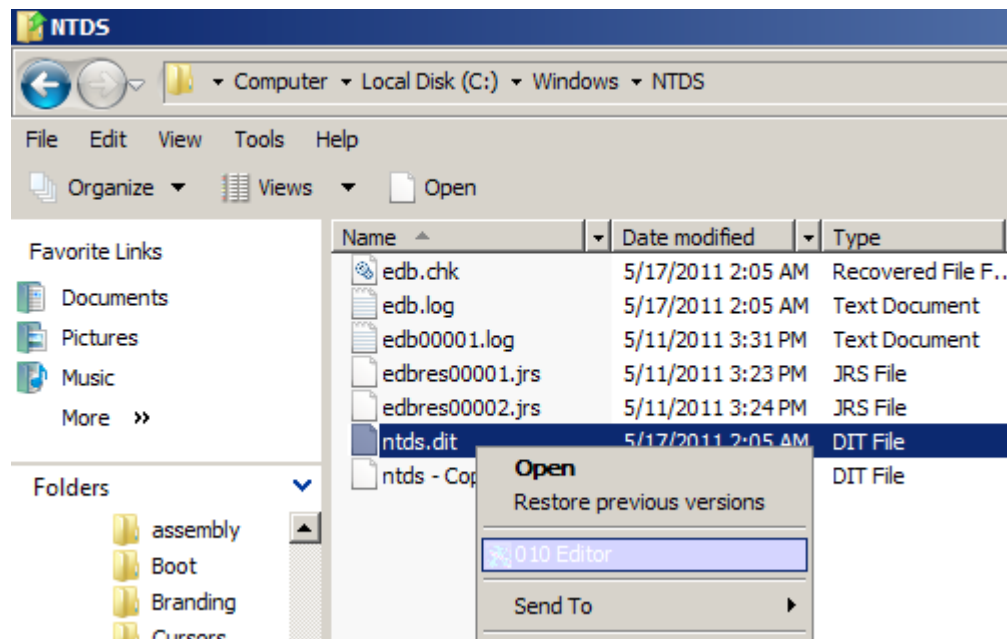
The operating system warns you, it will stop some dependent services as well, just click to the **Yes** button, to stop those too.



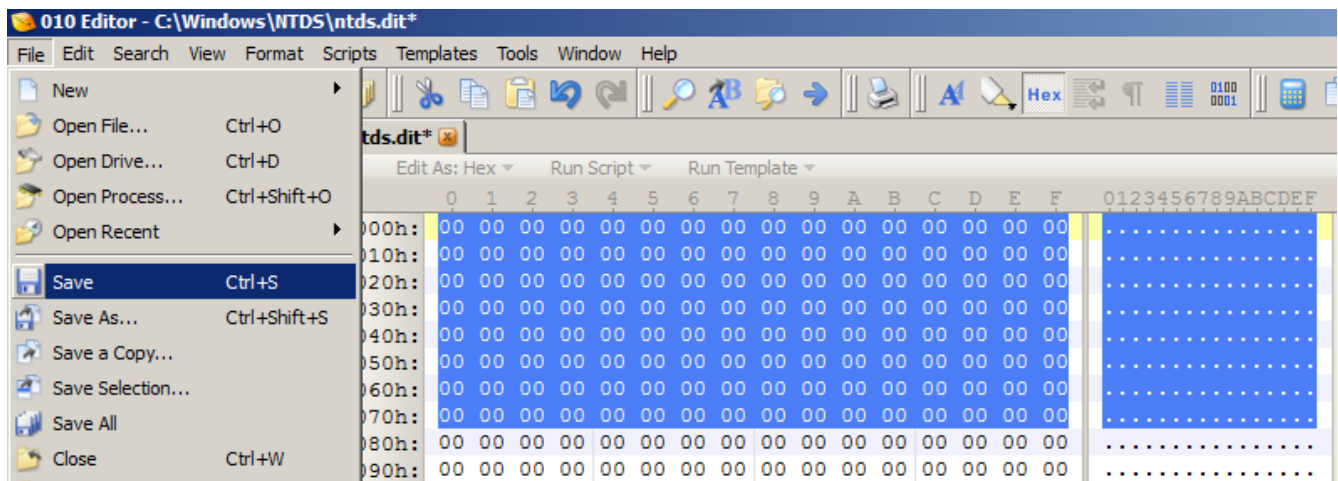
I recommend you to always create a copy of the file we work on, what can be used, if something goes wrong. So first **create a safe copy form the c:\windows\ntds\ntds.dit** file.

Then **install some hex editor**, I will use the 010 editor.

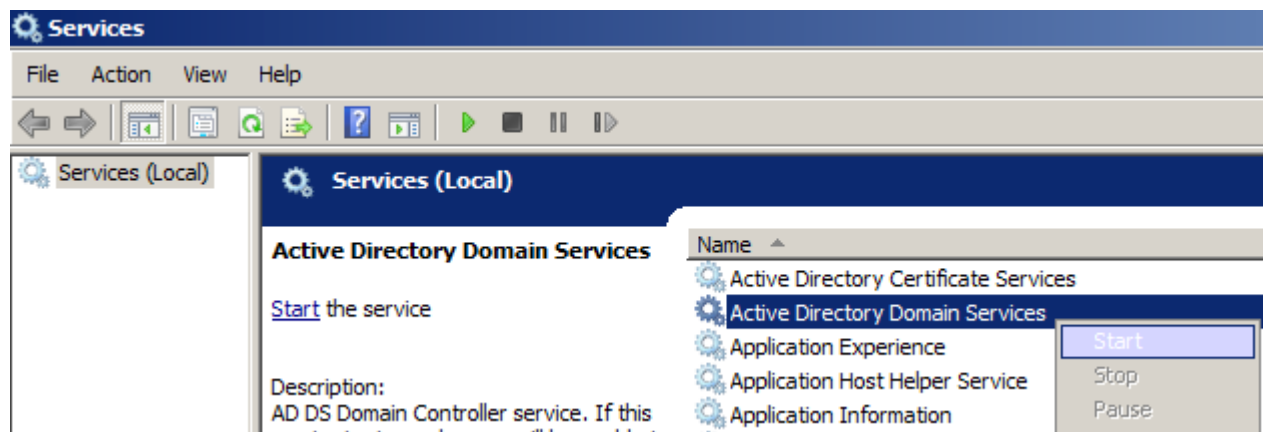
So I right click to the **c:\windows\ntds\ntds.dit** file, and **open it with my hex editor**.



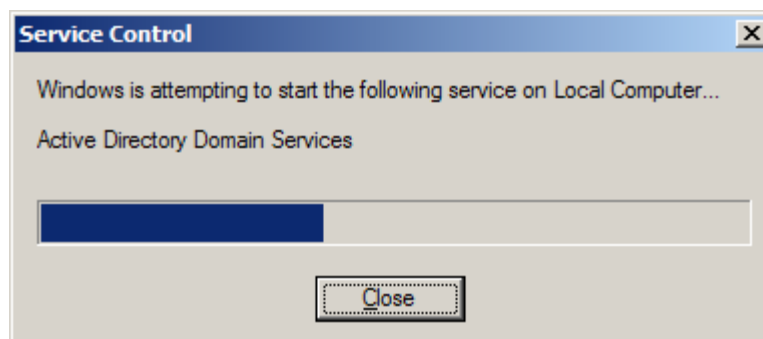
Now we start to simulate the injure of the file by overwriting some bytes. **Go to the beginning of the file, and select Edit \ Insert/Overwrite \ Overwrite Bytes...**



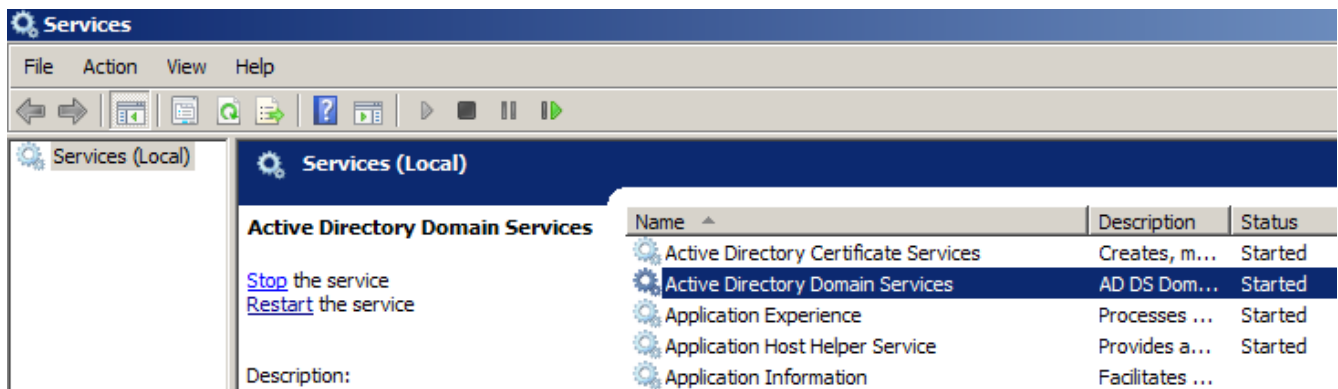
Now try to start the active directory domain services by starting the **administrative tools \ Services**, then **right click to the Active Directory Domain Services**, and from the popup menu select the **Start** command



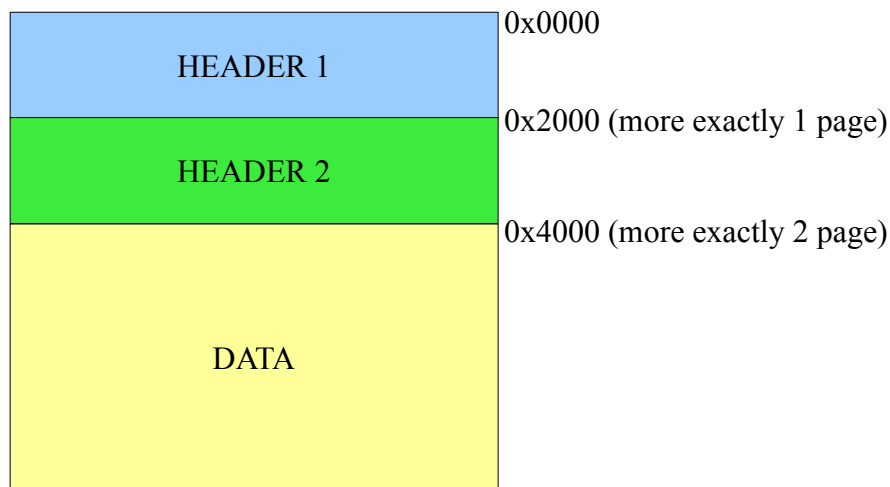
The Service tries to start



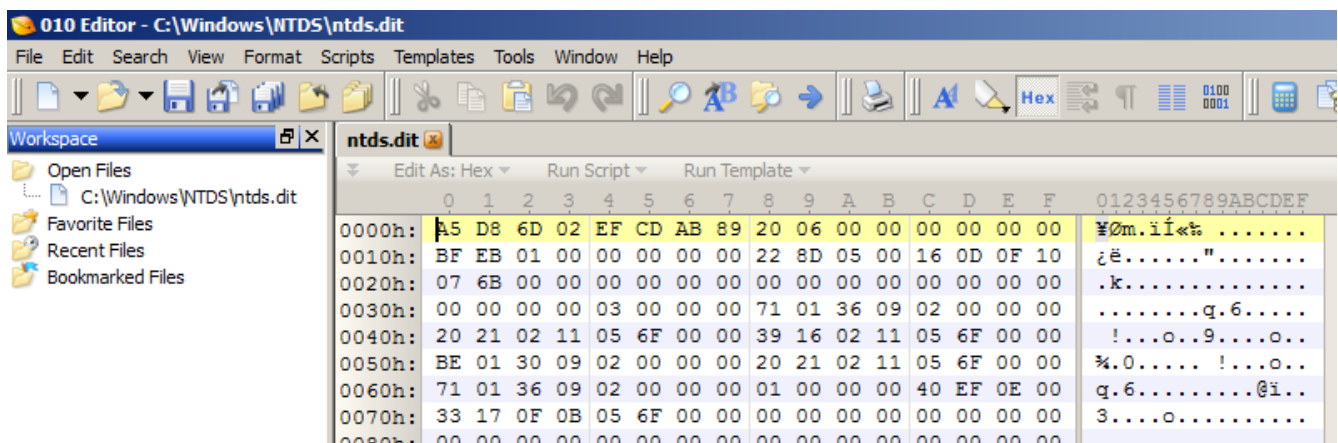
And finally...
it starts:



Now one may ask, how can it start, if we destroyed the first 128 bytes of it? What happens? The answer is simple, the ESE files has two headers, both of them 8192 bytes at the beginning of the file. (more exactly one header is one page long, and the page size can vary in case of ESE files, but in case on ntds.dit it is always 8 kB.)

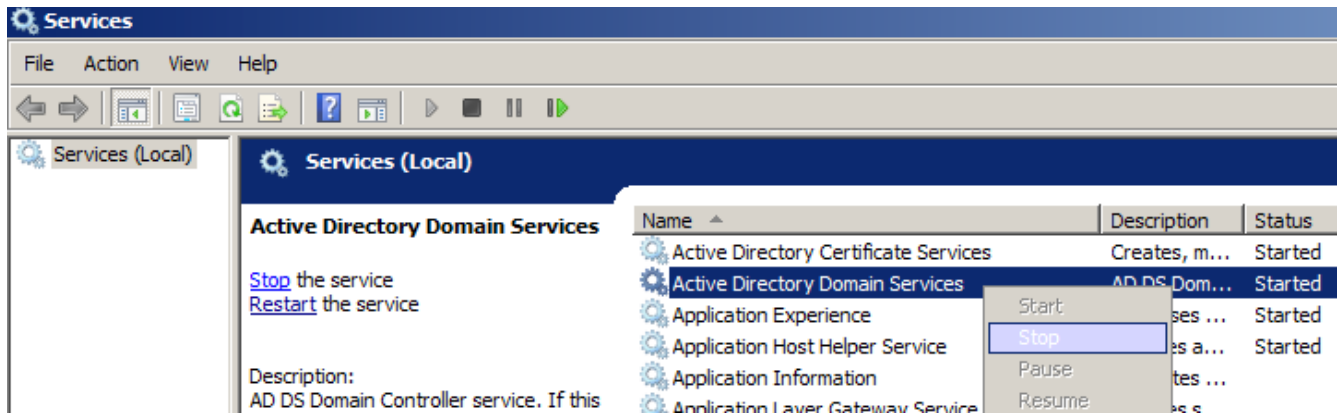


So when we destroyed the first 128 bytes only the first header was destroyed, and the application system was able to correct it by the help of the second header.

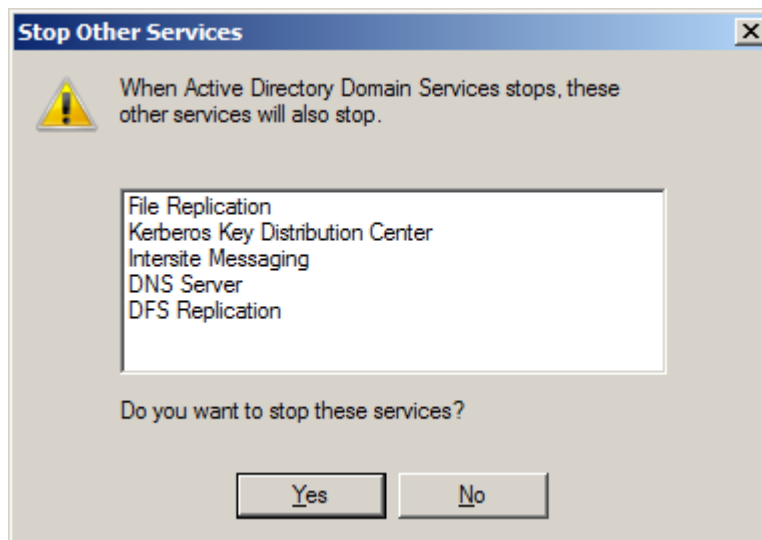


Simulate the injure of the first 8192 bytes of the ntds.dit file

Again, to be able to manipulate the files of the active directory first you must stop the active directory service. To do it start the **administrative tools \ Services**, right click to the **Active Directory Domain Services** and from the popup menu select the **Stop** command.

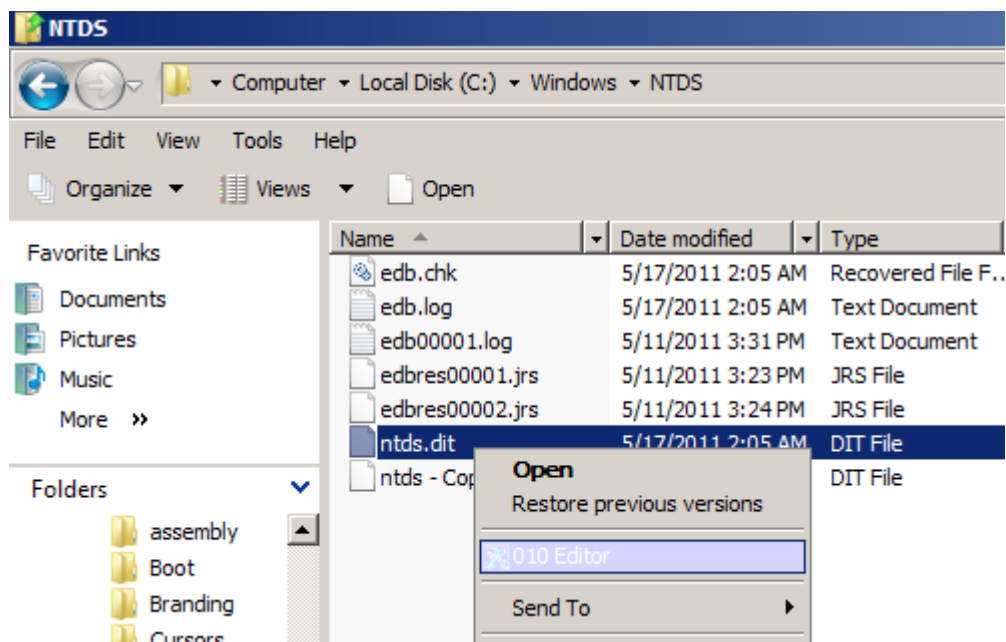


The operating system warns you, it will stop some dependent services as well, just click to the **Yes** button, to stop those too.

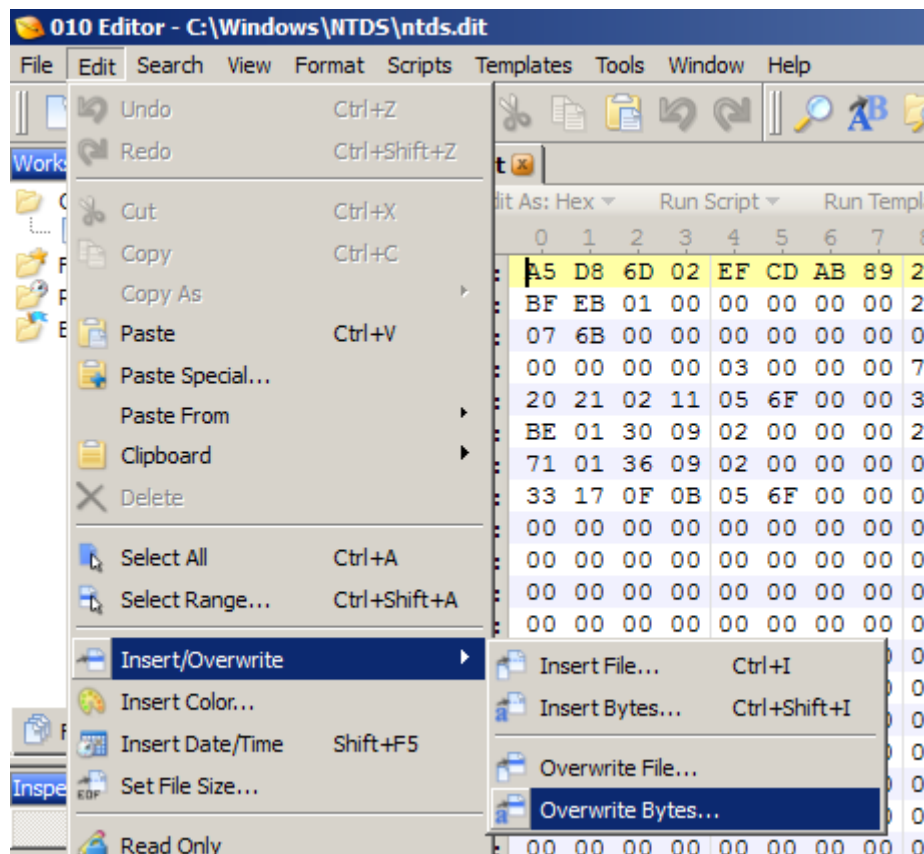


I recommend you to always create a copy of the file we work on, what can be used, if something goes wrong. So first **create a safe copy form the c:\windows\ntds\ntds.dit** file. (I recommend to create a new one, because we restarted the service, and the file modified because of that).

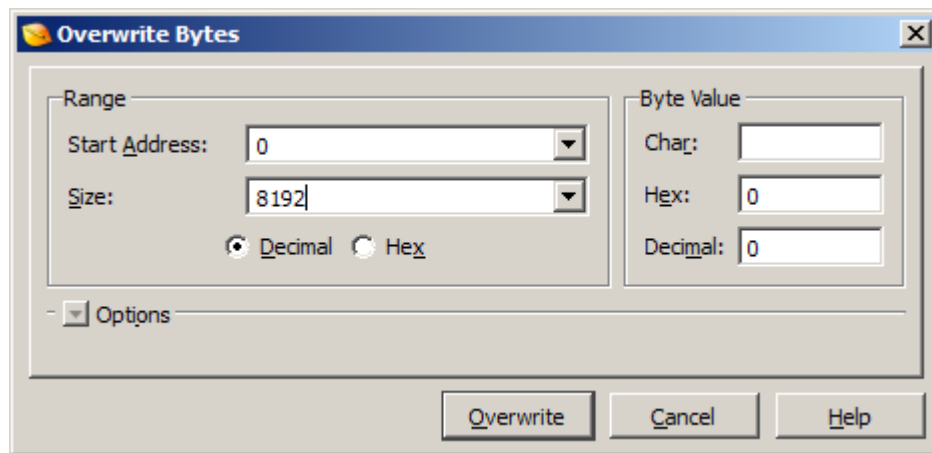
Right click to the **c:\windows\ntds\ntds.dit** file, and **open it with a hex editor**.



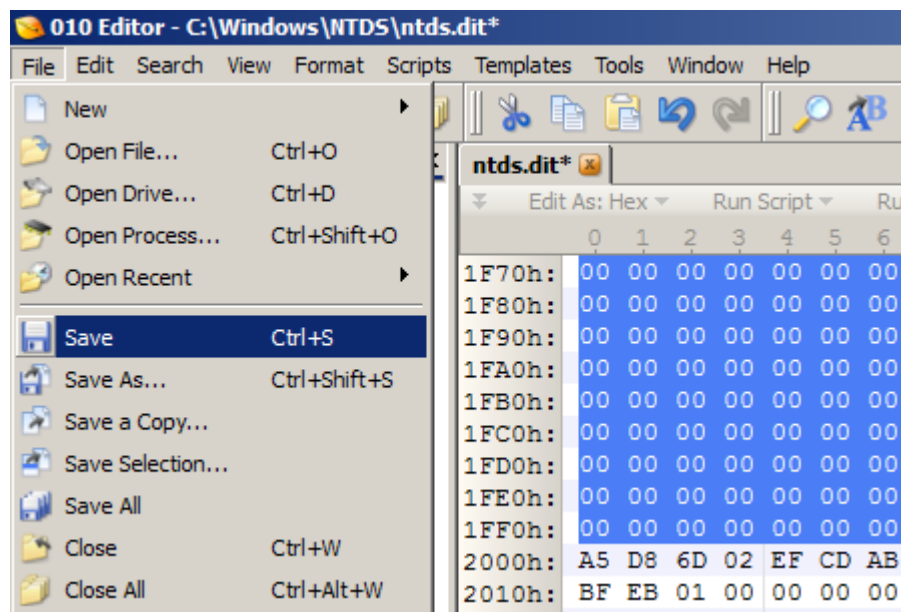
Now we start to simulate the injure of the file by overwriting the first header (8192 bytes). **Go to the beginning of the file, and select Edit \ Insert/Overwrite \ Overwrite Bytes...**



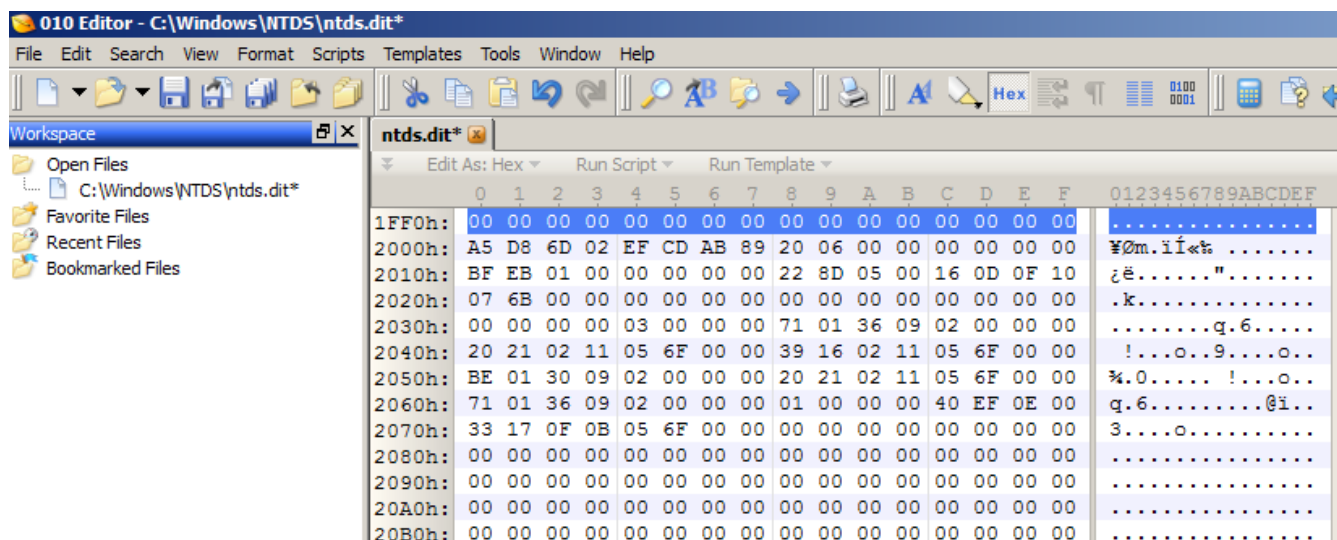
In the popup window **define the size 8192 byte**, and **set the byte value to hex 0**, to overwrite the first 8192 bytes of the file with null bytes. Then click to the **Overwrite** button.



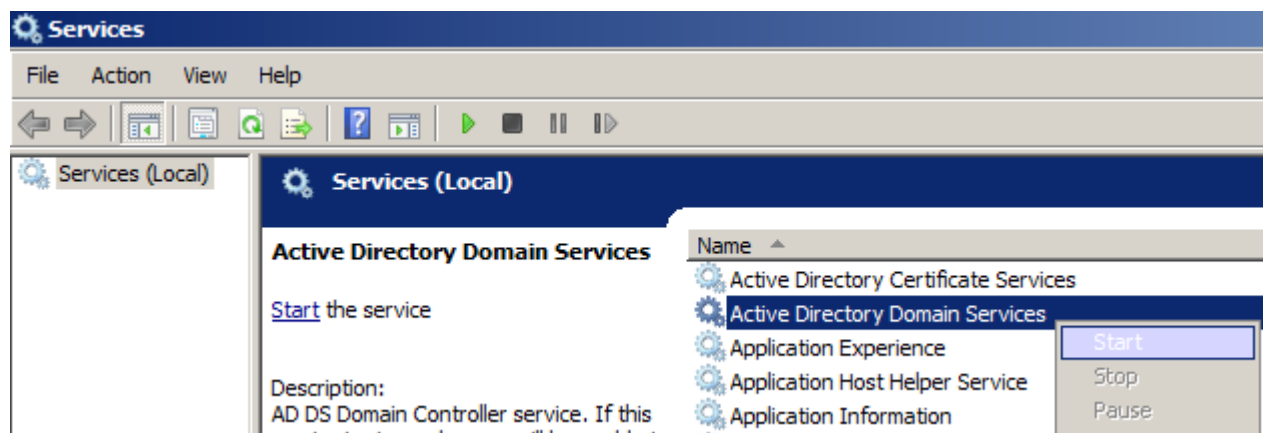
Then save the file, select the **File / Save** command.



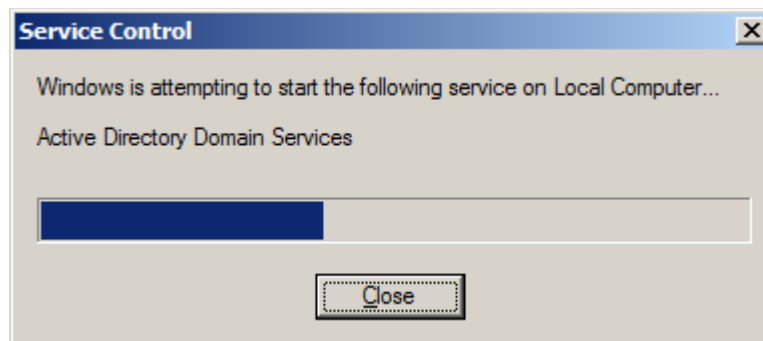
The second header remain intact:



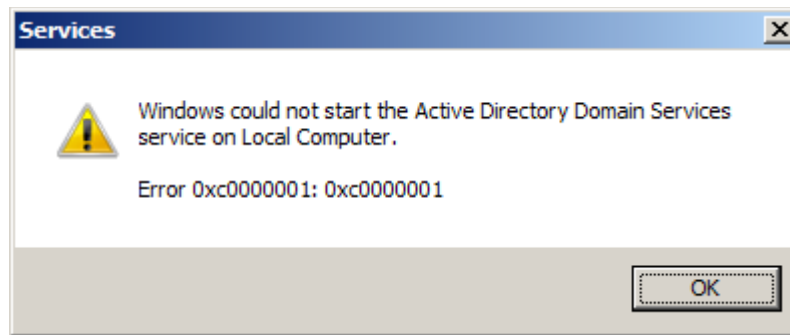
Now try to start the active directory domain services by starting the **administrative tools \ Services**, then **right click to the Active Directory Domain Services**, and from the popup menu select the **Start** command



The Service tries to start



And finally...
It can not start:



OK, previous it surprised us, because it simply corrected the error, now there is another surprise, if there are two headers, why it can not repair it now again?

The answer is simple again, just we should examine the structure of the header shown in the next picture (all the numbers must be stored in little endian format):

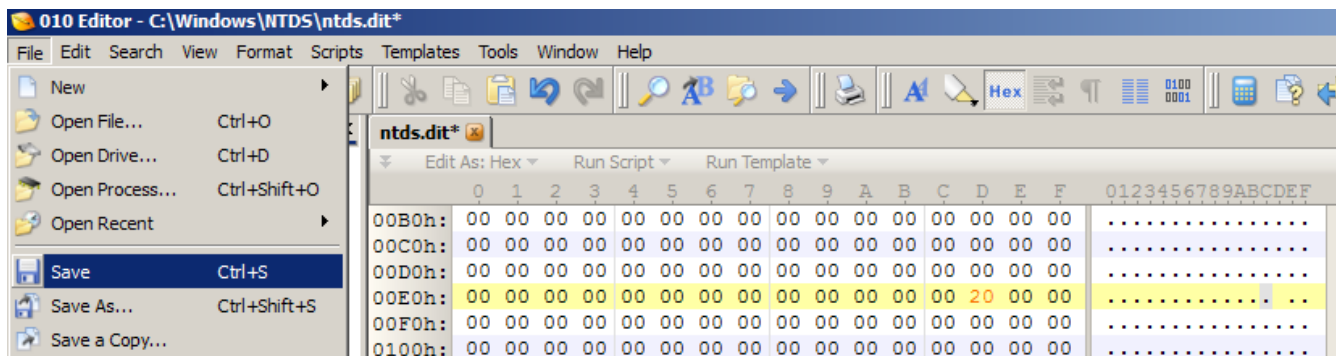
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0000	Header checksum (MANDATORY)				File ulMagic 0x89abcdef (MANDATORY)				File format verison 0x00000620 (MANDATORY)				File type 0x00000000 (MANDATORY) [0: db, 1: stream]			
0x0010	Highest DB time (MANDATORY)								RAND ? (NOT mandatory)				DB signature time: sec, min			
0x0020	hour, day, month, year (1990=0) (NOT mandatory)															
0x0030					DB state 0x00000003 (MANDATORY) [1: just created 2: dirty 3: clean 4: database is being upgraded 5: ForceDetach]				log position that was used when the database was last brought to a clean shutdown state or NULL if the database is in a dirty state (NOT mandatory) [block: 2byte, sector: 2byte, generation: 4 byte]							
0x0040	Last Consistent time cont sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Last Attach time sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x0050	log position that was used the last time the database was attached If 0 dirty (NOT mandatory)								Last Detach time sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x0060	log position that was used the last time the database was detached If 0 dirty (NOT mandatory)								DBID ? (NOT mandatory)				RAND ? (NOT mandatory)			
0x0070	Log signature time: sec, min, hour, day, month, year (0=1900) (NOT mandatory)															
0x0080									Consists of a log position ?at full backup? (NOT mandatory)							
0x0090	Previous full backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Generation lower number The lower log generation number associated with the backup (NOT mandatory)							
0x00A0	Consists of a log position ?at incremental backup? (NOT mandatory)								Previous incremental backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x00B0	Generation lower number The lower log generation number associated with the backup (NOT mandatory)								Consists of a log position ?at current full backup? (NOT mandatory)							
0x00C0	Current full backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Generation lower number The lower log generation number associated with the backup (NOT mandatory)							
0x00D0	Shadow disabled				Last object ID (NOT mandatory)				Major version (NOT mandatory)				Minor version (NOT mandatory)			
0x00E0	Build number (NOT mandatory)				Servicepack (NOT mandatory)				file format version				pagesize 0x00002000 (MANDATORY)			
0x00F0	repair count				Recovery Time (NOT mandatory)											
0x0100																
0x0110																
0x0120																
0x0130																
0x0140																
0x0150					File format verison ? (NOT mandatory)				File format verison minor? (NOT mandatory)							

Previously we overwrite the first 128 bytes (0x0080) bytes. Fortunately it was not containing a very important information, the page size at position 0x00EC..0x00EF. This is why previously the operating system was able to correct it. Now the page size is overwritten, so the operating system can not find out the size of the header so it can not find the secondary header, and if it did not find the secondary header of course it can not use it. *(In case of ntds.dit the page size is always 8192 bytes as I mentioned*

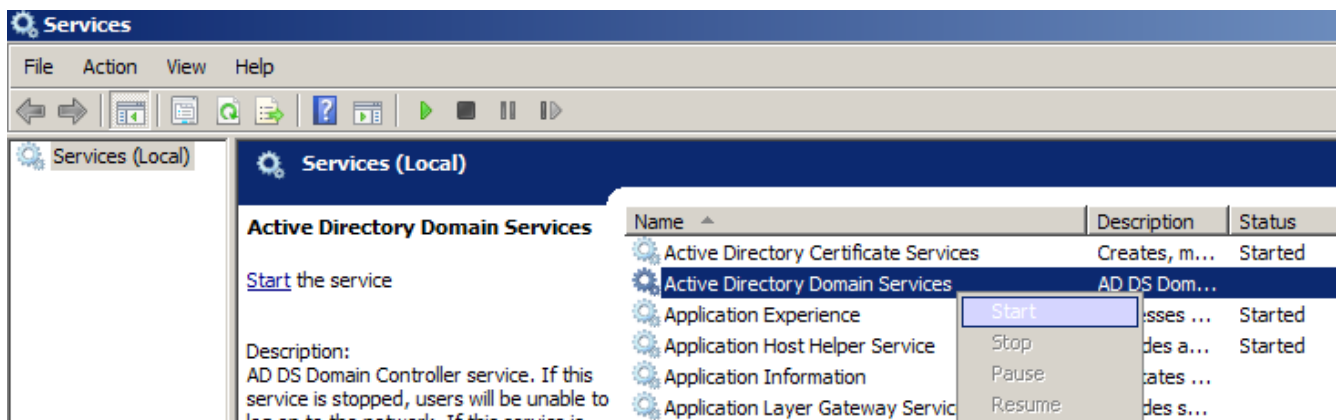
already, so theoretically it could find it, but as one can see practically the operating system not use this assumption.)

Now we know the problem, the page size at position 0x00EC..0x00EF is overwritten. And the idea is that, if we write back there the correct page size (8192 byte) it should start to work.

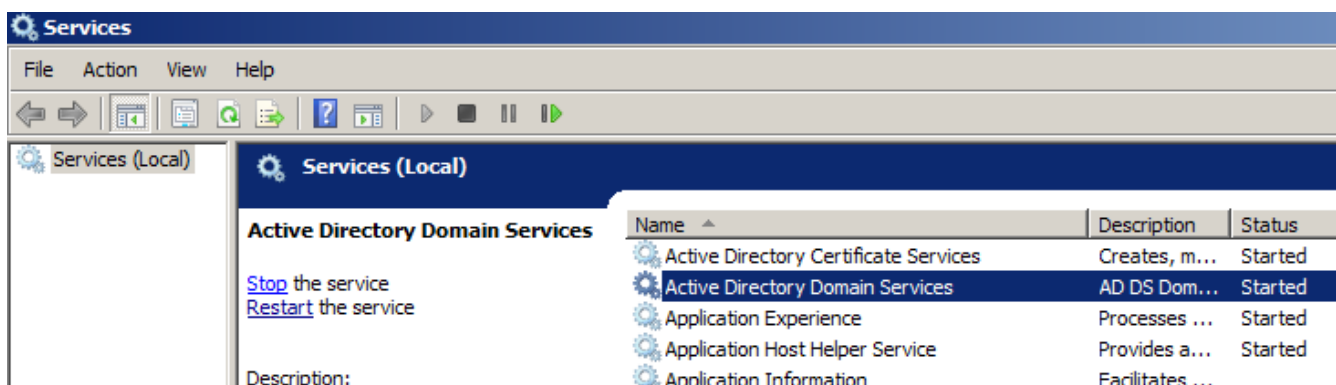
Let us try it in practice. **Open the ntds.dit in your hex editor**, and go to the **position 0xEC**, and write there the value 8192 (**0x2000**) in **little endian** notation, then click to the **File \ Save**, to save the changes:



Now try to start the service again.

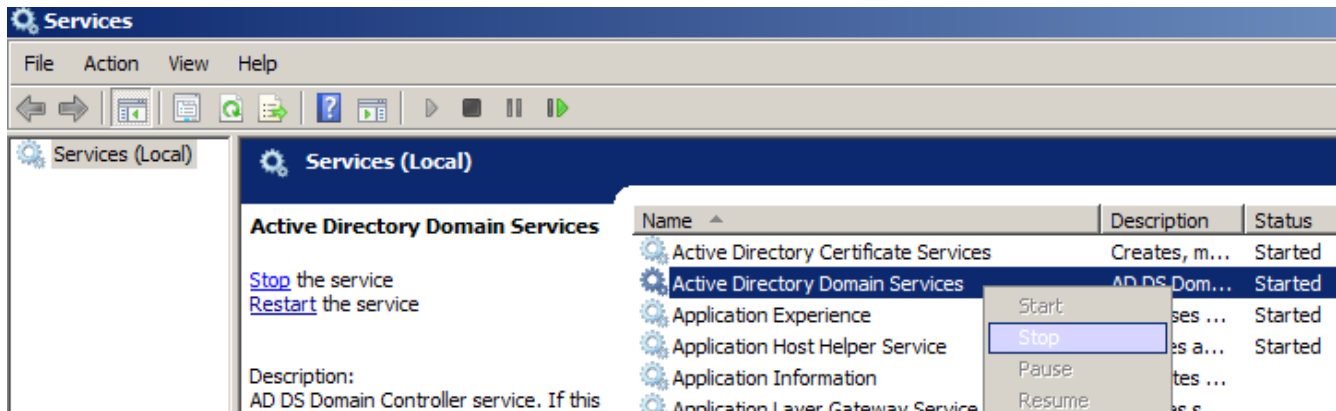


And it starts as expected. (Nice if the practice follows the theory).

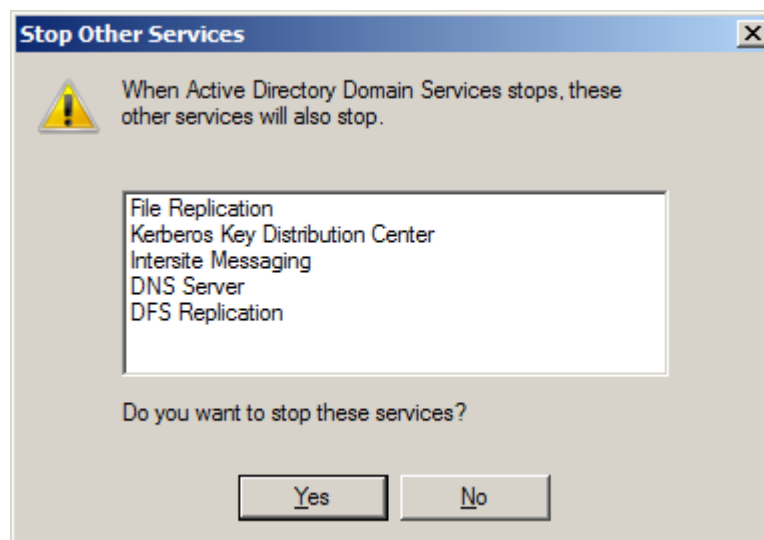


Simulate the injure of the first 16384 bytes of the ntds.dit file

Again, to be able to manipulate the files of the active directory first you must stop the active directory service. To do it start the **administrative tools \ Services**, right click to the **Active Directory Domain Services** and from the popup menu select the **Stop** command.

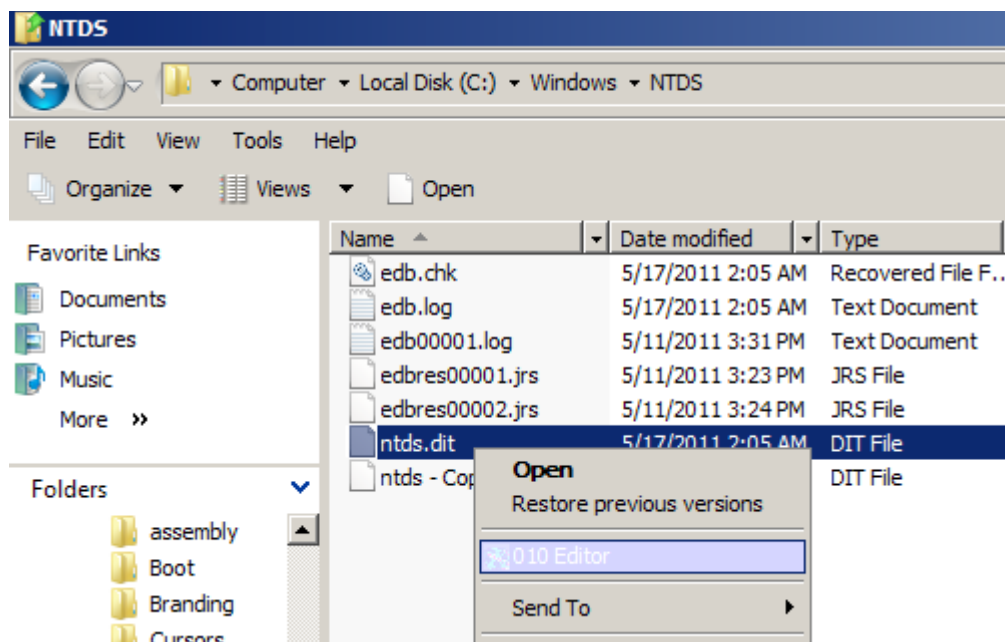


The operating system warns you, it will stop some dependent services as well, just click to the **Yes** button, to stop those too.

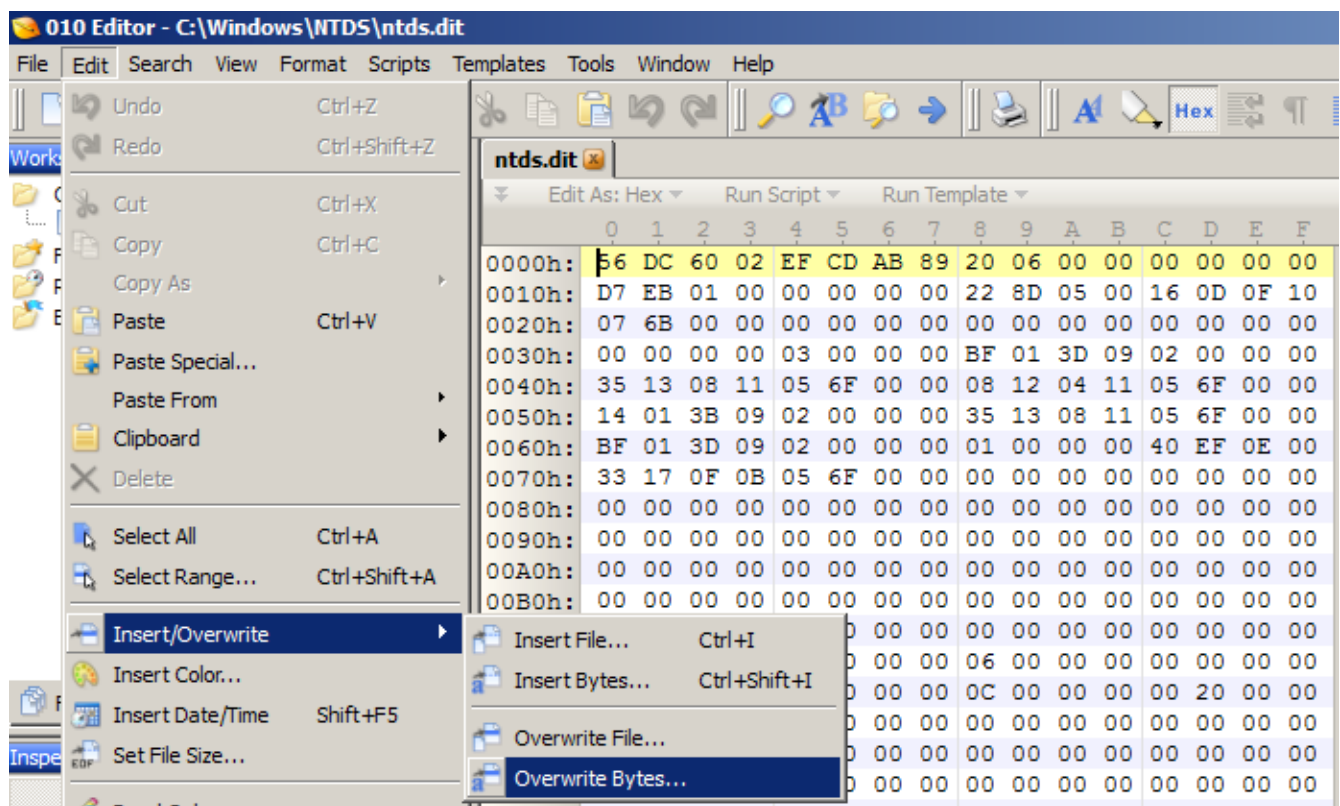


I recommend you to always create a copy of the file we work on, what can be used, if something goes wrong. So first **create a safe copy form the c:\windows\ntds\ntds.dit** file. (I recommend to create a new one, because we restarted the service, and the file modified because of that).

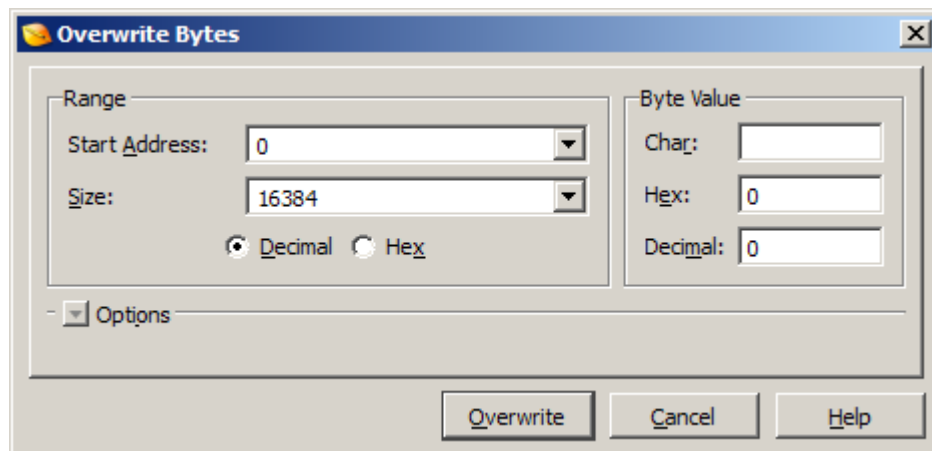
Right click to the **c:\windows\ntds\ntds.dit** file, and **open it with a hex editor**.



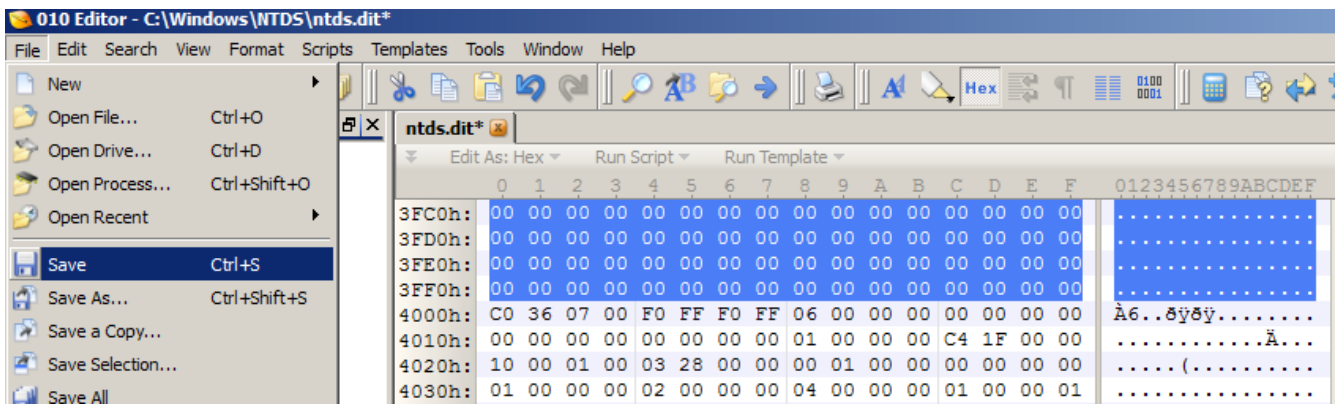
Now we start to simulate the injure of the file by overwriting both headers (16384 bytes). **Go to the beginning of the file, and select Edit \ Insert/Overwrite \ Overwrite Bytes...**



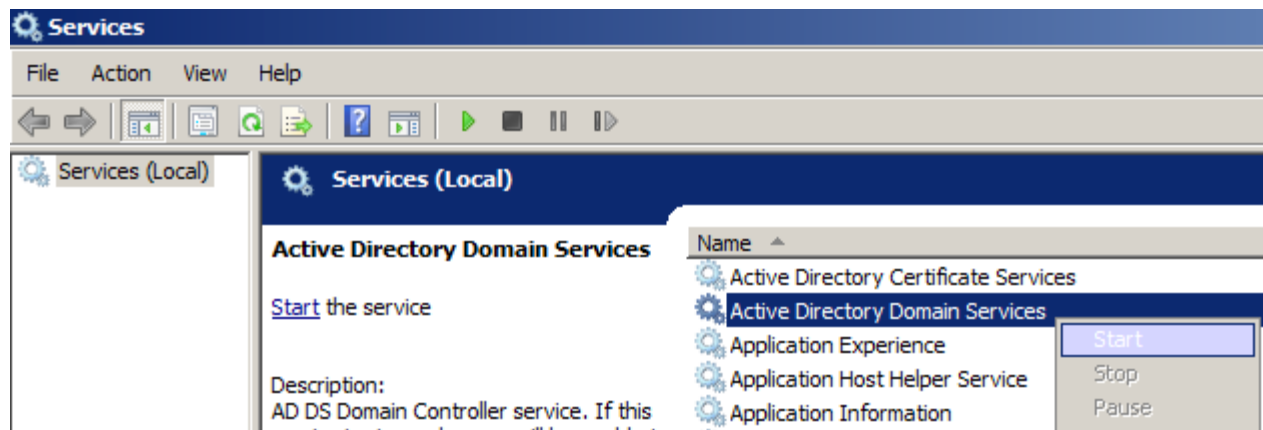
In the popup window **define the size 16384 byte**, and **set the byte value to hex 0**, to overwrite the first 16384 bytes of the file with null bytes. Then click to the **Overwrite** button.



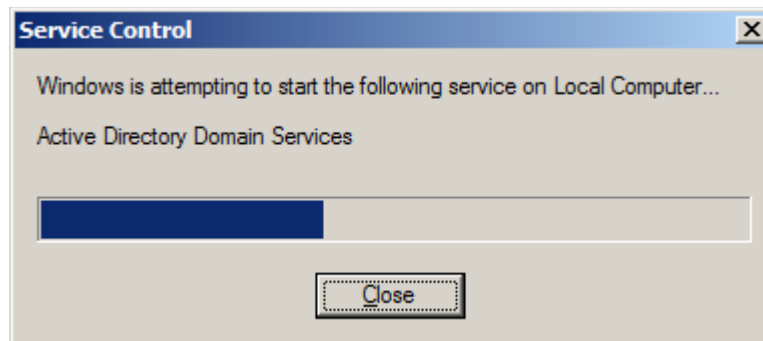
Then save the file, select the **File / Save** command.



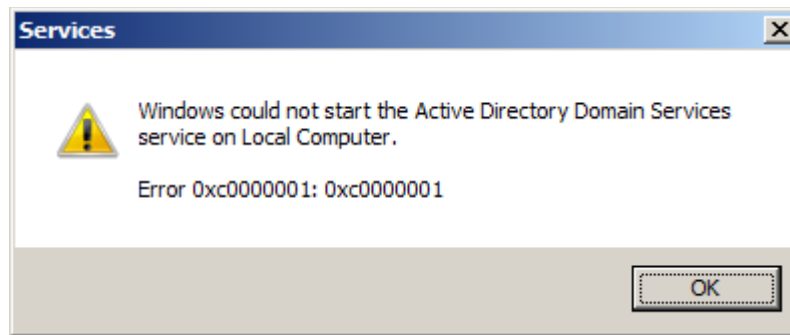
Now try to start the active directory domain services by starting the **administrative tools \ Services**, then **right click to the Active Directory Domain Services**, and from the popup menu select the **Start** command



The Service tries to start



It can not start as we expected:



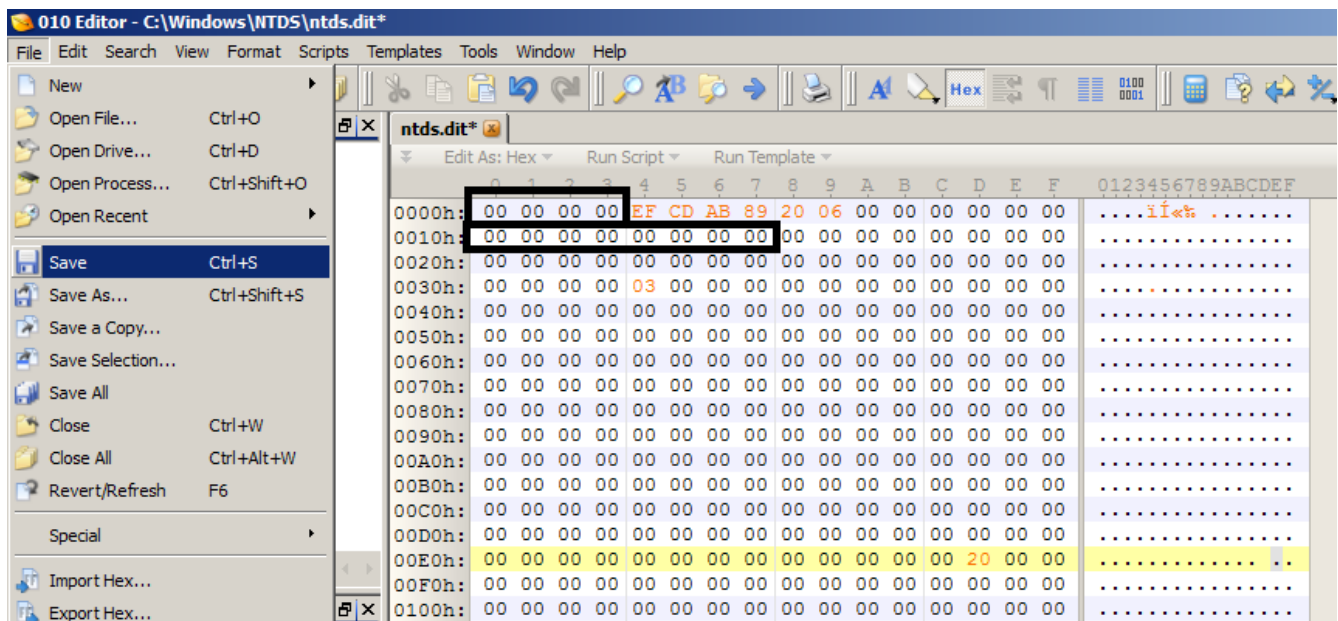
Now it is not enough to correct the page size parameter, but we must correct all the mandatory parameters. There are seven such a parameter altogether:

- **Header checksum**
- Magic (constant 0x89ABCDEF)
- File format version (constant 0x00000620)
- File type (0x00000000)
- **Highest DB time**
- DB state (0x00000003)
- Page size (0x00002000)

Fortunately from these seven parameters five are constant, so we know them, only two must be adjusted to the actual ntds.dit file:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0000	Header checksum (MANDATORY)				File ulMagic 0x89abcdef (MANDATORY)				File format verison 0x00000620 (MANDATORY)				File type 0x00000000 (MANDATORY) [0: db, 1: stream]			
0x0010	Highest DB time (MANDATORY)								RAND ? (NOT mandatory)				DB signature time: sec, min			
0x0020	hour, day, month, year (1990=0) (NOT mandatory)															
0x0030					DB state 0x00000003 (MANDATORY) [1: just created 2: dirty 3: clean 4: database is being upgraded 5: ForceDetach]				log position that was used when the database was last brought to a clean shutdown state or NULL if the database is in a dirty state (NOT mandatory) [block: 2byte, sector: 2byte, generation: 4 byte]							
0x0040	Last Consistent time cont sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Last Attach time sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x0050	log position that was used the last time the database was attached If 0 dirty (NOT mandatory)								Last Detach time sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x0060	log position that was used the last time the database was detached If 0 dirty (NOT mandatory)								DBID ? (NOT mandatory)				RAND ? (NOT mandatory)			
0x0070	Log signature time: sec, min, hour, day, month, year (0=1900) (NOT mandatory)															
0x0080									Consists of a log position ?at full backup? (NOT mandatory)							
0x0090	Previous full backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Generation lower number The lower log generation number associated with the backup (NOT mandatory)							
0x00A0	Consists of a log position ?at incremental backup? (NOT mandatory)								Previous incremental backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)							
0x00B0	Generation lower number The lower log generation number associated with the backup (NOT mandatory)								Consists of a log position ?at current full backup? (NOT mandatory)							
0x00C0	Current full backup sec, min, hour, day, month, year (1900=0) (NOT mandatory)								Generation lower number The lower log generation number associated with the backup (NOT mandatory)							
0x00D0	Shadow disabled				Last object ID (NOT mandatory)				Major version (NOT mandatory)				Minor version (NOT mandatory)			
0x00E0	Build number (NOT mandatory)				Servicepack (NOT mandatory)				file format version				pagesize 0x00002000 (MANDATORY)			
0x00F0	repair count				Recovery Time (NOT mandatory)											
0x0100																
0x0110																
0x0120																
0x0130																
0x0140																
0x0150					File format verison ? (NOT mandatory)				File format verison minor? (NOT mandatory)							

So we fix the five constant parameters, and now we leave the two varing ones now, then select the **File / Save** command:



Find the Highest DB time

From the two varying parameters the Highest DB time is the simpler one so start with this. The bytes 0x08..0x0F at the beginning of every page contains the DB Time of that page. So we must go through the ntds.dit file page by page (the page size is still 8192 bytes), and find the highest value at the position 0x08..0x0F. It must be written there.

Of course it is a bit difficult to do manually, so I create a simple python script, to do it:

```
import struct
f = open("ntds.dit", "rb")
f.read(8192)
f.read(8192)
f.read(8)
n2 = struct.unpack("i", f.read(4)) [0]
max = n2
while n2 != '':
    f.read(8188)
    n2 = f.read(4)
    if n2 != '':
        n2 = struct.unpack("i", n2) [0]
        if max < n2:
            max = n2
f.close()
print "MAX : " + format(max & 0xffffffff, "08x")
```

If you run this script it will give you the required Highest DB time value:

```

Administrator: Command Prompt

C:\Windows\NTDS>dir
Volume in drive C has no label.
Volume Serial Number is 188C-1DCF

Directory of C:\Windows\NTDS

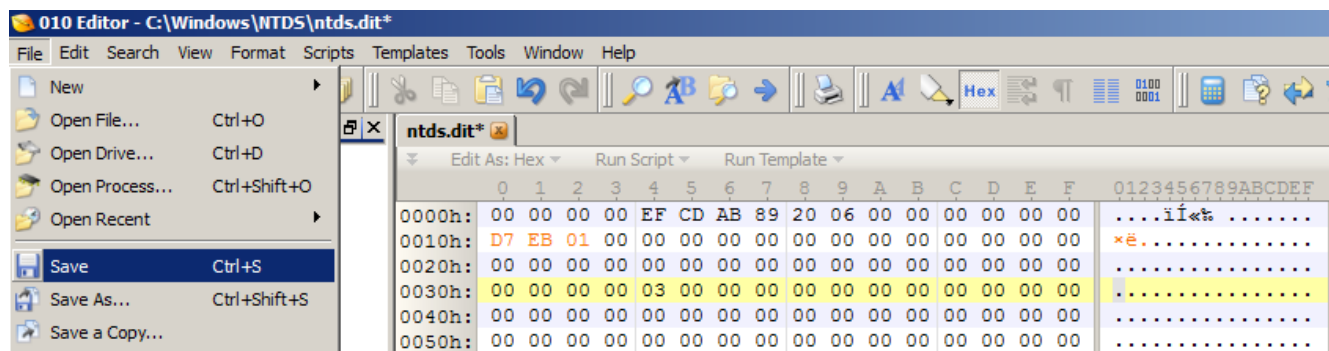
05/17/2011    08:20 AM         <DIR>          .
05/17/2011    08:20 AM         <DIR>          ..
05/17/2011    05:52 AM             325      chk2.py
05/17/2011    08:19 AM             8,192      edb.chk
05/17/2011    08:19 AM      10,485,760      edb.log
05/11/2011    03:30 PM      10,485,760      edb00001.log
05/11/2011    03:23 PM      10,485,760      edbres00001.jrs
05/11/2011    03:23 PM      10,485,760      edbres00002.jrs
05/17/2011    08:19 AM      12,599,296      ntds - Copy.dit
05/17/2011    08:24 AM      12,599,296      ntds.dit
               8 File(s)              67,150,149 bytes
               2 Dir(s)      21,548,580,864 bytes free

C:\Windows\NTDS>chk2.py
MAX : 0001ebd7

C:\Windows\NTDS>_

```

And we can write it to the position 0x0010..0x0017, then save itm select the **File / Save** command:



Find the Checksum

Now comes the more difficult part, the calculation of the checksum. First we should figure out, how dows the checksum calculated, then, we can write a program to calculate it.

One can try to start the service, and debug it, how does it test the checksum, but better to find a simpler application. For example the ntdsutil is such an application, it opens the ntds.dit, and test it, and a simpler application. First test, how dows it behaves. **Start the ntdsutil** with the command `ntdsutil`

from a command prompt. Then connect to the active directory with the

```
activate instance ntds
```

command. Then use the

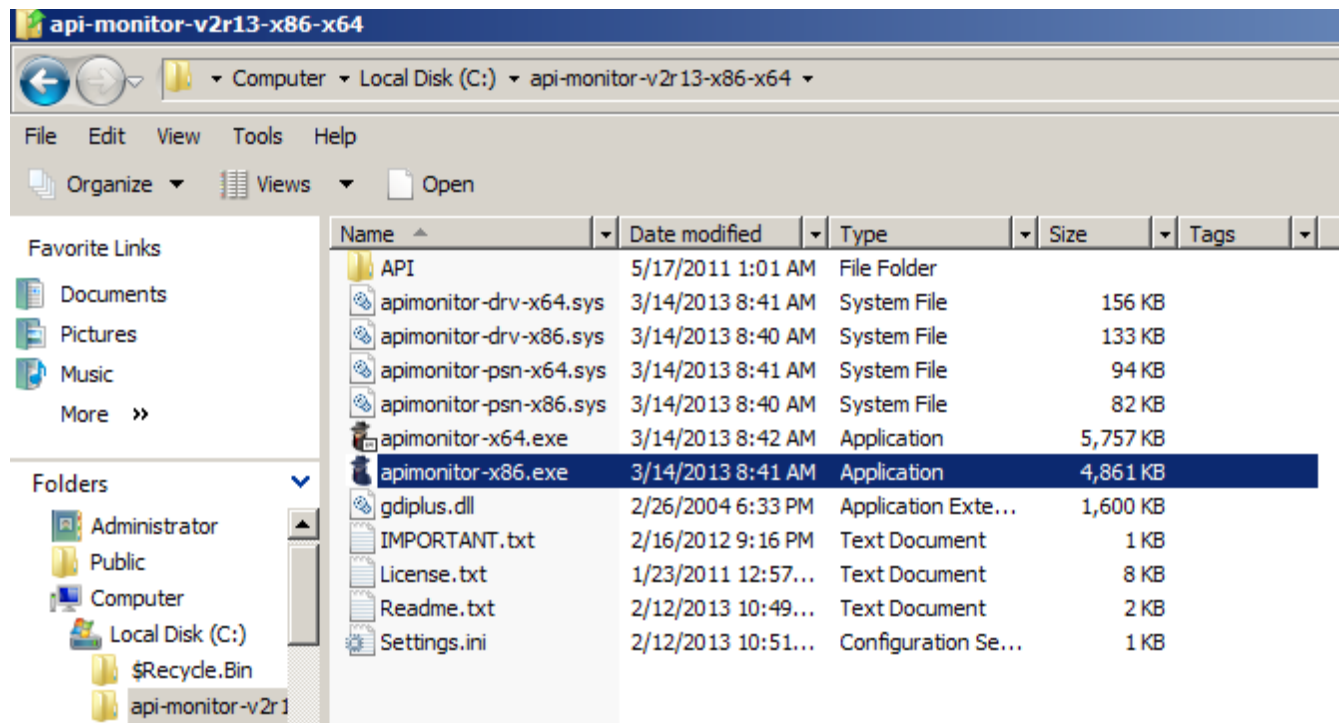
```
files
```

command, to start to examine the files, you will get some error message:

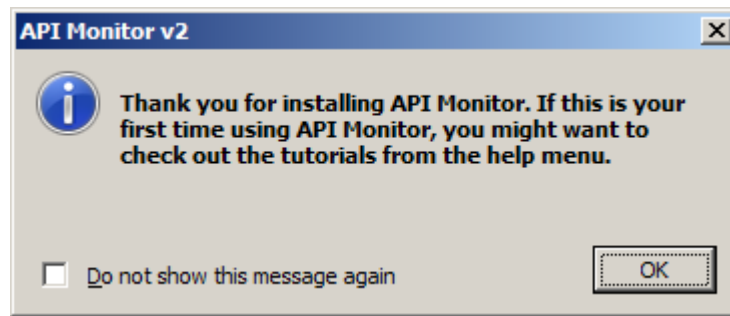
```
Administrator: Command Prompt - ntdsutil
C:\Windows\NTDS>ntdsutil
ntdsutil: Activate Instance ntds
Active instance set to "ntds".
ntdsutil: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
ntdsutil: _
```

Exit from the ntdsutil with the **quit** command.

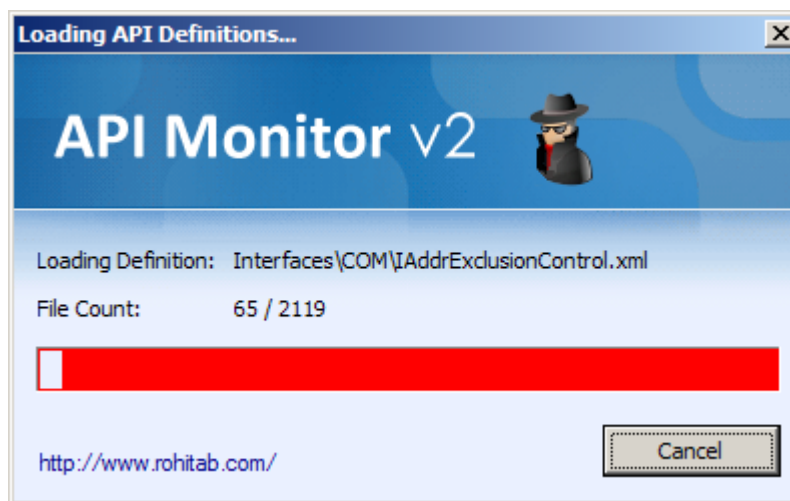
Now **start** the **api monitor**.



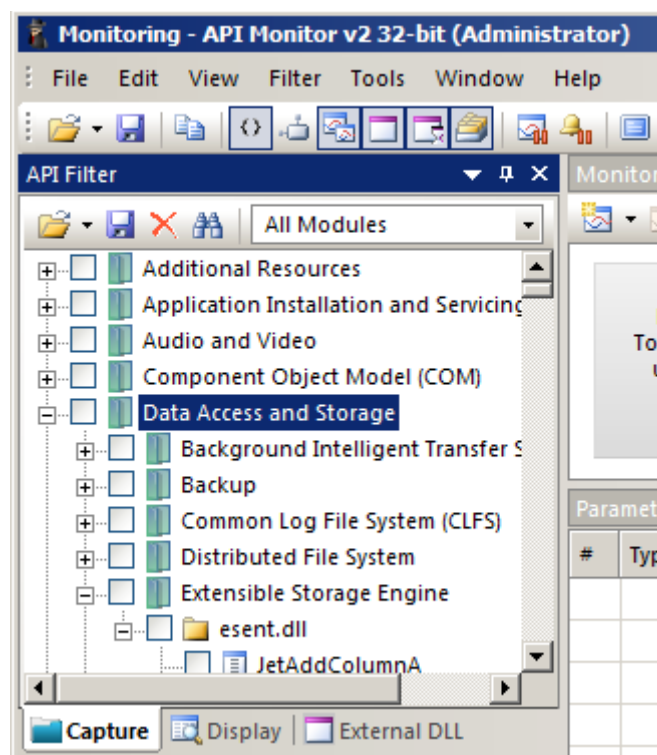
On the popup welcome message click to the OK button.



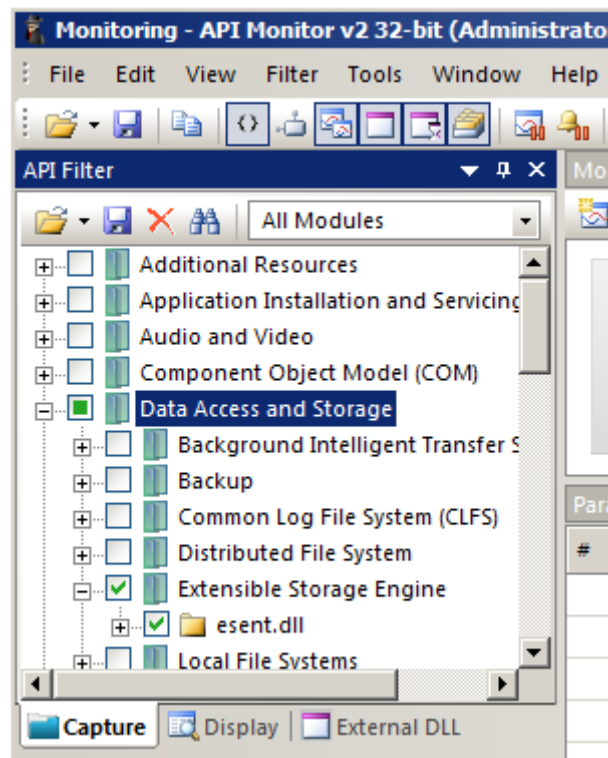
It starts to load the Definition files:



Find in the API filter the **Data Access and Storage \ Extensible Storage Engine \ esent.dll**



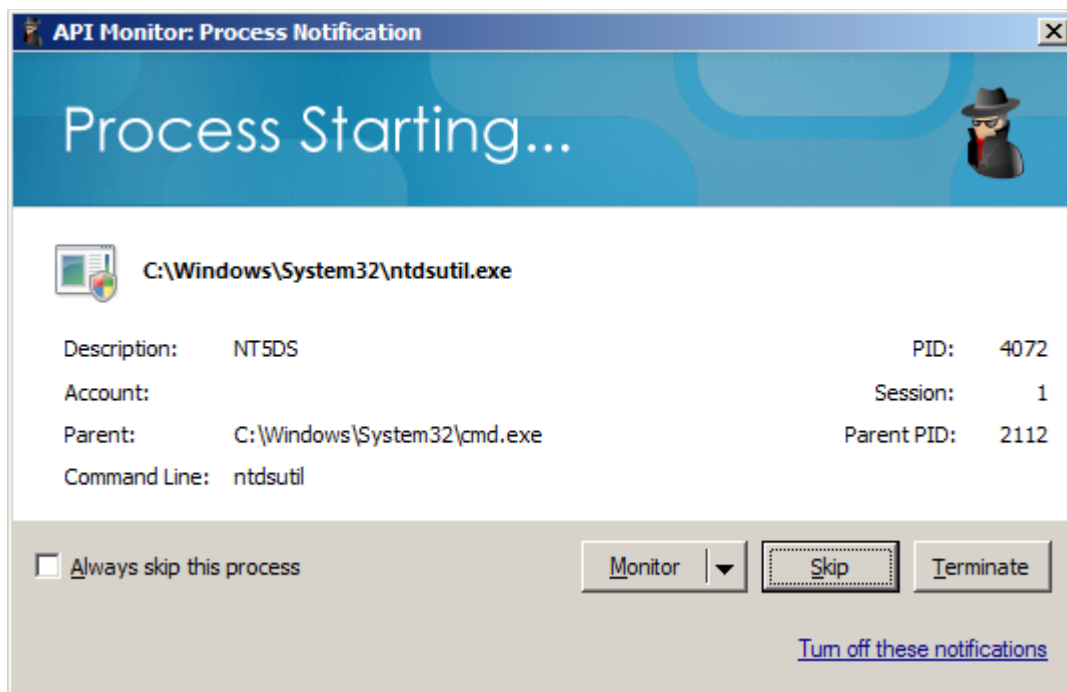
And check it as it can be seen in the picture:



Then go to the **command prompt**, and **start** again the **ntdsutil**

You will get a popup window, where the API monitor asks if you want to monitor the newly created process. Take care to check if it really the ntdsutil, because the windows often used to start various processes.

Click to the monitor button



Start the ntdsutil with the command `ntdsutil` **from a command prompt**. Then connect to the active directory with the

`activate instance ntds`

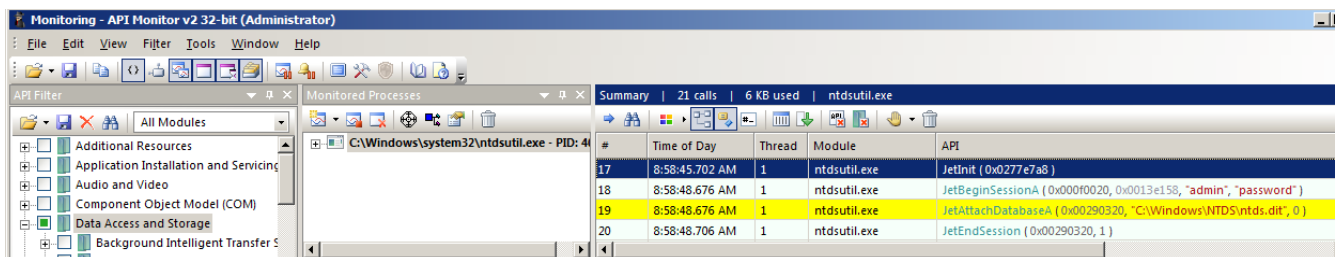
command. Then use the

`files`

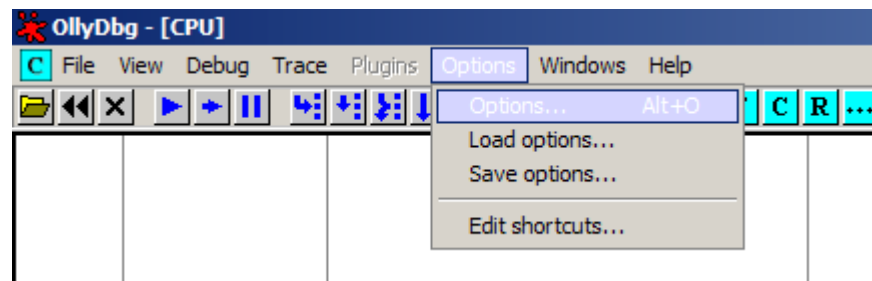
command, to start to examine the files, you will get some error message:

```
Administrator: Command Prompt - ntdsutil
C:\Windows\NTDS>ntdsutil
ntdsutil: activate instance ntds
Active instance set to "ntds".
ntdsutil: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
ntdsutil: _
```

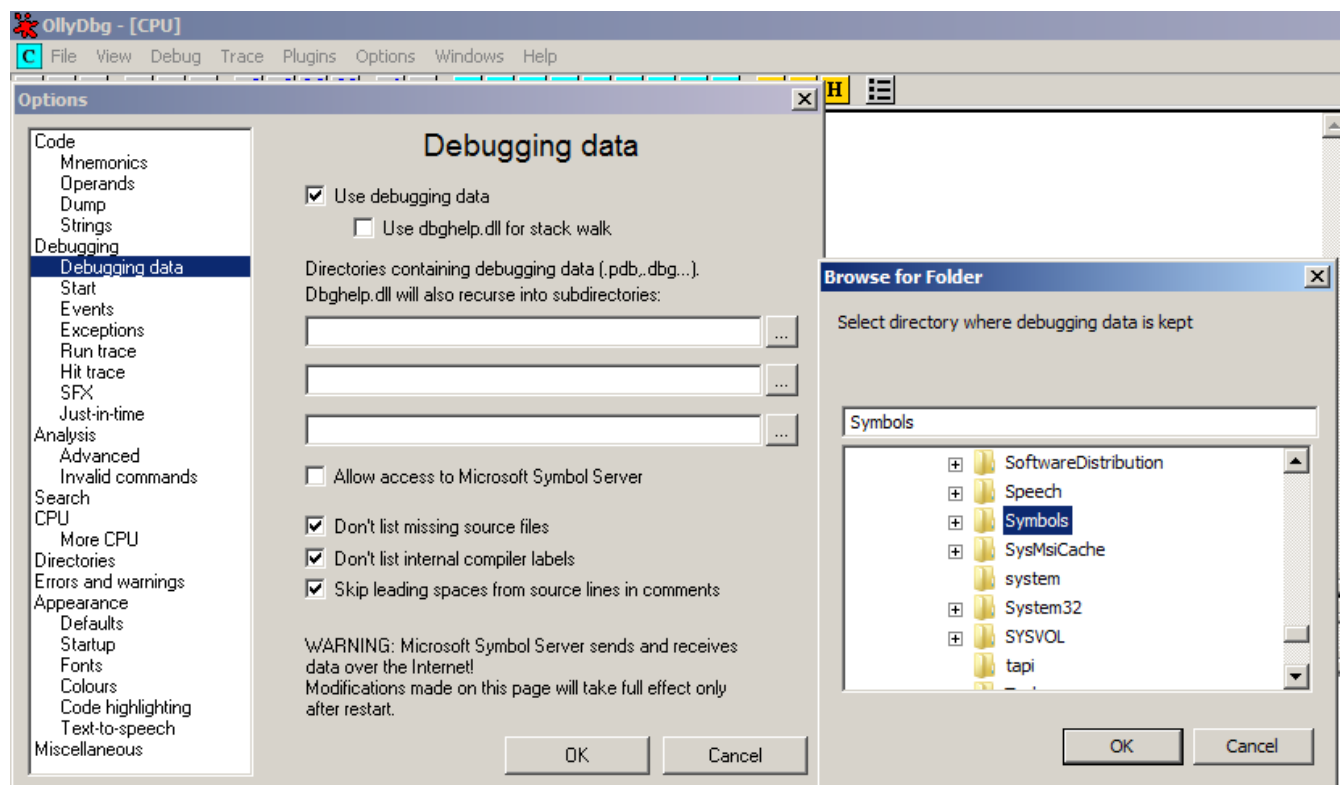
But the error message is unimportant. Go back to the api monitor, and scroll over the captured events. You will find some with yellow background. The yellow background means the function returned with some error. It is just fine for us, because we got some error. We can read the function name, `JetAttachDatabaseA`:



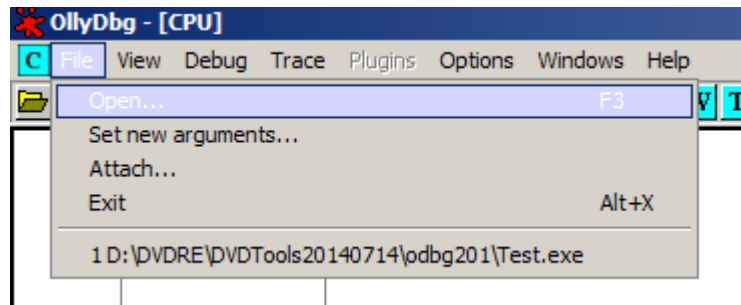
Now we can analyze this function, to find the cause of the error (most probably the wrong checksum). To do it **start the Olly Debugger**, and set up it to use the symbol files by clicking to the **Options \ Options...**



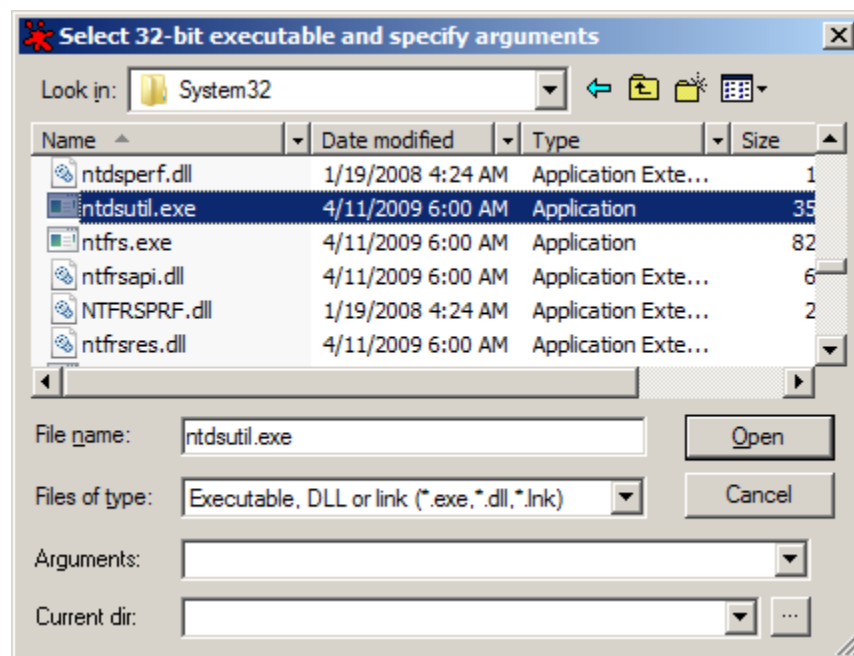
In the Opening window select **Debugging \ Debugging data**, and define the symbol path by clicking to one of the ... icons.



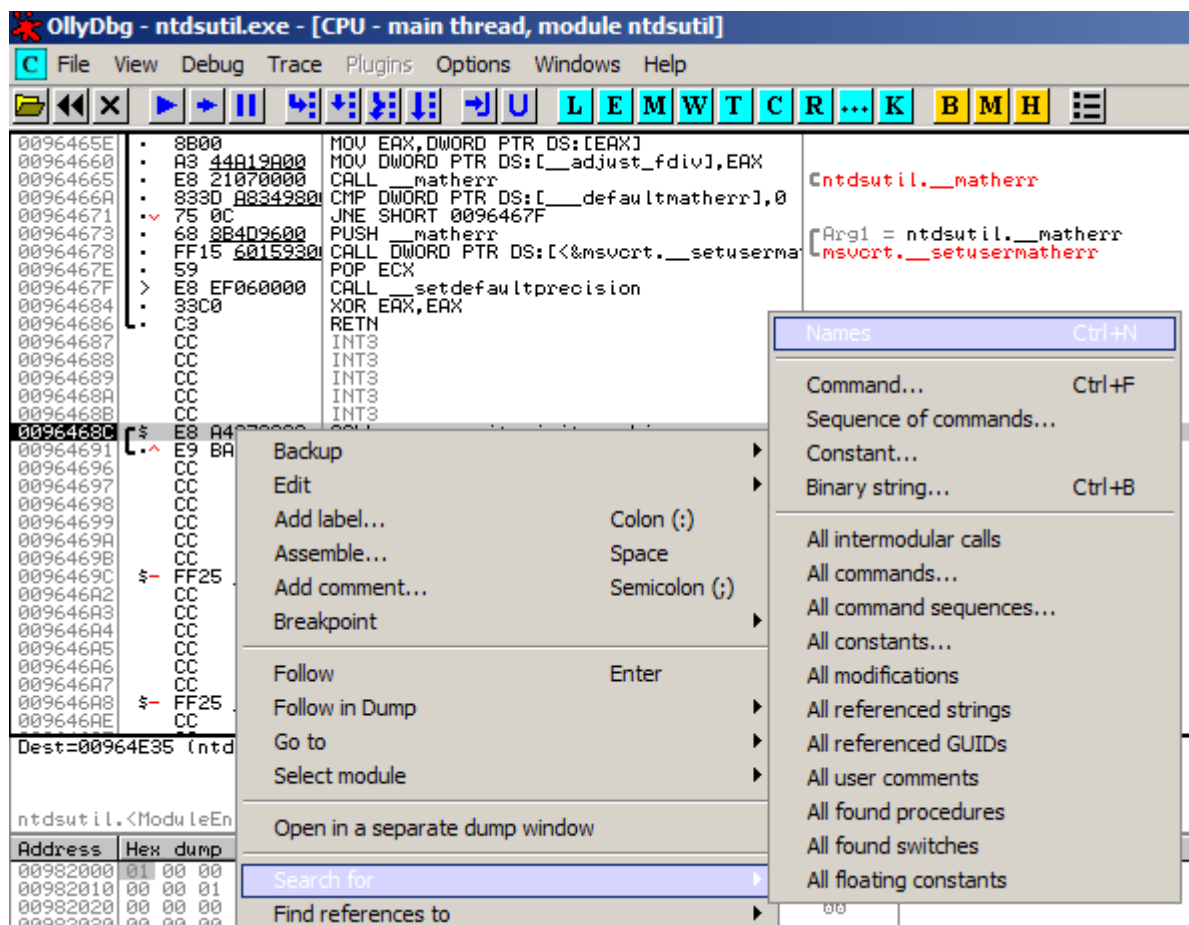
The open the ntdsutil by clicking the **Files \ Open...**



Then **find** the **windows\System32\ntdsutil.exe**, and click to the **open** button



When the application opens **right click to the disassembly window**, and from the popup menu select **search for \ Names**.

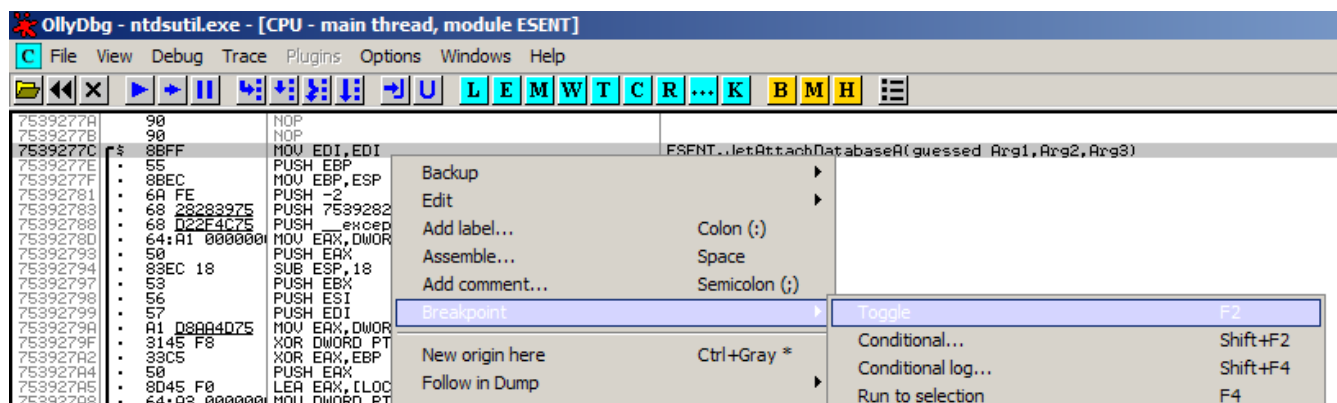


In the appearing window start to **type JeatAttachDatabaseA**, the function we want to examine. **Double click to it**, then we jump to the start of the function.

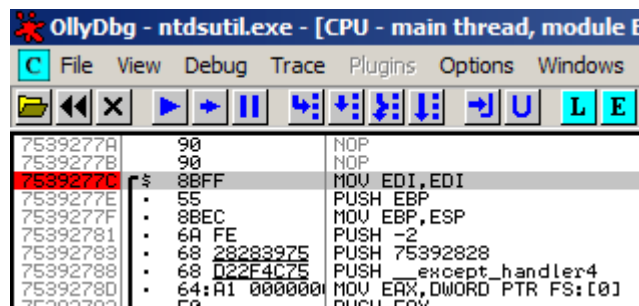
Names in ntdsutil					
Address	Section	Type	Ordinal	Name	Comments
00931104	.text	Import		&ESSENT.JetTerm2	
00931108	.text	Import		&ESSENT.JetOpenTableA	
0093110C	.text	Import		&ESSENT.JetGetColumnInfoA	
00931110	.text	Import		&ESSENT.JetSetCurrentIndex2A	
00931114	.text	Import		&ESSENT.JetGetTableColumnInfoA	
00931118	.text	Import		&ESSENT.JetOpenTempTable	
0093111C	.text	Import		&ESSENT.JetSetTableSequential	
00931120	.text	Import		&ESSENT.JetGetLock	
00931124	.text	Import		&ESSENT.JetSetColumns	
00931128	.text	Import		&ESSENT.JetAttachDatabaseA	
0093112C	.text	Import		&ESSENT.JetOpenDatabaseA	
00931130	.text	Import		&ESSENT.JetGetTableIndexInfoA	
00931134	.text	Import		&ESSENT.JetCreateTableColumnIndexA	
00931138	.text	Import		&ESSENT.JetDetachDatabaseA	
0093113C	.text	Import		&ESSENT.JetGetSystemParameterA	

Search - JETATTACHDATABASEA

Put there a breakpoint by the **F2 button**, or by right clicking to the disassembly window and select the Breakpoint \ Toggle command.



Then run the application by the **F9** button.



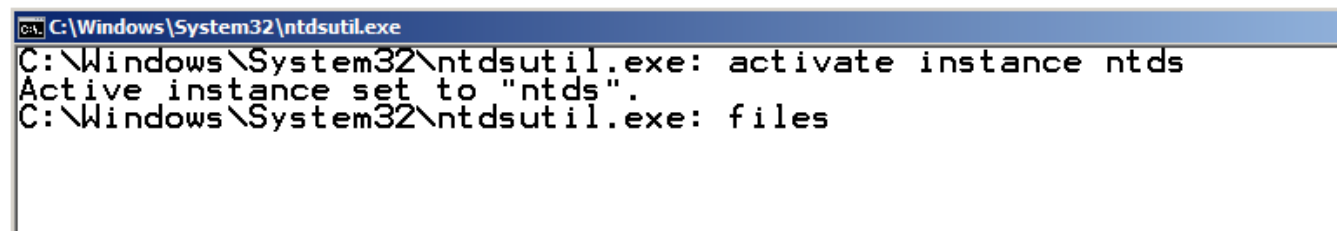
connect to the active directory with the

activate instance ntds

command. Then use the

files

command, to start to examine the files



The debugger stops at the break point:

Address	Disassembly	Comment
7539277A	90	NOP
7539277B	90	NOP
7539277C	8BFF	MOV EDI,EDI
7539277E	55	PUSH EBP
7539277F	8BEC	MOV EBP,ESP
75392781	6A FE	PUSH -2
75392783	68 28283975	PUSH 75392828
75392788	68 022F4C75	PUSH _except_handler4
7539278D	64:A1 000000	MOV EAX,DWORD PTR FS:[0]

ASCII "C:\Windows\NTDS\ntds.dit"

Start to scroll down, and find the CALL instructions, because most probably one of the internal functions will return with error. Put a break point to every call instruction (there are only two of them), and let the application run until that. Then press the F8 button, to step over the call function, and examine the return value of it (the return value is most of the time in the EAX register). If you have two test machines you can compare the function results of a machine with good ntds.dit and one with the modified ntds.dit to see when we get some difference.

Address	Disassembly	Comment
7539279A	A1 08A04D75	MOV EAX,DWORD PTR DS:[_security_cookie]
7539279F	3145 F8	XOR DWORD PTR SS:[LOCAL.2],EAX
753927A2	33C5	XOR EAX,EBP
753927A4	50	PUSH EAX
753927A5	8D45 F0	LEA EAX,[LOCAL.4]
753927A8	64:A3 000000	MOV DWORD PTR FS:[0],EAX
753927AE	8965 E8	MOV DWORD PTR SS:[LOCAL.6],ESP
753927B1	33C0	XOR EAX,EAX
753927B3	8945 E4	MOV DWORD PTR SS:[LOCAL.7],EAX
753927B6	8945 FC	MOV DWORD PTR SS:[LOCAL.11],EAX
753927B9	8B45 08	MOV EAX,DWORD PTR SS:[ARG.1]
753927BC	50	PUSH EAX
753927BD	6A 05	PUSH 5
753927BE	E8 AB00100	CALL ?FValid@CResource@@SGH4JET_RESID@@@
753927C4	8945 E4	MOV DWORD PTR SS:[LOCAL.7],EAX
753927C7	BF FEFFFFFF	MOV EDI,-2
753927CC	897D FC	MOV DWORD PTR SS:[EBP-4],EDI
753927CF	837D E4 00	CMPL DWORD PTR SS:[EBP-1C],0
753927D3	0F84 DC07080	JE 75412FB5
753927D9	C745 FC 0100	MOV DWORD PTR SS:[EBP-4],1
753927E0	8B4D 10	MOV ECX,DWORD PTR SS:[EBP+10]
753927E3	51	PUSH ECX
753927E4	6A 00	PUSH 0
753927E6	6A 00	PUSH 0
753927E8	6A 00	PUSH 0
753927EA	8B55 0C	MOV EDX,DWORD PTR SS:[EBP+0C]
753927ED	52	PUSH EDX
753927EE	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
753927F1	50	PUSH EAX
753927F2	E8 5E000000	CALL _JetAttachDatabaseExA@24
753927F7	8BF0	MOV ESI,EAX
753927F9	8975 D8	MOV DWORD PTR SS:[EBP-28],ESI
753927FC	833D 6C024075	CMPL DWORD PTR DS:[?g_fTracing@@3HA],0

Dest=75392855 (ESENT._JetAttachDatabaseExA@24)

Arg2 => [ARG.1]
Arg1 = 5
ESENT._FValid@CResource@@SGH4JET_RESID@@@QBx@2

Arg6 = 0
Arg5 = 0
Arg4 = 0
Arg3 = 0
Arg2 = 0
Arg1 = 0
ESENT._JetAttachDatabaseExA@24

The first CALL returns with 1 in EAX register. It is normal, one get the same return value, if we run the ntdsutil for a good file.

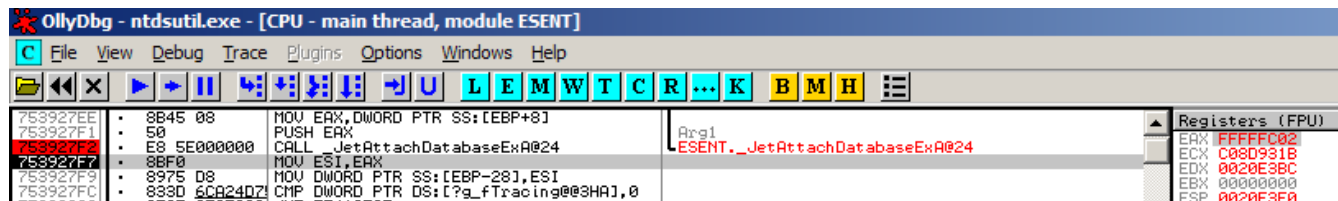
Address	Disassembly	Comment
753927BC	50	PUSH EAX
753927BD	6A 05	PUSH 5
753927BE	E8 AB00100	CALL ?FValid@CResource@@SGH4JET_RESID@@@
753927C4	8945 E4	MOV DWORD PTR SS:[LOCAL.7],EAX
753927C7	BF FEFFFFFF	MOV EDI,-2
753927CC	897D FC	MOV DWORD PTR SS:[EBP-4],EDI
753927CF	837D E4 00	CMPL DWORD PTR SS:[EBP-1C],0
753927D3	0F84 DC07080	JE 75412FB5
753927D9	C745 FC 0100	MOV DWORD PTR SS:[EBP-4],1

Arg2 => [ARG.1]
Arg1 = 5
ESENT._FValid@CResource@@SGH4JET_RESID@@@QBx@2

Registers (FPU)
EAX 00000001
ECX 00920320
EDX 00000000
EBX 00000000
ESP 0020E3F0
EBP 0020E428
ESI 00000001

But the second call returns with an interesting result 0xFFFFC02. If you change this DWORD to decimal you get -1022, what is the error message we saw in the ntdsutil. So most probably this is what

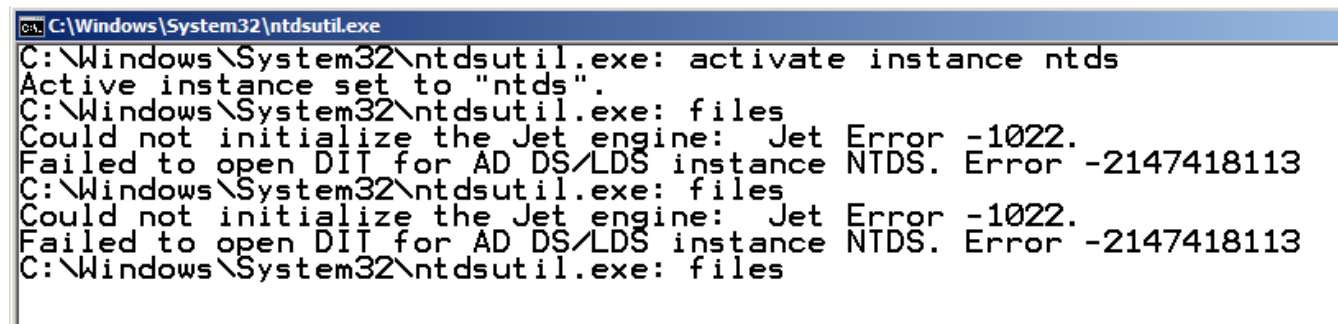
we are looking for:



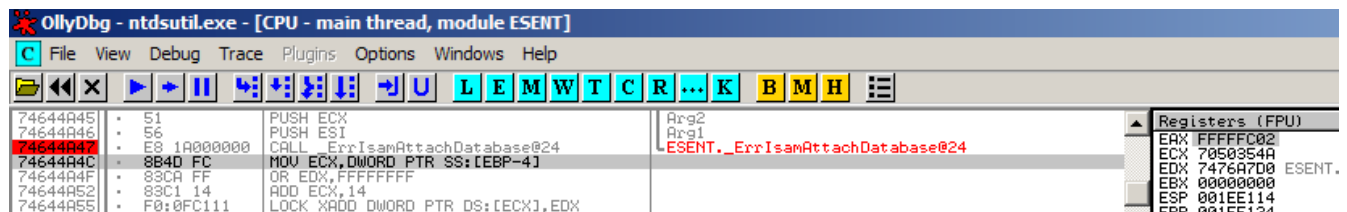
Now we can remove all the previous breakpoints, and follow the suspicious CALL by clicking it once, and pressing the ENTER button.

Then we put a break point by the F2 button to the beginning of this function.

Now we type again the files command in the ntdsutil, and it stops on our new breakpoint.



Again as we did earlier we **put a breakpoint to every call**, and let the application to **run until the CALL**, Then press the **F8** button on the CALL, to step over it. Then **examine the return value** of the CALLs (EAX), **if** the result is the **0xFFFFFC02**. We will find the call returns this value:



Then we repeat the previous step. Again we remove all the previous breakpoints, and follow the suspicious CALL by clicking it once, and pressing the ENTER button.

Then we put a break point by the F2 button to the beginning of this function.

Now we type again the

files

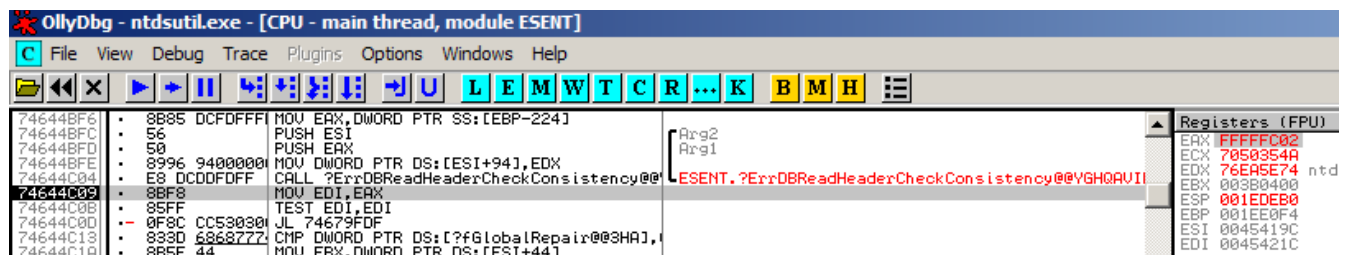
command in the ntdsutil, and it stops on our new breakpoint.

```

C:\Windows\System32\ntdsutil.exe
C:\Windows\System32\ntdsutil.exe: activate instance ntds
Active instance set to "ntds".
C:\Windows\System32\ntdsutil.exe: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
C:\Windows\System32\ntdsutil.exe: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
C:\Windows\System32\ntdsutil.exe: files

```

Again as we did earlier we **put a breakpoint to every call**, and let the application to **run until the CALL**, Then press the **F8** button on the CALL, to step over it. Then **examine the return value** of the CALLs (EAX), **if the result is the 0xFFFFFC02**. We will find the call returns this value:



Now repeat the previous step. **Remove all the previous breakpoints**, and follow the **suspicious CALL** by **clicking it once**, and pressing the **ENTER** button.

Then we put a break point by the **F2** button to the beginning of this function.

Now go back to the ntdsutil in the command prompt, and type again the

files

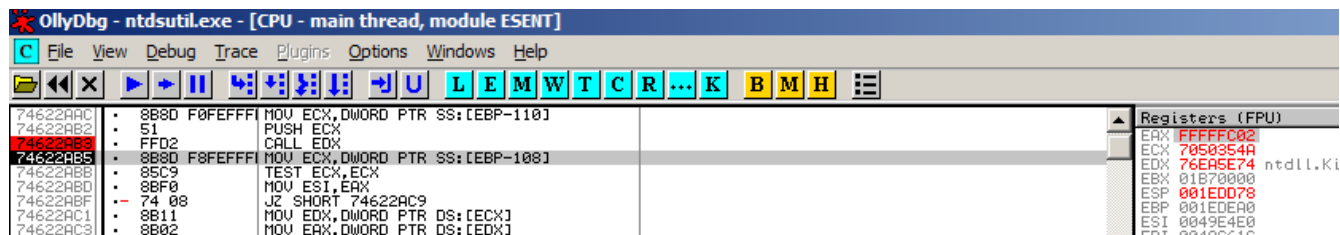
command, and it stops on our new breakpoint.

```

C:\Windows\System32\ntdsutil.exe
C:\Windows\System32\ntdsutil.exe: activate instance ntds
Active instance set to "ntds".
C:\Windows\System32\ntdsutil.exe: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
C:\Windows\System32\ntdsutil.exe: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
C:\Windows\System32\ntdsutil.exe: files
Could not initialize the Jet engine: Jet Error -1022.
Failed to open DIT for AD DS/LDS instance NTDS. Error -2147418113
C:\Windows\System32\ntdsutil.exe:

```

Again as we did earlier we **put a breakpoint to every call**, and let the application to **run until the CALL**, Then press the **F8** button on the CALL, to step over it. Then **examine the return value** of the CALLs (EAX), **if the result is the 0xFFFFFC02**. We will find the call returns this value:



As one can see now a CALL EDX is the suspicious function. It is a bit different like earlier.

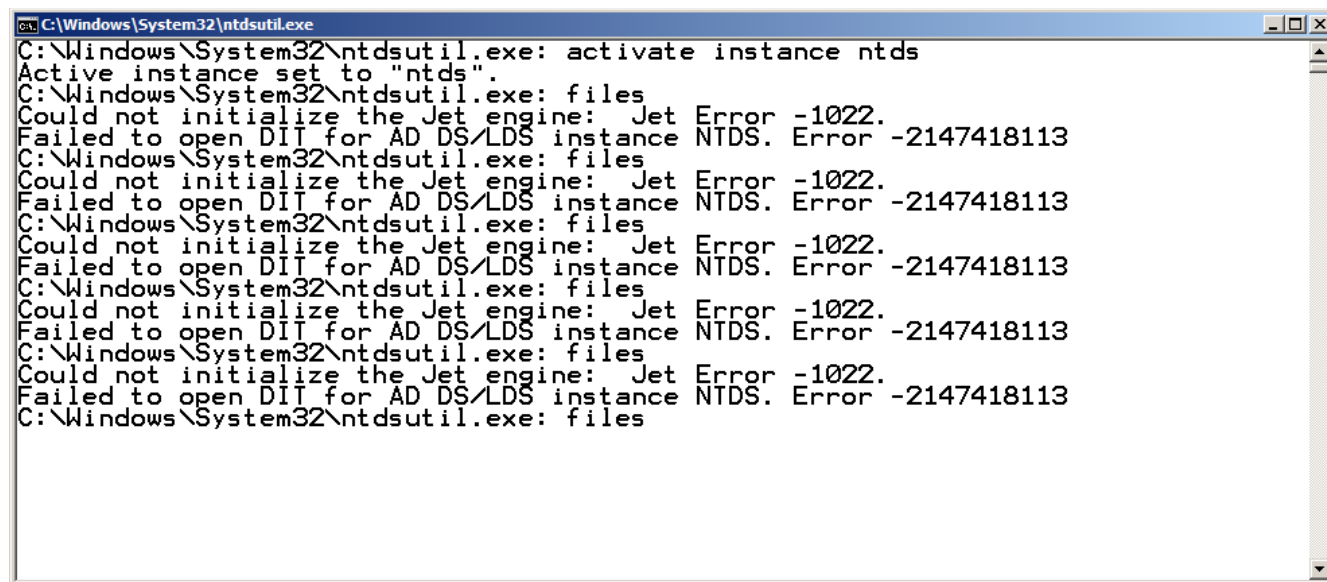
Now repeat the previous step. **Remove all the previous breakpoints**, and follow the **suspicious CALL** by **clicking it once**, and pressing the **ENTER** button.

Then we put a break point by the **F2** button to the beginning of this function.

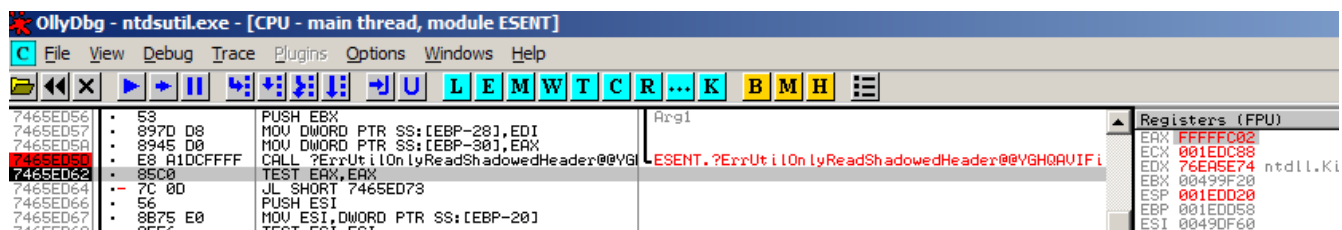
Now go back to the ntdsutil in the command prompt, and type again the

files

command, and it stops on our new breakpoint.



Again as we did earlier we **put a breakpoint to every call**, and let the application to **run until the CALL**, Then press the **F8** button on the CALL, to step over it. Then **examine the return value** of the CALLs (EAX), **if** the result is the **0xFFFFFC02**. We will find the call returns this value:



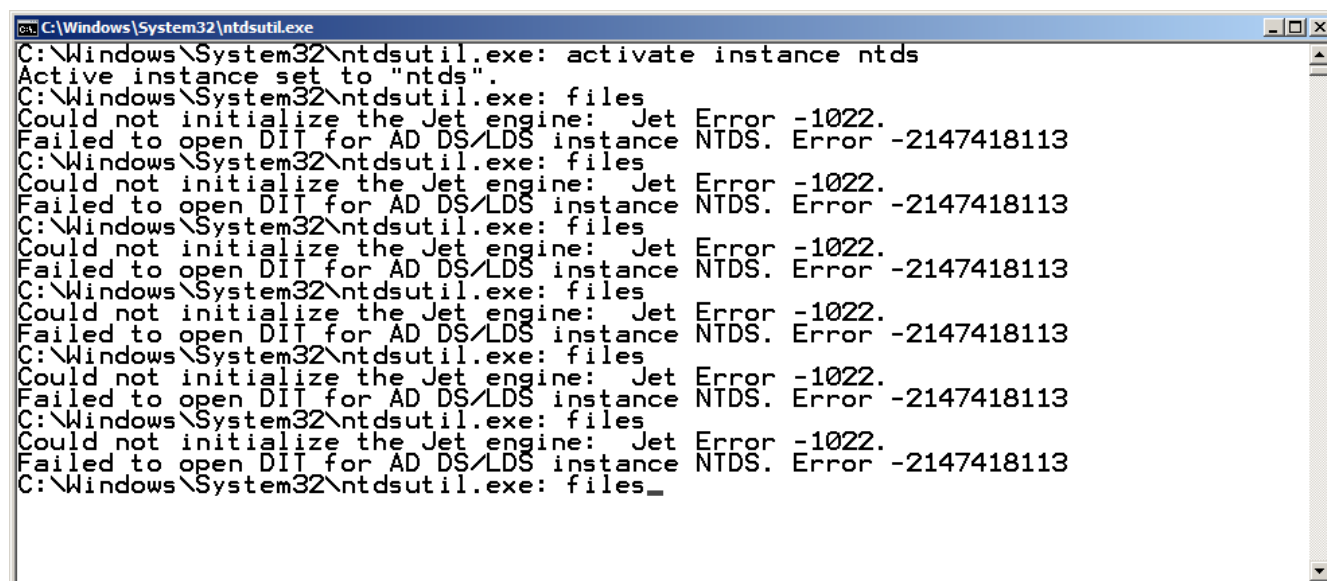
Now repeat the previous step. **Remove all the previous breakpoints**, and follow the **suspicious CALL** by **clicking it once**, and pressing the **ENTER** button.

Then we put a break point by the **F2** button to the beginning of this function.

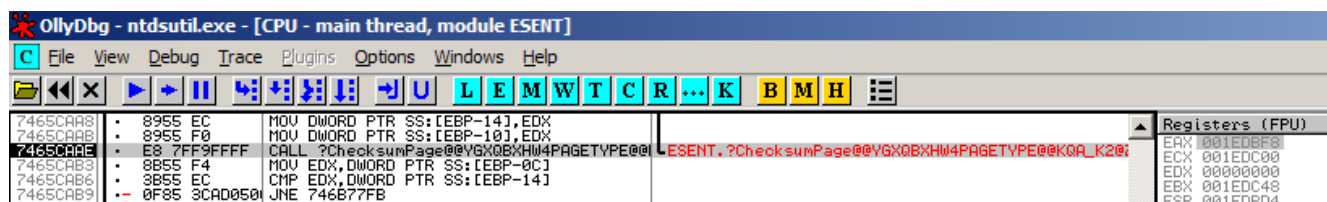
Now go back to the ntdsutil in the command prompt, and type again the

files

command, and it stops on our new breakpoint.

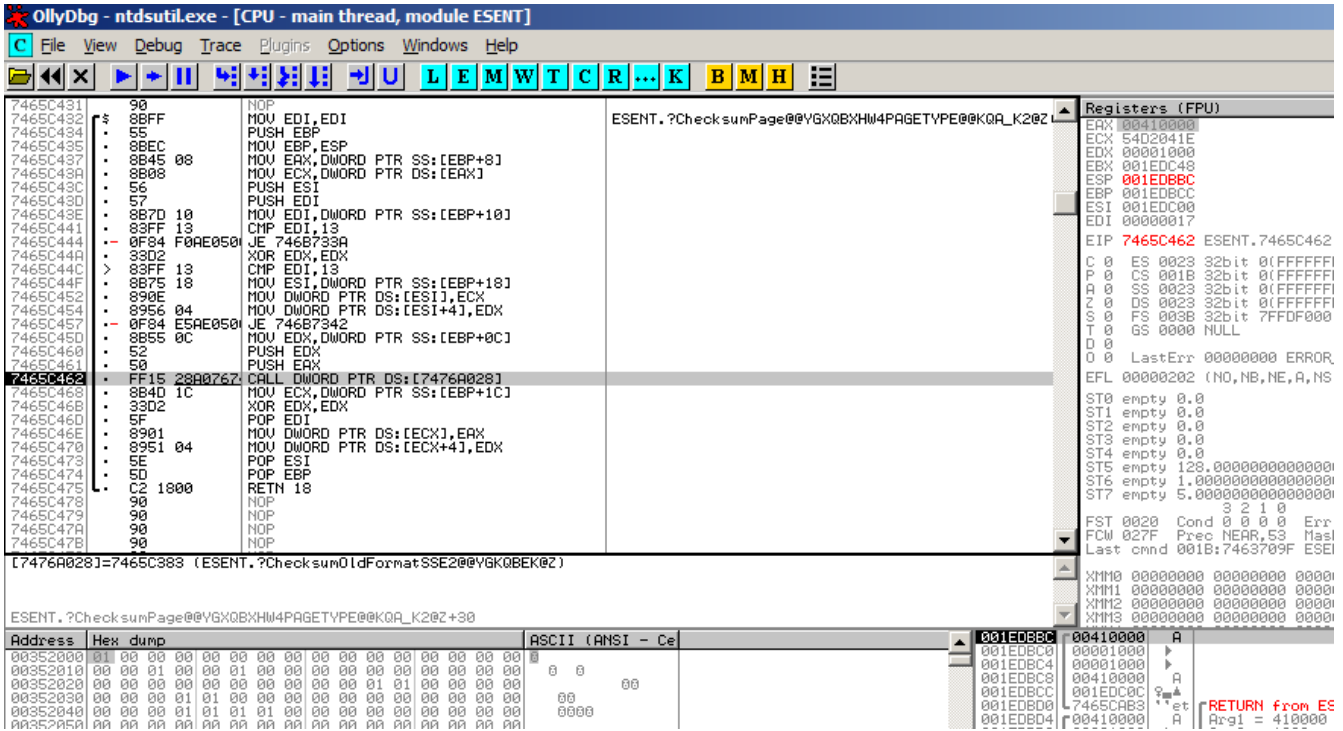


Again as we did earlier we **put a breakpoint to every call**, and let the application to **run until the CALL**, Then press the **F8** button on the CALL, to step over it. Then **examine the return value** of the CALLs (EAX), **if the result is the 0xFFFFFFFFC02**. Now we do not find any call returns this value, but there is a suspicious one. It returns some interesting value, and the name of it ChecksumPage... make it interesting :



Enter to this function with the F7 button:

In this function there is an interesting CALL, we do not see the name of it, or any information about it. Let us enter to this CALL with the F7 button again:



We will find there the following code:

OllyDbg - ntdsutil.exe - [CPU - main thread, module ESENT]

File View Debug Trace Plugins Options Windows Help

File View Debug Trace Plugins Options Windows Help

7465C382 • 90 NOP

7465C383 • 8BFF MOV EDI,EDI

7465C385 • 53 PUSH EBX

7465C386 • 8BDC MOV EBX,ESP

7465C388 • 83EC 08 SUB ESP,8

7465C38B • 83E4 F0 AND ESP,FFFFFFF0

7465C38E • 83C4 04 ADD ESP,4

7465C391 • 55 PUSH EBP

7465C392 • 8B6B 04 MOV EBP,DWORD PTR DS:[EBX+4]

7465C395 • 896C24 04 MOV DWORD PTR SS:[ESP+4],EBP

7465C399 • 8BEC MOV EBP,ESP

7465C39B • 8B43 08 MOV EAX,DWORD PTR DS:[EBX+8]

7465C39E • 8B10 MOV EDX,DWORD PTR DS:[EAX]

7465C3A0 • 8B4B 0C MOV ECX,DWORD PTR DS:[EBX+0C]

7465C3A3 • 83EC 1C SUB ESP,1C

7465C3A6 • 03C8 ADD ECX,EAX

7465C3A8 • 81F2 EFCDA8 XOR EDX,89ABCDEF

7465C3AE • 56 PUSH ESI

7465C3AF • 660FEFC0 PXOR XMM0,XMM0

7465C3B3 • 83C0 10 ADD EAX,10

7465C3B6 > 660F6F10 MOVDQA XMM2,DWORD PTR DS:[EAX]

7465C3BA • 660F6F48 F0 MOVDQA XMM1,DWORD PTR DS:[EAX+10]

7465C3BF • 660F6F58 20 MOVDQA XMM3,DWORD PTR DS:[EAX+20]

7465C3C4 • 0F1800 F00000 PREFETCHNTA DS:[EAX+0F0]

7465C3C8 • 660FEFCA PXOR XMM1,XMM2

7465C3CF • 660F6F50 10 MOVDQA XMM2,DWORD PTR DS:[EAX+10]

7465C3D4 • 660FEFD3 PXOR XMM2,XMM3

7465C3D8 • 660F6F58 60 MOVDQA XMM3,DWORD PTR DS:[EAX+60]

7465C3DD • 660FEFCA PXOR XMM1,XMM2

7465C3E1 • 660F6F50 40 MOVDQA XMM2,DWORD PTR DS:[EAX+40]

7465C3E6 • 660FEFC1 PXOR XMM0,XMM1

7465C3EA • 660F6F48 30 MOVDQA XMM1,DWORD PTR DS:[EAX+30]

7465C3EF • 660FEFCA PXOR XMM1,XMM2

7465C3F3 • 660F6F50 50 MOVDQA XMM2,DWORD PTR DS:[EAX+50]

7465C3F8 • 05 80000000 ADD EAX,80

7465C3FD • 8D70 F0 LEA ESI,[EAX+10]

7465C400 • 3BF1 CMP ESI,ECX

7465C402 • 660FEFD3 PXOR XMM2,XMM3

7465C406 • 660FEFCA PXOR XMM1,XMM2

7465C40A • 660FEFC1 PXOR XMM0,XMM1

7465C40E • 72 A6 JB SHORT 7465C3B6

7465C410 • 660F7F45 F0 MOVDQA DWORD PTR SS:[EBP+10],XMM0

7465C415 • 8B45 FC MOV EAX,DWORD PTR SS:[EBP+4]

7465C418 • 3345 F8 XOR EAX,DWORD PTR SS:[EBP+8]

7465C41B • 5E POP ESI

7465C41C • 3345 F4 XOR EAX,DWORD PTR SS:[EBP+0C]

7465C41F • 3345 F0 XOR EAX,DWORD PTR SS:[EBP+10]

7465C422 • 33C2 XOR EAX,EDX

7465C424 • 8BE5 MOV ESP,EBP

7465C426 • 5D POP EBP

7465C427 • 8BE3 MOV ESP,EBX

7465C429 • 5B POP EBX

7465C42A • C2 0800 RETN 8

7465C42D • 90 NOP

EDX=00000017 (decimal 23.)

DWORD (16.-byte) stack alignment

As one can see it uses a lot of XOR operation, and uses the 128 bit XMM registers. Both of these is often happens in case of HASHING or ENCRYPTION or DECRYPTION algorithms. Now we are searching for a HASH algorithm, so hopefully this is what we were looking for.

If you put a break point after the calculation in EAX we see the result, probably the checksum.

OllyDbg - ntdsutil.exe - [CPU - main thread, module ESENT]

File View Debug Trace Plugins Options Windows Help

File View Debug Trace Plugins Options Windows Help

7465C400 • 3BF1 CMP ESI,ECX

7465C402 • 660FEFD3 PXOR XMM2,XMM3

7465C406 • 660FEFCA PXOR XMM1,XMM2

7465C40A • 660FEFC1 PXOR XMM0,XMM1

7465C40E • 72 A6 JB SHORT 7465C3B6

7465C410 • 660F7F45 F0 MOVDQA DWORD PTR SS:[EBP+10],XMM0

7465C415 • 8B45 FC MOV EAX,DWORD PTR SS:[EBP+4]

7465C418 • 3345 F8 XOR EAX,DWORD PTR SS:[EBP+8]

7465C41B • 5E POP ESI

7465C41C • 3345 F4 XOR EAX,DWORD PTR SS:[EBP+0C]

7465C41F • 3345 F0 XOR EAX,DWORD PTR SS:[EBP+10]

7465C422 • 33C2 XOR EAX,EDX

7465C424 • 8BE5 MOV ESP,EBP

7465C426 • 5D POP EBP

7465C427 • 8BE3 MOV ESP,EBX

7465C429 • 5B POP EBX

7465C42A • C2 0800 RETN 8

Registers (FPU)

EAX 0001CDF4

ECX 01B32000

EDX 89ABCDEF

EBX 001EDCA4

ESP 001EDC74

EBP 001EDC90

ESI 001EDCF0

EDI 00000017

EIP 7465C424 ESE

C 0 ES 0023 32b

P 0 CS 001B 32b

A 0 SS 0023 32b

Z 0 DS 0023 32b

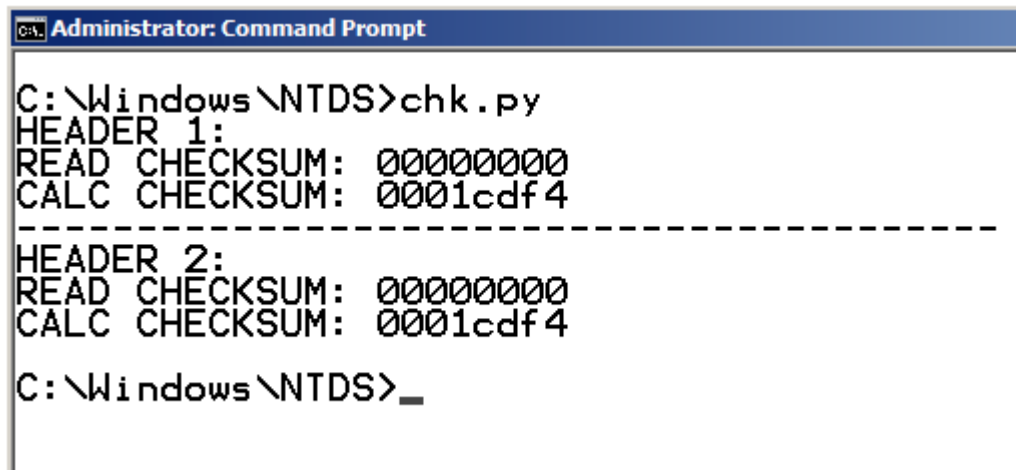
S 0 FS 003B 32b

T 0 CC 0000 Nil

If one analyze the checksum calculation code of the assembly, then after a time one can figure out, it just creates a simple XOR checksum. Based on this information we can create a python script, to calculate the checksum of both header, and also print the read checksum:

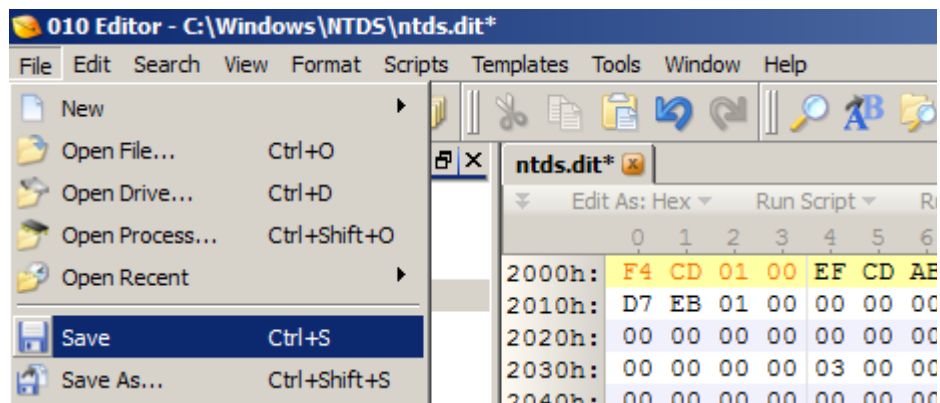
```
import struct
f = open("ntds.dit", "rb")
n1 = struct.unpack("i", f.read(4)) [0]
print "HEADER 1:"
print "READ CHECKSUM: " + format(n1 & 0xffffffff, "08x")
n1 = 0
for x in range(0, 2047):
    n2 = struct.unpack("i", f.read(4)) [0]
    n1 = n1 ^ n2
n1 = n1 ^ 0x89abcdef
print "CALC CHECKSUM: " + format(n1 & 0xffffffff, "08x")
print "-----"
print "HEADER 2:"
n1 = struct.unpack("i", f.read(4)) [0]
print "READ CHECKSUM: " + format(n1 & 0xffffffff, "08x")
n1 = 0
for x in range(0, 2047):
    n2 = struct.unpack("i", f.read(4)) [0]
    n1 = n1 ^ n2
f.close()
n1 = n1 ^ 0x89abcdef
print "CALC CHECKSUM: " + format(n1 & 0xffffffff, "08x")
f.close()
```

When we run this script it calculates the checksums:

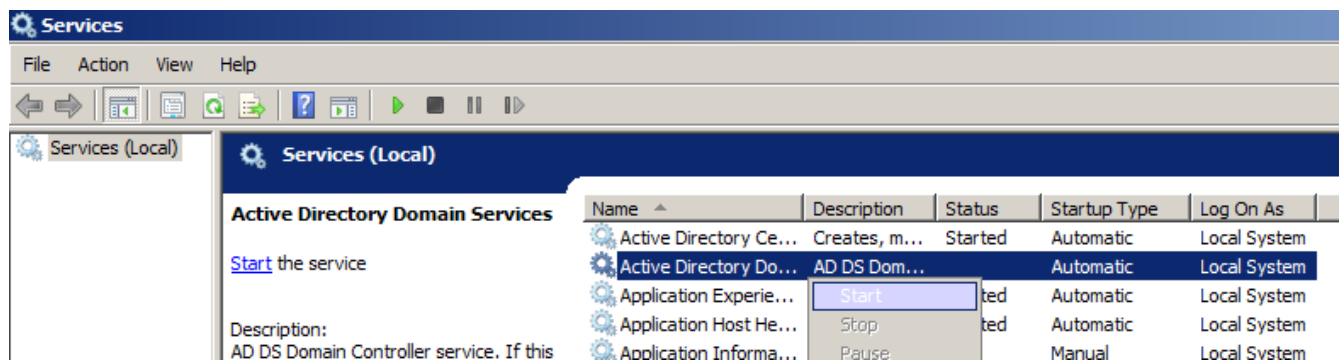


```
C:\Windows\NTDS>chk.py
HEADER 1:
READ CHECKSUM: 00000000
CALC CHECKSUM: 0001cdf4
-----
HEADER 2:
READ CHECKSUM: 00000000
CALC CHECKSUM: 0001cdf4
C:\Windows\NTDS>_
```

Now we can open the file again, and add the calculated checksum and save it **File \ Save**.



Now try to start the service again.



And it starts as expected:

