Hardcore debugging techniques for Web Application and WCF Services using NetExt

You will learn post-mortem details of Web Application and WCF Services from captured dumps

Rodney Viana (http://blogs.msdn.com/b/rodneyviana)

NetExt: http://netext.codeplex.com

Introduction

Estimated time to complete this lab

60 minutes

Objectives

After completing this lab, you will be able to:

- Analyze complex pattern from dump files or live applications in order to troubleshoot both simple and complex issues.
- Create scripts and sql-like commands to display sophisticated outputs based on:
  - Object type
  - Inheritance chain
  - Whether contain fields of a particular type
  - Where clauses filtering by object instance values.
- Use techniques to identify and troubleshoot common problems with Web Application (hosted or on premises)
- Apply data mining style debugging techniques using NetExt
- Identify the issuer of claims coming from Azure Control Service

Prerequisites

- You must have downloaded both **32 and 64-bits version of WinDBG**: See how to download here: https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063(v=vs.85).aspx
- Download NetExt: http://netext.codeplex.com/
- Unzip the file
- Copy all content of x64 folder to your 64-bits WinDBG folder (normally here if you have installed from SDK for Windows 8.1 - C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x64)
- Copy all content of x86 folder to your 32-bits WinDBG folder normally here if you have installed from SDK for Windows 8.1 - C:\Program Files (x86)\Windows Kits\8.1\Debuggers\x86)

Before working on this lab, it is recommended that you have:

- Experience with managed debugging in WinDBG or Visual Studio

Overview of the lab

This lab will demonstrate how to use NetExt to make WinDBG a data-mining tool for managed debugging.

Exercise 1: Basics

This exercise will introduce the extension and the basic commands.

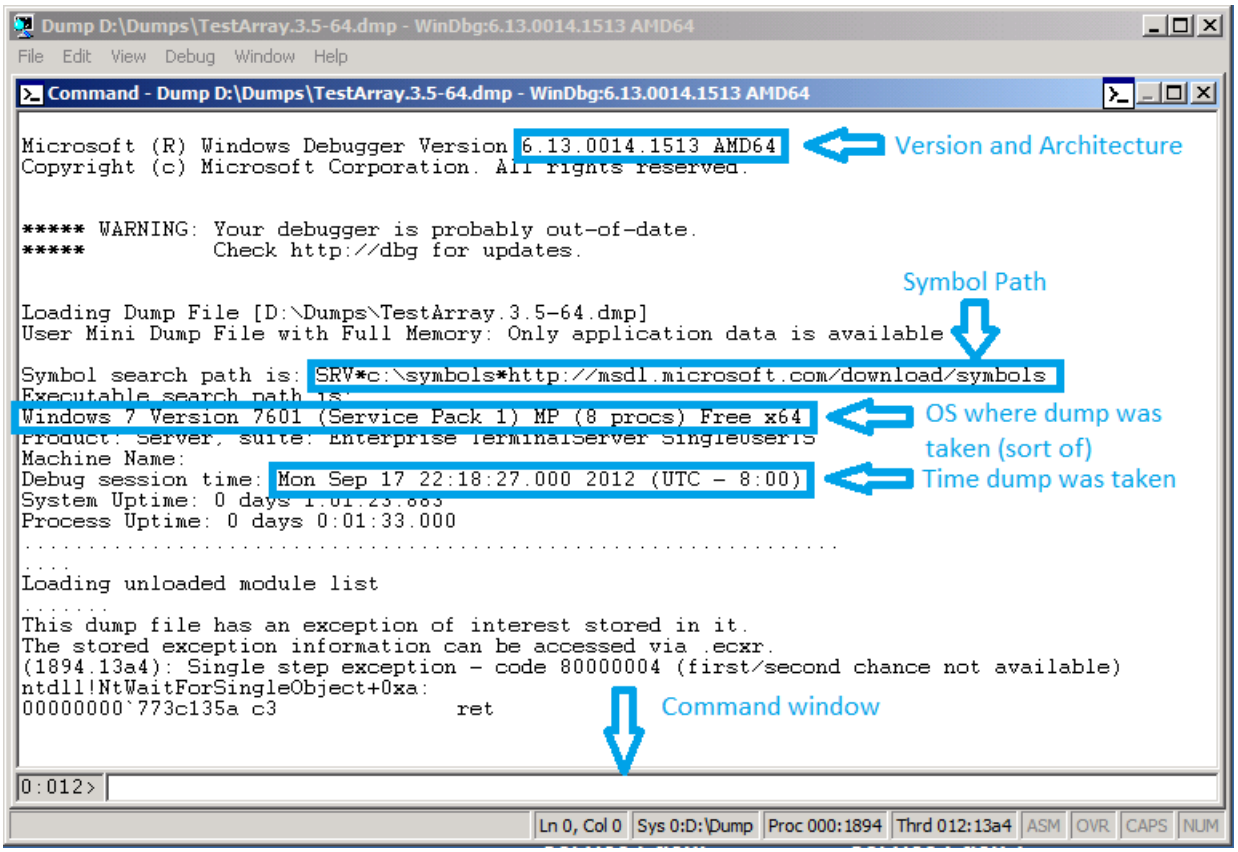
✎ IMPORTANT: Before starting the LAB run **d:\update.bat** to copy latest changes to this lab

Task 1 – How to open a dump file and choose the correct extension version to load

This task will show how to load a dump file from WinDBG, verify which extension to load the extension and count the objects in the heap.

✎ Task instructions

1. Open winddbg.exe for 64-bits (see pre-requisites for more information)
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file **TestArray.3.5-64.dmp** and click **Open**.
4. The dump file should open and you will see the following information:



✎ NOTE: The information we see when opening the dump confirms that the WinDBG we are running is 64-bits (AMD64) and it matches the dump architecture (x64). **To debug managed code they MUST match**.

5. Now let's verify the version of the .NET we are targeting by checking whether CLR.dll is present in the dump or not

✎ NOTE: .NET Runtime 2.0-3.5 is contained in mscorwks.dll while .NET Runtime 4.0 is contained in CLR.dll.

6. To check a module detail in WinDBG we type in the command lmv m<module-name>, so we type **lmv mclr** in the command window.

```
↪ lmv mclr
0:012> lmv mclr
start             end                 module name
0:012>
Ln 0, Col 0   Sys 0:D
```

NOTE: since there was no result it means the .NET Framework 4 and beyond is not loaded. We have not checked whether .NET Runtime is present or not.

7. Let's open another dump instance of WinDBG and repeat step 3 and 4 for file TestArray.4.0-64.dmp
8. In the new WinDBG instance after opening the dump, repeat step 7

```
0:011> lmv mclr
start             end                 module name
000007fe`f35f0000 000007fe`f3f54000   clr        (deferred)
    Image path: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\clr.dll
    Image name: clr.dll
    Timestamp:        Thu Dec 15 03:15:45 2011 (4EE9D6E1)
    CheckSum:         00964183
    ImageSize:        00964000
    File version:     4.0.30319.269
    Product version:  4.0.30319.269
    File flags:       8 (Mask 3F) Private
    File OS:          4 Unknown Win32
    File type:        2.0 Dll
    File date:        00000000.00000000
    Translations:     0409.04b0
    CompanyName:      Microsoft Corporation
    ProductName:      Microsoft® .NET Framework
    InternalName:     clr.dll
```

NOTE: The version in memory is 4.0.30319.269 (major=4, minor=0, build=30319, revision=**269**) which means it is .NET Runtime 4.0 and not .NET Runtime 4.5 which starts at about version 4.0.30319.**17000** (revision >= 17000).

9. Open yet another WinDBG instance to load a dump containing .NET Framework 4.5
10. Repeat step 3 and 4 for file TestArray.4.5-64.dmp
11. Repeat step 7 and you should see that CLR.DLL revision is greater than 17000 (.NET Framework 4.5).

```
0:009> lmv mclr
start             end                 module name
000007f8`3b300000 000007f8`3bc5e000   clr        (deferred)
    Image path: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\clr.dll
    Image name: clr.dll
    Timestamp:        Fri Aug 24 23:54:04 2012 (5038768C)
    CheckSum:         00967D93
    ImageSize:        0095E000
    File version:     4.0.30319.18010
    Product version:  4.0.30319.18010
    File flags:       8 (Mask 3F) Private
    File OS:          4 Unknown Win32
    File type:        2.0 Dll
    File date:        00000000.00000000
    Translations:     0409.04b0
```

12. For each architecture (32 or 64-bits) there is a different extension dll. Load the appropriate version depending on the appropriate target.

NOTE: All dump files we have opened so far are based on the same source code as shown below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TestArray
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            object[, ,] arr = new object[10, 5, 8];
            object[,] arr1 = new object[5, 8];
            for (int i = 0; i < 10; i++)
                for (int k = 0; k < 5; k++)
                    for (int l = 0; l < 8; l++)
                    {
                        if (l == 0)
                            arr[i, k, 0] = String.Format("[{0},{1},0]", i, k);
                        else
                            arr[i, k, l] = 1000 * i + k * 100 + l;
                        arr1[k, l] = k * 100 + l;
                    }
            Console.Write("Press anything...");
            Console.ReadKey();
        }
    }
}
```

13. For each opened WinDBG, load the appropriate version of the extension based on the bitness of the dump and debugger (open just one version per WinDBG instance).

```
.load netext
```

NOTE: Use .unload netext if you want to remove the extension from memory

14. To test if the extension is appropriate, let's run !windex -enumtypes to index and show the stats of the objects in heap as in this example (load netext)

```
0:009> .load netext
netext version 2.0.0.5000 Feb  9 2015
License and usage can be seen here: !whelp license
Check Latest version: !wupdate
For help, type !whelp (or in WinDBG run: '.browse !whelp')
Questions and Feedback: http://netext.codeplex.com/discussions
Copyright (c) 2014-2015 Rodney Viana (http://blogs.msdn.com/b/rodneyviana)
Type: !windex -tree or ~*e!wstack to get started

0:009> !windex -enumtypes
Starting indexing at 21:00:54 PM
Indexing finished at 21:00:54 PM
237,646 Bytes in 2,846 Objects
Index took 00:00:00
00000087c422acf0 Free (7)
000007f7dbb89690 Microsoft.VisualStudio.Debugger.Runtime.Main+_InitRuntimeDllImpl (1)
000007f7dbb89828 Microsoft.VisualStudio.Debugger.Runtime.Main+_ThrowCrossThreadMessageExcep
000007f7dbb85018 Microsoft.VisualStudio.HostingProcess.HostProc (1)
000007f7dbcf35d0 Microsoft.VisualStudio.HostingProcess.ParkingWindow (1)
```

- NOTE: At the ending of the listing there are the number of types (classes) and objects in memory
- NOTE: The memory address link on the left of the types listing is not the object address, but rather the method table, an internal structure that maintains the metadata information during runtime

15. Repeat step 14 (with the appropriate extension) for all WinDBG instances and knowing they are running the same code, verify which runtime is more efficient in the use of the heap memory.
16. **OPTIONAL:** In TestArray.4.5-64.dmp, unload netext for 64bits and load netext for 32-bits. Run command in step 14 again and see what happens (spoiler: it will crash).
17. In TestArray.4.0-64.dmp, locate the row with the type System.Object[,,,] (the object created in the application), and click on the method table link:

```
000007fef25e5880 System.Object (127)
000007ff00013f18 System.Object[,,] (1)
000007fef2606448 System.Object[,] (1)
000007fef25eac30 System.Object[] (233)
```

- NOTE: When you click on the method table it will list all objects of that type

18. Since there is only one object, click on the address link of that object

```
0:011> !windex -mt 000007ff00013f18
Index is up to date
 If you believe it is not, use !windex -flush to force reindex
0000000002671708 000007ff00013f18 System.Object[,,]     3256   0 0
```

- NOTE: The first address with link is the object address, the second is the method table, followed by the object name, heap number and GC generation

19. Verify the items of the array and check with the source code whether the values as expected or not by clicking on the array item object link

```
0:011> !wdo 0000000002671708
Address: 0000000002671708
Method Table/Token: 000007ff00013f18/200000004
Class Name: System.Object[,,]
Size : 3256
EEClass: 000007ff00013e58
Rank: 3 ( 10 5 8 )
Components: 400
Data Start: 0000000002671738
[0][0][0]: 0000000002672650 [0,0,0]
[0][0][1]: 0000000002672690
[0][0][2]: 00000000026726c0
[0][0][3]: 00000000026726f0
[0][0][4]: 0000000002672720
[0][0][5]: 0000000002672750
[0][0][6]: 0000000002672780
[0][0][7]: 00000000026727b0
[0][1][0]: 0000000002672900 [0,1,0]
```

NOTE: Some types like string are already displayed by default along with their object addresses

20. Using a more SQL-like command we can expand all items in the array. Type the command below and compare the results

```
!wselect * from 02671708
```

21. Close all WinDBG windows

Task 2 – Getting help and playing with objects in memory

This task will demonstrate the multiple ways of showing managed objects in memory.

Task instructions

1. Login to the lab machine if you have not done so
2. In the taskbar, click on the WinDBG icon  or execute c:\debuggers\winddbg.exe.
3. After WinDBG opens, select **File | Open Crash Dump…**.
4. Choose file "**w3wp.exe__ASP.NET v4.0__PID__11024__Date__10_26_2011__Time_06_32_29PM__396__Manual Dump.dmp"** and click **Open**.
5. As the name implies this is dump file of a .NET 4.0 process, so let's load netext

```
.load NetExt
```

6. To obtain hyperlink help, please issue the command below (the standard !help will work but it will show you very limited help information).

```
!whelp
```

NOTE: You may add the command you are looking for help after whelp (e.g.: !whelp wdo)

NOTE: When you choose a hyperlink, the code for the specific command is shown instead

```
Show Object Detail Commands
---------------------------
!wdo - Display ad-hoc objects or arrays from GAC or Stack
!wselect - Display ad-hoc fields (and level fields) for an
!wfrom - Perform SQL-like analysis of Heap objects enablin
*(new)* !wpe - Dump Exception Object

Enumerate objects
-----------------
!windex - index and display objects based in different fil
!wstack - dump unique stack objects
!wheap - list objects without indexing and show thottled he
!wgchandle - Dump GC root handles
*(new)* !wdae - Dump All Exceptions

Process commands
----------------
!wclrstack - Dump current stack trace (only managed thread
*(new)* !wthreads
*(new)* !wver - Show CLR version and extension version
*(new)* !wupdate - Check for update

Special
-------

!wdict - Display dictionary objects
!whash - Display HashTable objects
!whttp - List HttpContext Objects
!wconfig - Show all .config file lines in memory
```

7. You may optionally display the help in a separated window that will allow moving back and forth. Use this instead.

```
↪ .browse !whelp
```

8. Look at the help text for !wstack by clicking in hyperlink in the main help page
9. In the command window, type the command below to move thread context to thread 14

```
↪ ~14s
```

10. Type the command below to show all unique objects in thread 14's stack

```
↪ !wstack
```

```
0:014> !wstack

Listing objects from: 0000000002a25000 to 0000000002a30000 from thread: 14 [1ab0]


Address           Method Table     Heap Gen       Size Type
00000001957774c8 000007fef0ba0b40    1  0        280 System.Byte[]
00000001957775e0 000007fef0b9c950    1  0         72 System.RuntimeFieldInfoStub
00000001957ab4a8 000007fef0ba0b40    1  0       4120 System.Byte[]
0000000155980f60 000007fee79e9e18    0  0        136 System.ServiceModel.OperationCon
000000015597e340 000007fee79ce240    0  0        112 System.ServiceModel.Dispatcher.D
000000015585fa78 000007fee79a5710    0  0        256 System.ServiceModel.ServiceHost
00000001556c6938 000007fee79c19f8    0  1         88 System.ServiceModel.Dispatcher.F
00000001556fc4c8 000007fee79c5c80    0  1         48 System.ServiceModel.Diagnostics.
00000001556c6990 000007fef0b95a48    0  1         24 System.Object
000000015572e3d0 000007fee79ce578    0  1         64 System.ServiceModel.Dispatcher.M
00000001959079c8 000007fee79d2ba8    1  0        208 System.ServiceModel.Dispatcher.C
0000000155980ea8 000007fee79e9ec0    0  0         24 System.ServiceModel.OperationCon
```

- NOTE: At the end of the listing it will show you the number of unique objects in the thread's stack and the base size of the objects. The objects are actually in the heap and only the pointer is in the stack, so the 8,758 bytes reported does not mean the stack is holding 8KB of data, but rather "rooting" 8KB of heap objects.
- The listing of stack objects is not deterministic. You may see false positives. The command will also show managed stack for threads where .NET was uninitialized so you can view stack objects for abandoned threads. This last was an extension design decision to enable some special cases debugging that was not available before.

11. Click at the second object of type Byte[] and see the results

```
Address           Method Table     Heap Gen       Size Type
00000001957774c8  000007fef0ba0b40    1   0        280 System.Byte[]
00000001957775e0  000007fef0b9c950    1   0         72 System.RuntimeFie
00000001957ab4a8  000007fef0ba0b40    1   0       4120 System.Byte[]
0000000155980f60  000007fee79e9e18    0   0        136 System.ServiceMod
000000015597e340  000007fee79ce240    0   0        112 System.ServiceMod
000000015585fa78  000007fee79a5710    0   0        256 System.ServiceMod
00000001556c6938  000007fee79c19f8    0   1         88 System.ServiceMod
00000001556fc4c8  000007fee79c5c80    0   1         48 System.ServiceMod
00000001556c6990  000007fef0b95a48    0   1         24 System.Object
```

- NOTE: Some special array types (as Byte[]) will show the best representation. As Byte[] is normally used to store ASCII bytes, showing the characters is the default behavior.

12. To force the array to display as a regular array, use the following command instead

```
↪ !wdo -forcearray 00000001957ab4a8
```

13. For arrays you can also display a range of values instead of the full array. The example below shows bytes 10 to 15 of this byte array.

```
↪ !wdo -forcearray -start 0n10 -end 0n15 00000001957ab4a8
```

```
0:014> !wdo -forcearray -start 0n10 -end 0n15 00000001957ab4a8
Address: 00000001957ab4a8
EEClass: 000007fef0722310
Method Table: 000007fef0ba0b40
Class Name: System.Byte[]
Size : 4120
Rank: 1 ( 1044660549 )
Components: 4096
[10]: 0x44 (0n68)
[11]: 0x3e (0n62)
[12]: 0x3c (0n60)
[13]: 0x6c (0n108)
[14]: 0x69 (0n105)
[15]: 0x6e (0n110)
```

- CAUTION: WinDBG considers numbers as hexadecimal. To enter decimal parameters in WinDBG you should add 0n before the number as in the example shown or they hexadecimal equivalents (0a and 0f).

14. Now go back to the **!wstack** output (or issue the command again) and look for the first System.Uri object and click on the link. The object will show in a very detailed way.

```
↪ !wdo 000000019588b000
↪ Address: 000000019588b000
↪ EEClass: 000007feee995d30
↪ Method Table: 000007feeec9b358
↪ Class Name: System.Uri
↪ Size : 72
↪ Instance Fields: 7
↪ Static Fields: 21
↪ Total Fields: 28
↪ Heap/Generatin: 1/0
↪ Module: 000007feee981000
↪ Assembly: 0000000003ec58a0
```

```
↪   Domain: 000007fef3f75580
↪   Dynamic: false
↪   Assembly name: C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System\v4.0_4.0.0.0__b77a5c561934e089\System.dll
↪   Inherits: System.Uri System.Object (000007FEEEC9B358 000007FEF0B95A48)

 000007fef0b968f0                          System.String +0000                m_String 000000019571ede0
 http://rviana-serv.northamerica.corp.microsoft.com:2000/Service.svc
↪   000007fef0b968f0                          System.String +0008  m_originalUnicodeString 0000000000000000 NULL
↪   000007feeec9f698                        System.UriParser +0010                 m_Syntax 000000019568C020
↪   000007fef0b968f0                          System.String +0018            m_DnsSafeHost 0000000000000000 NULL
↪   000007feeecf2778                        System.Uri+Flags +0028        m_Flags 8c2930000 (0n37624152064)
    IPv6HostType|IPv4HostType|DnsHostType|AuthorityFound|NotDefaultPort|CanonicalDnsHost|MinimalUriInfoSet|AllUriIn
    foSet
↪   000007feeec9fbb8                     System.Uri+UriInfo +0020                  m_Info 000000015572EA20
↪   000007fef0b9d608                         System.Boolean +0030            m_iriParsing 0 (False)
↪   000007fef0b968f0 Static                   System.String +0c90           UriSchemeFile 000000019568b688 file
↪   000007fef0b968f0 Static                   System.String +0c98            UriSchemeFtp 000000019568b668 ftp
↪   000007fef0b968f0 Static                   System.String +0ca0         UriSchemeGopher 000000019568b6b0 gopher
↪   (...)
```

NOTE: Some known types like string, enums, datetime, Boolean etc show they string representation along with the internal value. m_Flags, for example, is a enum with flag attribute, so all the flags active are shown by name.

Header	Meaning
Address	The object address
EEClass	The runtime internal structure for the class
Method Table	The address of the method table for the type during runtime
Class Name	Self-explanatory
Size	The base size (not counting referenced objects but counting array elements)
Instance Fields	Non-static fields for the instance
Static Fields	Fields shared by the class among instances (it does not count towards base size)
Total Fields	Instance + Static Fields
Heap/Generation	Heap number and generation of the instance
Module	Address of the managed module structure containing the type
Assembly	Address of the managed assembly containing the type
Domain	Domain on which the object instance was loaded from
Dynamic	True if it is a dynamic type (generated during runtime)
Assembly name	Self-explanatory
Inherits	Chain inheritance list (and their Method Tables) from the current class until Object

15. Use the command below to display the object without the headers

```
↪   !wdo -noheader 000000019588b000
```

```
000007fef0b9d608                          System.Boolean +0030            m_iriParsing 0 (False)
000007fef0b968f0 Static                    System.String +0c90           UriSchemeFile 000000019568b688 file
000007fef0b968f0 Static                    System.String +0c98            UriSchemeFtp 000000019568b668 ftp
```

0 1 2 3 4 5 6

Column	Meaning
0	Address of the method definition table of the field type
1	Empty for instance fields, static otherwise

Column	Meaning
2	Type name
3	Field offset within the instance for instance fields (add the size of the pointer for real offset) or relative offset of the static field in the domain or module (in real life, offset for static fields are not very useful).
4	Field name
5	Content: raw pointer link or hex and decimal representation if the type is basic (int, uint, etc)
6	Advanced string representation of a well-known type (like datetime, string, guid, ect)

16. If you do not need all details, !wselect can offer a much more simplified output, try the command below.

```
↪ !wselect * from 000000019588b000
```

```
0:014> !wselect * 000000019588b000
Syntax error:
wselect * 000000019588b000
0:014> !wselect * from 000000019588b000
[System.Uri]
System.String m_String = 000000019571ede0 http://rviana-serv.northamerica.corp.microsoft.com:
System.String m_originalUnicodeString = 0000000000000000 NULL
System.UriParser m_Syntax = 000000019568C020
System.String m_DnsSafeHost = 0000000000000000 NULL
(uint64)System.Uri+Flags m_Flags = 8c2930000 (0n37624152064) IPv6HostType|IPv4HostType|DnsHos
System.Uri+UriInfo m_Info = 000000015572EA20
(bool)System.Boolean m_iriParsing = 0 (False)
static System.String UriSchemeFile = 000000019568b688 file
static System.String UriSchemeFtp = 000000019568b668 ftp
static System.String UriSchemeGopher = 000000019568b6b0 gopher
static System.String UriSchemeHttp = 000000019568b618 http
static System.String UriSchemeHttps = 000000019568b640 https
static System.String UriSchemeMailto = 000000019568b728 mailto
static System.String UriSchemeNews = 000000019568b700 news
static System.String UriSchemeNntp = 000000019568b6d8 nntp
static System.String UriSchemeNetTcp = 000000019568b7c8 net.tcp
static System.String UriSchemeNetPipe = 000000019568b7f0 net.pipe
static System.String SchemeDelimiter = 000000019568b5f8 ://
static Microsoft.Win32.IInternetSecurityManager s_ManagerRef = 0000000000000000
static System.Object s_IntranetLock = 000000019568c250
static (bool)System.Boolean s_ConfigInitialized = 0 (False)
static (bool)System.Boolean s_ConfigInitializing = 0 (False)
static (int32)System.UriIdnScope s_IdnScope = 0 (0n0) None
static (bool)System.Boolean s_IriParsing = 0 (False)
static System.Object s_initLock = 0000000000000000
static System.Char[] HexUpperChars = 000000019568C268
static System.Char[] HexLowerChars = 000000019568C2A0
static System.Char[] _WSchars = 000000019568C2D8
```

17. To show only the fields you want to list, you may use something like the command below.

```
↪ !wselect m_String, m_Flags from 000000019588b000
↪ [System.Uri]
↪ System.String m_String = 000000019571ede0 http://rviana-serv.northamerica.corp.microsoft.com:2000/Service.svc
↪ (uint64)System.Uri+Flags m_Flags = 8c2930000 (0n37624152064)
  IPv6HostType|IPv4HostType|DnsHostType|AuthorityFound|NotDefaultPort|CanonicalDnsHost|MinimalUriInfoSet|AllUriIn
  foSet
```

NOTE: You can also list inner fields using !wselect the same way you reference inner fields in C# (with . to separate inner fields).

18. Use the command below to list important inner fields in a HttpRequest object.

```
↪ !wselect _utcTimestamp, _request._httpMethod,
  _request._url.m_String, _response._statusCode from
  00000001956f77a8
↪ [System.Web.HttpContext]
↪ System.DateTime _utcTimestamp = -mt 000007FEF0BB96C8 00000001956F78D8 10/26/2011 11:29:15 PM
↪ System.String _request._httpMethod = 00000001956fa860 GET
↪ System.String _request._url.m_String = 0000000195711240 http://localhost:2000/Service.svc
↪ (int32)System.Int32 _response._statusCode = c8 (0n200)
```

Though we have seen that !wselect is a good help to generate diagnostic-friendly output specially for array as we saw in the previous task, we will see the unleashed power of data-mining by using the command wfrom in the Exercise 2

Exercise 2: The cool Data Mining stuff

This exercise will build the knowledge to create complex queries and elaborated query output based on field filtering and inheritance chain

Task 1 – How to index the heap and generate a tree from all objects in the heap

This task will show how to index a dump file, generate a type tree, perform click-and-go object exploration and perform elaborated object queries.

Task instructions

1. Open WinDBG 64-bits
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file "**w3wp.exe__ASP.NET v4.0__PID__11024__Date__10_26_2011__Time_06_32_29PM__396__Manual Dump.dmp"** and click **Open**.
4. As the name implies this is dump file of a .NET 4.0 process, let's load netext

```
↪ .load netext
```

We will use !wheap to investigate the heap for this process. We used !windex in Exercise 1 to enumerate the types. !windex is almost always the preferred command to deal with heap objects as windex as the name implies, keep an index of heap objects for reuse. If it is a dump file there is no need for reindexing. For live targets and iDNA traces the heap is reindexed if necessary. It is also possible to force reindexing manually if you feel the extension missed a change in the heap. The !wheap should only be used to display heap structure information and/or heap objects sampling. Using !wheap to filter objects is never a good idea.

5. Use the command below to display the heap structure and memory areas

```
↪ !wheap -detailsonly
↪ Heaps: 2
↪ Server Mode: 1
↪
↪ Heap Areas:
↪ Area [0000000155660068]: 0000000155660068-0000000155660080 (          24) Heap: 0 Generation: 2 Large: 0
↪ Area [0000000155660080]: 0000000155660080-0000000155793728 (   1,259,176) Heap: 0 Generation: 1 Large: 0
↪ Area [0000000155793728]: 0000000155793728-0000000155a448d0 (   2,822,568) Heap: 0 Generation: 0 Large: 0
↪ Area [0000000155a45020]: 0000000155a45020-0000000155a472c0 (       8,864) Heap: 0 Generation: 0 Large: 0
↪ Area [0000000155a49020]: 0000000155a49020-0000000155a54000 (      45,024) Heap: 0 Generation: 0 Large: 0
↪ Area [0000000155a54fe8]: 0000000155a54fe8-0000000155a56fe8 (       8,192) Heap: 0 Generation: 0 Large: 0
↪ Area [0000000195660068]: 0000000195660068-0000000195660080 (          24) Heap: 1 Generation: 2 Large: 0
↪ Area [0000000195660080]: 0000000195660080-000000019573e270 (     909,808) Heap: 1 Generation: 1 Large: 0
↪ Area [000000019573e270]: 000000019573e270-000000019573e2e0 (         112) Heap: 1 Generation: 0 Large: 0
↪ Area [0000000195740288]: 0000000195740288-00000001957ad678 (     447,472) Heap: 1 Generation: 0 Large: 0
↪ Area [00000001957ae288]: 00000001957ae288-00000001957aedd0 (       2,888) Heap: 1 Generation: 0 Large: 0
↪ Area [00000001957b0288]: 00000001957b0288-0000000195907af8 (   1,407,088) Heap: 1 Generation: 0 Large: 0
↪ Area [00000001d5660068]: 00000001d5660068-00000001d569bf10 (     245,416) Heap: 0 Generation: 3 Large: 1
↪ Area [00000001e5660068]: 00000001e5660068-00000001e5660080 (          24) Heap: 1 Generation: 3 Large: 1
↪
↪ Total Bytes Used for GC objects: 7,156,680
↪
```

Since the framework is in server mode (Server Mode=1), .NET will create a heap for each processor core (2 in this case). In workstation mode there is only one heap. Each heap will contain its separated structure and allocations. Each heap also keeps its own segments. The generation information is kept in the structure as well

and except for the ephemeral segment it may span on more than one segment. There are 4 generation tables, gen 0 is where the newly allocated objects are, gen 1 and gen 2 contain objects that were not garbage collected at least during one garbage collection process. Gen 3 is not actually a generation but rather where the large objects are kept since moving large objects are much more process intensive. Heap Areas in the listing show you the logical memory used by heap and generation.

6. To display a sampling list of the first 500 objects of each area, please type the command below.

```
↪ !wheap
↪ (...)
↪ 00000001d5693f40 0000000000e111d0       24   0 3 Free
↪ 00000001d5693f58 000007fef0b9adf8    32672   0 3 System.Object[]
↪ 00000001d569bef8 0000000000e111d0       24   0 3 Free
↪ 00000001e5660068 0000000000e111d0       24   1 3 Free
↪ This output was throttled. Only the first 500 objects of each heap range has been shown.
↪ Use -nothrottle to list all objects or any limiting parameter (-type for example)
```

NOTE: The first column is the object address, the second contains the method table, followed by base size, heap and generation (gen 3 means it is in the large objects segment) and type name. Free is not actually a type, it is a marking to make it easy for GC to clear memory space without relocating objects if it is not necessary.

From this point on we will not use wheap at all. We suggest you do not use wheap beyond what we showed here. All heap walking will be done using the heap index that we will create with !windex.

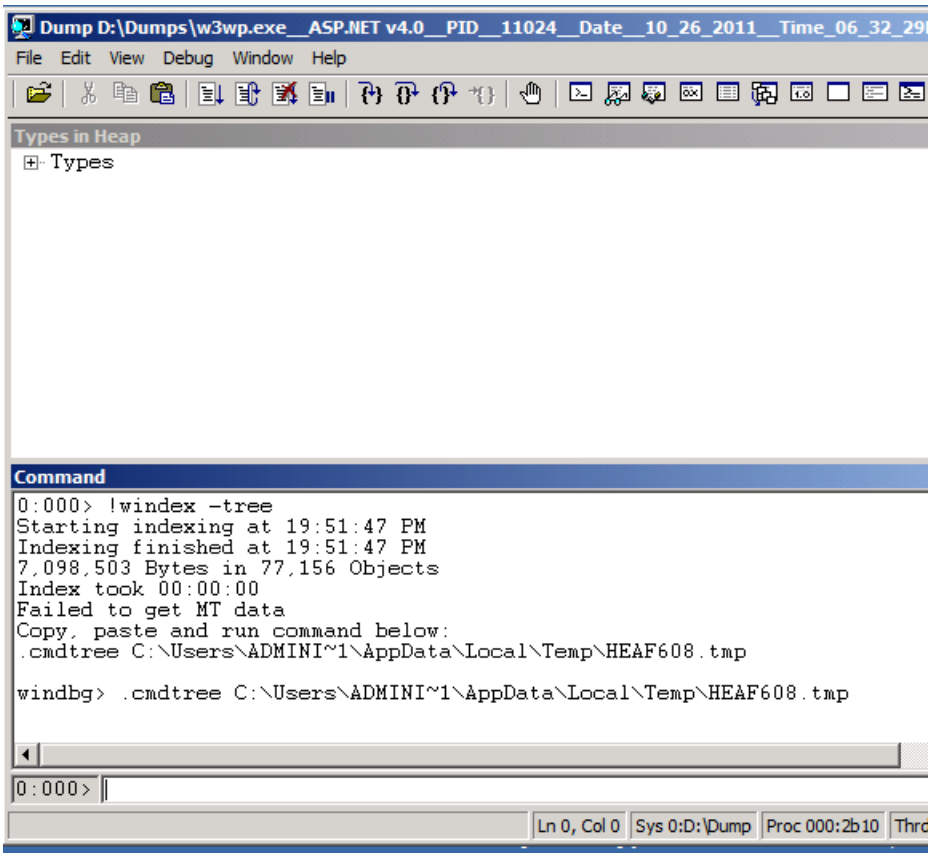
7. Index heap and create an interactive tree representation of all types in heap using the command below.

```
↪ !windex -tree
```

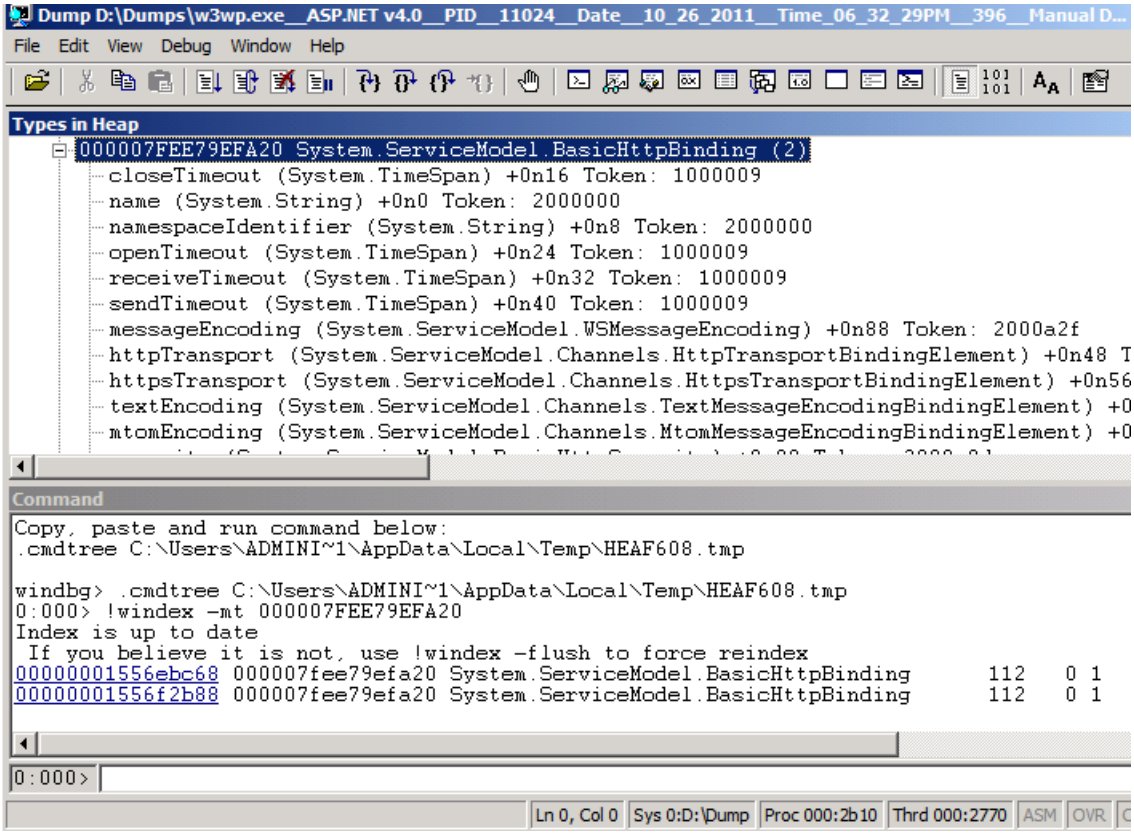
```
0:000> !windex -tree
Starting indexing at 19:51:47 PM
Indexing finished at 19:51:47 PM
7,098,503 Bytes in 77,156 Objects
Index took 00:00:00
Failed to get MT data
Copy, paste and run command below:
.cmdtree C:\Users\ADMINI~1\AppData\Local\Temp\HEAF608.tmp
```

NOTE: The temporary file name will be different every time you generate a tree view. For lack of supported API to load the generated tree you will have to copy and paste the generated command to open the tree window. It is always safe to ignore the error “Failed to get MT data”.

8. Copy and output command and paste in the command window (the output will be different every time you run the command).
9. You may want to re-arrange the windows position as below to have a better environment to investigate your objects

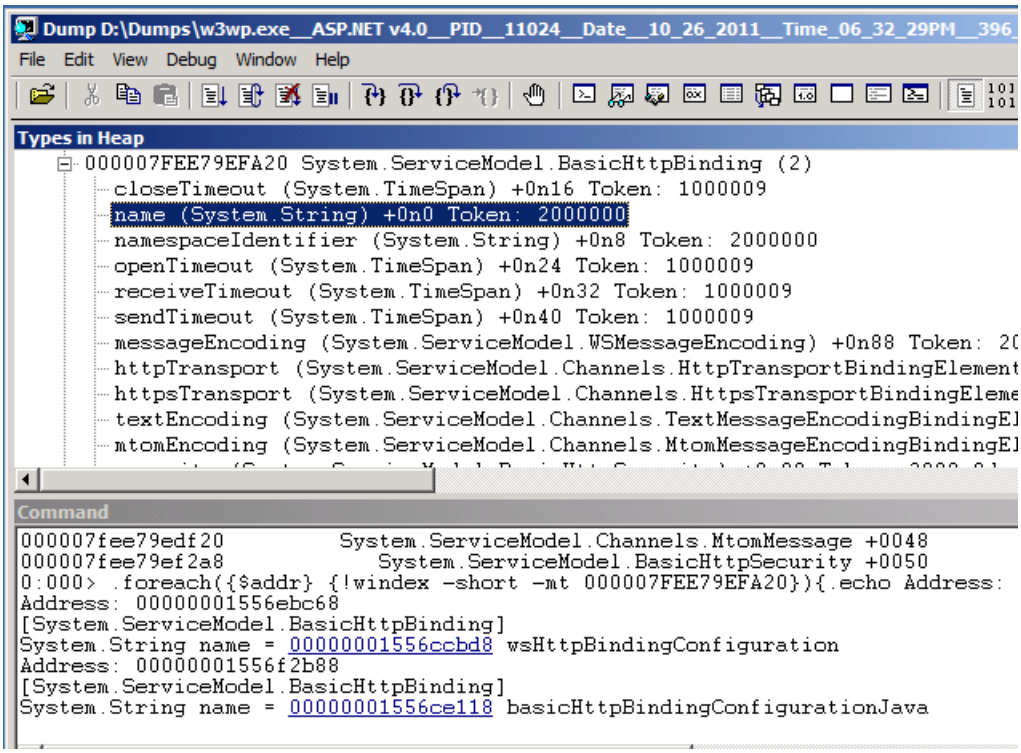


10. Click on the + button in Types to expand all types. Locate the type System.ServiceModel.BasicHttpBinding and click on the plus button to expand the fields.
11. Double click on the selected type to display the 2 objects of type System.ServiceModel.BasicHttpBinding.



NOTE: The double click will list all objects with the method table of the selected type. As you could see the underlying command used was !windex with parameter -mt, leveraging the index already and memory.

12. Click on the first link to show the object detail.
13. Now, back on the type's tree window, double click on the field name to display only this field for all objects.



NOTE: Though this .foreach command can help automate object listing by selected fields and seems like a natural extension for what we learned about wselect in the previous task, this is not a good approach if you are already familiar with the object. As we will learn later, the real data-mining command is !wfrom.

14. Since we learned a little bit more about what we have in memory, we will list all BasicHttpBinding objects using the command below.

```
↪ !windex -type *.BasicHttpBinding
↪ Index is up to date
↪  If you believe it is not, use !windex -flush to force reindex
↪ 00000001556ebc68 000007fee79efa20 System.ServiceModel.BasicHttpBinding      112   0 1
↪ 00000001556f2b88 000007fee79efa20 System.ServiceModel.BasicHttpBinding      112   0 1
```

NOTE: We use a wildcard string (*.BasicHttpBinding) to avoid typing the full type. Wildcard is allowed in parameter -type. The first column of the output is the object address, the second is the method table followed by base size, heap and generation.

Now we will present the !wfrom which along with !windex is the base for all data-mining in the extension. !wfrom is very powerful but it requires a lot of research on the targeted types with !wdo and !wselect before you are able to create a valid request. There is no defaults and known-types when use !wfrom and all the work is on the engineer.

15. We will now use !wfrom to list only the field name for all objects of type System.ServiceModel.BasicHttpBinding. Use the command below.

```
↪ !wfrom -type *.BasicHttpBinding select name
↪ name: wsHttpBindingConfiguration
↪ name: basicHttpBindingConfigurationJava
↪
↪ 2 Object(s) listed
```

16. In order to show the object address in the output we will need to add an extension's built-in function - $addr() – to the command, as shown below.

```
↪ !wfrom -type *.BasicHttpBinding select $addr(),name
↪ calculated: 00000001556EBC68
↪ name: wsHttpBindingConfiguration
↪ calculated: 00000001556F2B88
↪ name: basicHttpBindingConfigurationJava
↪
↪ 2 Object(s) listed
```

NOTE: All calculated results (built-in functions and expressions) show "calculated" as field name. if you use the function $a() you will create an alias to the calculated field. We will see more about functions later and you can learn more using !whelp functions.

17. To try an even closer result we can add the type name in the results. Use the command below.

```
↪ !wfrom -type *.BasicHttpBinding select $a("Address",$addr()),
  $typename(), name
↪ Address: 00000001556EBC68
↪ calculated: System.ServiceModel.BasicHttpBinding
↪ name: wsHttpBindingConfiguration
↪ Address: 00000001556F2B88
↪ calculated: System.ServiceModel.BasicHttpBinding
↪ name: basicHttpBindingConfigurationJava
↪
↪ 2 Object(s) listed
```

Task 2 – Advanced querying and data-mining

In this task, we will leverage !wfrom/!windex as data-mining commands to operate on filtered objects and arrays to create complex reports. We will also use !wclass to display the class layout.

Task instructions

18. Open WinDBG 64-bits
19. After WinDBG opens, select **File | Open Crash Dump…**.
20. Choose file "**leakstack.dmp"** and click **Open**.
21. This is a SharePoint 2010 dump file (.NET 2.0 process), let's load netext

```
↪ .load netext
```

22. Index the heap (no need to show the tree, tree)

```
↪ !windex
```

23. Type the command below to show fields _commandText and _commandType as shown below and press enter (this command is a mix of WinDbg loop and WCFWIFEXT).

```
↪ .foreach({$addr} {!windex -short -mt 000007FEE7E1D180}){.echo
  Address: {$addr}; !wselect _commandText, _commandType from
  {$addr}}
```

- NOTE: The output will show the command string and the command type both as numerical and the correspondent enum name (1=Text, 4=StoredProcedure)

24. Use the command below to display only commands that are Text (or 1) using !wfrom.

```
!wfrom -type *.sqlcommand where (_commandType == 1) select
_commandText, _commandType
```

- NOTE: Unlike !wselect, !wfrom never sugarcoats the output and the enumeration is shown as their memory type (int32 for _commandType). To enable a more readable output, it is recommended that you use a transforming function.

25. Use the command below to display the object address and _commandType enumeration name instead of int32 representation (filtered by stored procedure this time). The first asterisks string is to make it easy to see where the object starts.

```
↪ !wfrom -type *.sqlcommand where (_commandType == 4) select
  "**********",$addr(), _commandText, $enumname(_commandType),
  _parameters._items._items
↪ calculated: **********
↪ calculated: 00000001011FCD90
↪ _commandText: dbo.proc_getSiteIdOfHostHeaderSite
↪ calculated: StoredProcedure
↪ _parameters._items._Items: 00000001011FCF70
↪ calculated: **********
↪ calculated: 0000000101686948
↪ _commandText: dbo.proc_getSiteIdOfHostHeaderSite
↪ calculated: StoredProcedure
↪ _parameters._items._Items: 0000000101686B28
↪ (...)
↪ 1,019 Object(s) listed
↪ 349 Object(s) skipped by filter
```

- NOTE: All columns generated via an expression (like "******") or function ($enumname) will show "calculated" as they are not an object field.
- NOTE: For brevity, we skipped the process of identifying which inner field (_parameters._items._items in this case) contains the actual parameters list. This process was already explained in the previous task in this exercise for another object.

26. Use the command below to list the help for all functions to serve as reference for the remaining commands

```
↪ .browse !whelp functions
```

One of the common issues in SharePoint is to run a query that is not throttled in a large database. Below we will list all queries that are not throttled (they do not have "SELECT TOP"). We will search all SELECT commands that are not SELECT TOP.

27. Run the following command:

```
↪ !wfrom -type *.SqlCommand where ( $contains(_commandText,
  "SELECT") && (!$contains(_commandText,"SELECT TOP")) ) select
  _commandText
```

```
0:067> !wfromˆ -type *.SqlCommand where ( $contains(_commandText, "SELECT
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:  SELECT t1.[Type] AS c0, t3.[nvarchar4] AS c12c9, UserData
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:  SELECT t1.[Type] AS c0, t3.[nvarchar4] AS c12c9, UserData
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:  SELECT t1.[Type] AS c0, t3.[nvarchar4] AS c12c9, UserData
_commandText:   SELECT t1.[TimeCreated] AS c0, t3.[nvarchar4] AS c10c5,
_commandText:  SELECT t1.[Type] AS c0, t3.[nvarchar4] AS c12c9, UserData

10 Object(s) listed
1,358 Object(s) skipped by filter
```

NOTE: The syntax is similar to C++ and C#. Logical AND is && and NOT is !. You can see more details running !whelp expression

Some classes are abstract, some are normally found only in their inherited form. This make the regular -type in !wfrom and !windex not to be very useful to find these types. The same is valid when you are trying to identify types that may hold references to another type: !windex and !wfrom offer some powerful inheritance filter.

28. To identify all types that implements System.IO.Stream, type:

```
↪ !windex -implement System.IO.Stream
↪ Index is up to date
↪  If you believe it is not, use !windex -flush to force reindex
↪ 000000010191f7d0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream
  88   0 0
↪ 0000000101dd0f78 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream
  88   0 0
↪ 00000001023a74f0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream
  88   0 0
↪ 000000010294abc8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream
  88   0 0
↪ (...)
↪ 000000010167e618 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 000000010168e3f8 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 000000010169c088 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 000000010171e380 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 0000000101732468 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 000000010173b9f0 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ 00000001017683c8 000007fef2f8ec28 System.IO.MemoryStream       80   0 0
↪ (...)
↪ 00000000ffc25670 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
↪ 00000001011f8358 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
↪ 0000000101201ff8 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
↪ 000000010120e5e0 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
↪ 0000000101212c80 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
↪ 0000000101215b98 000007feead2fd68 System.Web.HttpResponseStream       64   0 2
(...)
```

29. Click on the object it inherits from (System.IO.Stream initially) until you reach a class that implements System.Object
30. To display all exceptions, use the command below, this is especially helpful in dumps for .NET 4.5 that do not have an implementation of psscor and thus you cannot run !dae. The command is as below:

```
↪ !windex -implement System.Exception
```

31. To list all objects containing fields of type System.IO.Stream, type the command below

```
↪ !windex -fieldtype *.Stream
```

```
0:067> !windex -fieldtype *.Stream
Index is up to date
 If you believe it is not, use !windex -flush to force reindex
0000000010191f7d0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000101dd0f78 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001023a74f0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
000000010294abc8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001030f5db8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001039142f8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000140da5320 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001411ce438 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000141877638 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000141a59e10 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001421da8d0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000142655a28 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
000000014273c4a8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001437bf058 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000143c02468 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000144296438 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001449a5070 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
000000018140d0d0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001818e5e98 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000181deaee8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001826c0698 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000018301d120 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
0000000183965e10 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001843d9ab0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001844fab60 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c11fe3e8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c23fb518 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c309d3a8 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c381d120 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c3decb58 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c45c8888 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000001c4a8e9b0 000007feeec55290 Microsoft.SharePoint.CoordinatedStreamBuffer.SPCoordinatedNativeBufferStream       88
00000000ffc45c60 000007feeec17f30 Microsoft.SharePoint.Upgrade.SPXmlConfiguration      336   0 2
00000000ffc5ac70 000007feeec17f30 Microsoft.SharePoint.Upgrade.SPXmlConfiguration      336   0 2
```

Task 3 – Making the output looks good

In this task, we will leverage !wfrom/!windex as data-mining commands to operate on filtered objects and arrays to create complex reports. We will also use !wclass to display the class layout.

Task instructions

1. Load WinDBG 64-bits.
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file "**leakstack.dmp"** and click **Open**.
4. Load netext

```
↪ .load netext
```

5. Run the command below to index the heap:

```
↪ !windex
```

6. Run the command below to show HttpContext information via !wfrom as we learned from previous lesson with !wselect

```
↪ !wfrom -type *.HttpContext select $addr(),_utcTimestamp,
  _request._httpMethod, _request._url.m_String,
  _response._statusCode
↪ calculated: 00000000FFC25190
↪ _utcTimestamp: -mt 000007FEF2FD8040 00000000FFC252B8
↪ _request._httpMethod: POST
↪ _request._url.m_String: http://sp:80/_vti_bin/Lists.asmx
↪ _response._statusCode: 0n200
↪ calculated: 00000001011F7E78
↪ _utcTimestamp: -mt 000007FEF2FD8040 00000001011F7FA0
↪ _request._httpMethod: GET
↪ _request._url.m_String: http://sp.contoso.com:80/_vti_bin/SiteData.asmx
```

```
↪ _response._statusCode: 0n401
↪ calculated: 00000001011FD7E0
↪ _utcTimestamp: -mt 000007FEF2FD8040 00000001011FD908
↪ (...)
```

NOTE: The output is not so appealing as the object address is showing as "calculated" and the known type DateTime is not showing anything meaningful. We will use functions to make it more reporting ready. For help on functions, run !whelp functions

7. Use this function for a slight improvement on the output:

```
↪ !wfrom -type *.HttpContext select $addr(),
  $tickstodatetime(_utcTimestamp.dateData), _request._httpMethod,
  _request._url.m_String, _response._statusCode
↪ calculated: 00000000FFC25190
↪ calculated: 1/17/2013 10:44:34 PM
↪ _request._httpMethod: POST
↪ _request._url.m_String: http://sp:80/_vti_bin/Lists.asmx
↪ _response._statusCode: 0n200
↪ calculated: 00000001011F7E78
↪ calculated: 1/17/2013 10:44:34 PM
↪ _request._httpMethod: GET
↪ _request._url.m_String: http://sp.contoso.com:80/_vti_bin/SiteData.asmx
↪ _response._statusCode: 0n401
↪ (...)
```

8. NOTE: Datetime is a type value (structure) where the inner field dateData holds the ticks. Since the function $tickstodatetime requires the ticks we should use _utcTimestamp.dateData.
9. We can use C++/C# escape sequences to improve the output as in the command below (/t = TAB):

```
↪ !wfrom -nofield -nospace -type *.HttpContext select $addr()," ",
  $tickstodatetime(_utcTimestamp.dateData), " ",
  _request._httpMethod, "\t", _request._url.m_String, " (",
  _response._statusCode, ")"
↪ 00000000FFC25190 1/17/2013 10:44:34 PM POST  http://sp:80/_vti_bin/Lists.asmx (0n200)
↪ 00000001011F7E78 1/17/2013 10:44:34 PM GET   http://sp.contoso.com:80/_vti_bin/SiteData.asmx (0n401)
↪ 00000001011FD7E0 1/17/2013 10:44:34 PM POST  http://sp.contoso.com:80/_vti_bin/Lists.asmx (0n200)
↪ 000000010120E100 1/17/2013 10:44:34 PM POST  http://sp.contoso.com:80/_vti_bin/Lists.asmx (0n200)
↪ 00000001012127A0 1/17/2013 10:44:34 PM POST  http://sp.contoso.com:80/_vti_bin/Lists.asmx (0n401)
↪ 00000001012156B8 1/17/2013 10:44:34 PM GET   http://sp.contoso.com:80/FormServerTemplates/Forms/All Forms.aspx
  (0n401)
↪ 000000010122E818 1/17/2013 10:44:58 PM GET   http://sp.contoso.com:80/SitePages/Forms/AllPages.aspx (0n200)
↪ 0000000101265D78 1/17/2013 10:45:07 PM GET   http://sp.contoso.com:80/Lists/Announcements/DispForm.aspx?ID=1
  (0n200)
↪ (...)
```

NOTE: -nofield is hiding the field name and -nospace is preventing a new line between fields.

10. OPTIONAL: Run this very complex query to obtain the list of HttpContext in memory:

```
↪ !wfrom -nospace -nofield -type *.HttpContext select
  $rpad($addr(),10)," ",$if(!_thread, "  --
  ",$lpad($thread(_thread.DONT_USE_InternalThread),4))," 
  ",$if((_timeoutSet==1),$tickstotimespan(_timeout._ticks), "Not set
  "), " ", $if(_response._completed ||
  _finishPipelineRequestCalled,"Finished", $tickstotimespan($now()-
  _utcTimestamp.dateData)), " ",
  $replace($lpad(_response._statusCode,8),"0n",""), " ",
```

```
    $rpad($isnull(_request._httpMethod,"NA"),8), " ",
    $isnull(_request._url.m_String, _request._filePath._virtualPath)
↪   0000000184167918   -- 00:01:50 Finished    401 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   00000001842FFD10   -- 00:01:50 Finished    200 GET      http://sp:80/Lists/Links/AllItems.aspx
↪   0000000184346F98   53 00:01:50 00:00:41    200 GET      http://sp:80/Lists/Team Discussion/DispForm.aspx?ID=1
↪   0000000184B4F248   -- 00:01:50 Finished    401 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   0000000184CE79A8   -- 00:01:50 Finished    401 GET      http://sp.contoso.com:80/SitePages/Forms/AllPages.aspx
↪   0000000184D0CA50   -- 00:01:50 Finished    401 GET      http://sp:80/Lists/Announcements/AllItems.aspx
↪   0000000184D262F8   -- 00:01:50 Finished    401 POST     http://sp:80/_vti_bin/Lists.asmx
↪   0000000184D2E120   -- 00:01:50 Finished    401 GET      http://sp:80/Lists/Team Discussion/AllItems.aspx
↪   0000000184D47FB8   -- 00:01:50 Finished    401 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   0000000184D512B8   -- 00:01:50 Finished    401 GET      http://sp:80/Lists/Team Discussion/AllItems.aspx
↪   0000000184D58F48   49 00:01:50 00:00:41    200 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   0000000184E1E8D0   -- 00:01:50 Finished    401 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   0000000184E3DC00   -- 00:01:50 Finished    401 GET      http://sp.contoso.com:80/FormServerTemplates/Forms/All
    Forms.aspx
↪   0000000184E46F68   -- 00:01:50 Finished    401 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪   0000000184E52280   55 00:01:50 00:00:41    200 GET      http://sp:80/Lists/Team Discussion/AllItems.aspx
↪   0000000184EBF228   47 00:01:50 00:00:41    200 POST     http://sp.contoso.com:80/_vti_bin/Lists.asmx
↪
```

NOTE: This is a similar output you can obtain with the special command !whttp

Exercise 3: Special Purpose Commands

This exercise will show some special purpose commands

Task 1 – Using !whttp, !whash, !wdict and wconfig special purpose commands

This task will show how to use special purpose commands .

Task instructions

1. Open WinDBG 64-bits.
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file "**leakstack.dmp"** and click **Open**.
4. Let's load netext

```
↪ .load netext
```

5. Index the heap and display the runtime information for all Http Runtime objects (check the number of active requests to be 11):

```
↪ !windex
↪ !wruntime
```

```
0:040> !wruntime
Runtime Settings per Application Pool

=====================================================================
Address          : 000000013FB90990
First Request    : 1/17/2013 10:41:47 PM
App Pool User    : CONTOSO\SPSvc
Trust Level      : WSS_Minimal
App Domnain Id   : /LM/W3SVC/1067026433/ROOT-1-130029361030647561
Debug Enabled    : False
Active Requests  : 0n11
Path             : C:\inetpub\wwwroot\wss\VirtualDirectories\80\ (local disk)
Temp Folder      : C:\Windows\Microsoft.NET\Framework64\v2.0.50727\Temporary ASP.NET Files
Compiling Folder: C:\Windows\Microsoft.NET\Framework64\v2.0.50727\Temporary ASP.NET Files\root\93c52e73\59b7e362
Shutdown Reason  : Not shutting down
```

6. Use the following commands to show only running requests ordered by request time:

```
↪ !whttp -order -running
```

7. Locate the first running request with verb POST:

```
0:067> !whttp -order -running
HttpContext      Start Time                    Thread Time Out Running  Status Verb     Url
000000014213e818 1/17/2013 10:45:09 PM             57 00:01:50 00:00:41    200 GET      http://sp.c
000000001027b9c40 1/17/2013 10:45:09 PM            52 00:01:50 00:00:41    200 GET      http://sp.c
00000001c30d2878 1/17/2013 10:45:09 PM             40 00:01:50 00:00:41    200 POST     http://sp.c
0000000184346f98 1/17/2013 10:45:09 PM             53 00:01:50 00:00:41    200 GET      http://sp:8
00000001c3773a48 1/17/2013 10:45:09 PM             60 00:01:50 00:00:41    200 GET      http://sp:8
000000001448b6888 1/17/2013 10:45:09 PM            56 00:01:50 00:00:41    200 GET      http://sp:8
0000000184d58f48 1/17/2013 10:45:09 PM             49 00:01:50 00:00:41    200 POST     http://sp.c
0000000184e52280 1/17/2013 10:45:09 PM             55 00:01:50 00:00:41    200 GET      http://sp:8
0000000184ebf228 1/17/2013 10:45:09 PM             47 00:01:50 00:00:41    200 POST     http://sp.c
000000001045f2200 1/17/2013 10:45:09 PM            58 00:01:50 00:00:41    200 GET      http://sp.c
0000000103f94698 1/17/2013 10:45:14 PM             -- Not set  00:00:36    200 POST     /_vti_bin/L

11 HttpContext object(s) found matching criteria

212 HttpContext object(s) skipped by filter
```

- IMPORTANT: Notice that the number of running contexts are the same of the number of active connections even though one of the requests did not have a thread assigned to yet.

8. This is equivalent of typing:

```
↪ !whttp 00000001c30d2878
```

```
0:067> !whttp 00000001c30d2878

Context Info
================================
Address            : 00000001c30d2878
Target/Dump Time   : 1/17/2013 10:45:51 PM
Request Time       : 1/17/2013 10:45:09 PM
Running time       : 00:00:41
Timeout            : 00:01:50
Timeout Start Time: 1/17/2013 10:45:09 PM
Timeout Limit Time: 1/17/2013 10:46:59 PM
Managed Thread Id : 9e8
Managed Thread Id : a
HttpContext.Items[]: 00000001c3109ae8

Request Info
================================
POST http://sp.contoso.com:80/_vti_bin/SiteData.asmx
Content Type     : text/xml; charset=utf-8
Content Length   : 453
Target in Server: C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\isapi\SiteData.asmx
Body:
[--- Start ---]
<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmln
[---  End ---]

Response Info
================================
Warning: Response has not completed
Status           : 200 (NULL)

Server Variables
================================
ALL_HTTP: HTTP_CONTENT_LENGTH:453
HTTP_CONTENT_TYPE:text/xml; charset=utf-8
HTTP_AUTHORIZATION:NTLM TlRMTVNTUAADAAAAAAAAAFgAAAAAAAAAWAAAAAAAAABYAAAAAAAAAFgAAAAAAAAAWAAAAAAAAABYAAAANcKI4gYE
HTTP_EXPECT:100-continue
HTTP_HOST:sp.contoso.com
HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 2.0.50727.5448)
HTTP_SOAPACTION:"http://schemas.microsoft.com/sharepoint/soap/GetContent"

ALL_RAW: Content-Length: 453
Content-Type: text/xml; charset=utf-8
Authorization: NTLM TlRMTVNTUAADAAAAAAAAAFgAAAAAAAAAWAAAAAAAAABYAAAAAAAAAFgAAAAAAAAAWAAAAAAAAABYAAAANcKI4gYBsR0A
Expect: 100-continue
Host: sp.contoso.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 2.0.50727.5448)
```

- IMPORTANT: This listing tell us the thread running the request for active requests, the authorization type (NTLM in this case – rather because the token starts with T than the name NTLM), the remote and local addresses, user that made the request, the body of the request when applicable among other details.

9. Add the end of the Http Context listing you will see some recommendation. Try the ones to show the formatted soap message and the one to show the Xml as a tree, equivalent to !wfrom -nofield -nospace -obj 00000001c30d2878 select $xml($rawfield(_request._rawContent._data)) and !wfrom -nofield -nospace -obj 00000001c30d2878 select $xmltree($rawfield(_request._rawContent._data))

10. Click on the "Managed Thread Id" link in the http context detail to select the thread running the request (this is equivalent to):

```
↪ ~~[9e8]s
↪ ntdll!ZwAlpcSendWaitReceivePort+0xa:
↪ 00000000`77891b6a c3              ret
```

11. Verify the objects in the stack of this thread:

```
↪ !wstack
```

12. Identify a Hashtable in the stack at 000000010449d178 and click on it. It is the equivalent to:

```
↪ !wdo 000000010449d178
```

13. As the output is not useful let's use the special purpose command to see the hash table below:

```
↪ !whash 000000010449d178
```

```
0:040> !whash 000000010449d178
Buckets : 00000001045b0c90

[0]:=========================================
System.Object key = 00000001045b0b68 Lists/Tasks
System.Object val = 00000001045b0c78
[1]:=========================================
System.Object key = 00000001045abe10 IWConvertedForms
System.Object val = 00000001045abf30
[2]:=========================================
System.Object key = 00000001045acf68 Lists/Links
System.Object val = 00000001045ad078
[3]:=========================================
System.Object key = 00000001045ac9e0 FormServerTemplates
System.Object val = 00000001045acb00
[4]:=========================================
System.Object key = 00000001045ae330 Lists/Reporting Metadata
System.Object val = 00000001045ae460
[5]:=========================================
```

14. Use this command to list a Dictionary<string,string> at 00000001015F9730:

```
↪ !wdict 00000001015F9730
```

```
0:040> !wdict 00000001015F9730
Items   : 7
[0]:==============================================(Physical Index: 4)
System.__Canon key = 0000000180c0f988 StartRowIndex
System.__Canon value = 00000000ffdf1f18 0
[1]:==============================================(Physical Index: 3)
System.__Canon key = 0000000180c0f950 prevpagedata
System.__Canon value = 00000000ffdf1f18 0
[2]:==============================================(Physical Index: 5)
System.__Canon key = 00000001c071b070 dvt_RowCount
System.__Canon value = 00000000ffdf1f18 0
[3]:==============================================(Physical Index: 2)
System.__Canon key = 0000000180c0f7a0 nextpagedata
System.__Canon value = 00000000ffdf1f18 0
[4]:==============================================(Physical Index: 0)
System.__Canon key = 0000000180c0fe20 ListID
System.__Canon value = 0000000140b1a0f8 {A46EC59A-0F92-40EB-929F-AC4573C1DFE8}
[5]:==============================================(Physical Index: 6)
System.__Canon key = 000000018066cf78 RootFolder
System.__Canon value = 0000000000000000
[6]:==============================================(Physical Index: 1)
System.__Canon key = 0000000180c0f920 MaximumRows
System.__Canon value = 0000000140b565d0 30
```

15. The application/web configuration file has a short time alive in memory, but sometimes it is possible to retrieve a good deal of config information via the command below:

```
↪ !wconfig
↪ <--
```

```
↪   Key: system.web/customErrors
↪   Definition Config Path: machine/webroot/1067026433/_vti_bin
↪   Filename: C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\isapi\web.config
↪   Line: 0n12
↪    -->
↪
↪   <customErrors mode="On"/>
↪   <--
↪   Key: system.web/authorization
↪   Definition Config Path: machine/webroot/1067026433/_vti_bin
↪   Filename: C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\isapi\web.config
↪   Line: 0n16
↪    -->
↪
↪   <authorization>
↪                   <allow users="*"/>
↪               </authorization>
↪   (...)
```

16. OPTIONAL: Run this command to achieve similar result:

```
↪ !wfrom -type System.Configuration.SectionXmlInfo -nospace -nofield
  where (_rawXml) select "<--\nKey: ", _configKey,"\nDefinition
  Config Path: ",_definitionConfigPath,"\nFilename: ", _filename,
  "\nLine: ",_lineNumber,"\n -->\n","\n",_rawXml
```

Task 2 – Using !wservice special purpose commands

This task will show how to list the services in memory.

Task instructions

1. Open WinDBG 64-bits
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file "**w3wp.exe__DefaultAppPool__PID__11928__Date__06_18_2014__Time_07_00_26PM__486__Manual Dump.dmp"** and click **Open**.
4. Let's load netext (this dump is from a .NET 4.5 process)

```
↪ .load netext
```

5. Use the following command to show all WCF service objects in memory (click on !windex link if requested):

```
↪ !wservice
↪   Address              State        EndPoints BaseAddresses  Behaviors Throttled   Calls/Max   Sessions/Max
    ConfigName,.NET Type
↪   0000002be2021048     Opened              0n2                0n2                 0n11      True      0n0/0n128
     0n1/0n800 "WcfWifADFS051514.Service1",WcfWifADFS051514.Service1
↪
↪   1 ServiceHost object(s) found
```

6. Click on the only service listed, this is equivalent to typing:

```
↪ !wservice 0000002BE2021048
↪   Service Info
```

```
↪   ================================
↪   Address            : 0000002BE2021048
↪   Configuration Name : WcfWifADFS051514.Service1
↪   State              : Opened
↪   EndPoints          : 0n2
↪   Base Addresses     : 0n2
↪   Behaviors          : 0n11
↪   Runtime Type       : WcfWifADFS051514.Service1
↪   Is Throttled?      : True
↪   Calls/Max Calls    : 0n0/0n128
↪   Sessions/Max       : 0n1/0n800
↪   Events Raised      : No Event raised
↪   Handles Called     : OnOpeningHandle OnOpenedHandle
↪   Session Mode       : False
↪   Extensions         : 0000002be207c4b0
↪
↪   Service Behaviors
↪   ================================
↪   Concurrency Mode   : Multiple
↪   Instance Mode      : PerCall
↪   Add Error in Faults: false
↪   Max Items Obj Graph: 0n2147483647
↪   Isolation Level    : Unspecified
↪   Session Shutdown   : Automatic
↪   ASP.NET Compatib   : Allowed
↪   Http Get Enabled   : true
↪   Https Get Enabled  : true
↪   Mex Enabled        : true
↪   All Service Behav  : 0000002be207c540
↪
↪   Service Base Addresses
↪   ================================
↪   http://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪   https://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪
↪   Channels
↪   ================================
↪   Address            : 0000002BE20A69F8
↪   Listener URI       : http://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪   Binding Name       : http://tempuri.org/:BasicHttpBinding
↪   Aborted            : No
↪   State              : Opened
↪   Transaction Type   : No transaction
↪   Listener State     : Opened
↪   Timeout settings   : Open [00:01:00] Close [00:01:00] Receive: [00:10:00] Send: [00:01:00]
↪   Server Capabilities: SupportsServerAuth [No ] SupportsClientAuth [No ] SupportsClientWinIdent [No ]
↪   Request Prot Level : None
↪   Response Prot Level: None
↪   Events Raised      : No Event raised
↪   Handles Called     : OnOpeningHandle OnOpenedHandle
↪   Session Mode       : False
↪
↪   Address            : 0000002BE20B16E8
↪   Listener URI       : #INVALID#
↪   Binding Name       : http://tempuri.org/:WS2007FederationHttpBinding
↪   Aborted            : No
↪   State              : Opened
↪   Transaction Type   : No transaction
↪   Listener State     : Opened
↪   Timeout settings   : Open [00:01:00] Close [00:01:00] Receive: [00:10:00] Send: [00:01:00]
↪   Server Capabilities: SupportsServerAuth [Yes] SupportsClientAuth [Yes] SupportsClientWinIdent [No ]
↪   Request Prot Level : EncryptAndSign
↪   Response Prot Level: EncryptAndSign
↪   Events Raised      : No Event raised
↪   Handles Called     : OnOpeningHandle OnOpenedHandle
↪   Session Mode       : True
↪
↪   Address            : 0000002BE20B3AC0
↪   Listener URI       : http://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
```

```
↪   Binding Name       : ServiceMetadataBehaviorHttpGetBinding
↪   Aborted            : No
↪   State              : Opened
↪   Transaction Type   : No transaction
↪   Listener State     : Opened
↪   Timeout settings   : Open [00:01:00] Close [00:01:00] Receive: [00:10:00] Send: [00:01:00]
↪   Server Capabilities: SupportsServerAuth [No ] SupportsClientAuth [No ] SupportsClientWinIdent [No ]
↪   Request Prot Level : None
↪   Response Prot Level: None
↪   Events Raised      : No Event raised
↪   Handles Called     : OnOpeningHandle OnOpenedHandle
↪   Session Mode       : False
↪
↪   Address            : 0000002BE20B5460
↪   Listener URI       : https://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪   Binding Name       : ServiceMetadataBehaviorHttpGetBinding
↪   Aborted            : No
↪   State              : Opened
↪   Transaction Type   : No transaction
↪   Listener State     : Opened
↪   Timeout settings   : Open [00:01:00] Close [00:01:00] Receive: [00:10:00] Send: [00:01:00]
↪   Server Capabilities: SupportsServerAuth [Yes] SupportsClientAuth [No ] SupportsClientWinIdent [No ]
↪   Request Prot Level : EncryptAndSign
↪   Response Prot Level: EncryptAndSign
↪   Events Raised      : No Event raised
↪   Handles Called     : OnOpeningHandle OnOpenedHandle
↪   Session Mode       : False
↪
↪
↪   Endpoints
↪
↪   ==============================
↪
↪   Address            : 0000002BE208B068
↪   URI                : http://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪   Is Anonymous       : False
↪   Configuration Name : WcfWifADFS051514.IService1
↪   Type Name          : WcfWifADFS051514.IService1
↪   Listening Mode     : Explicit
↪   Class Definition   : 00007ff9b273d110 WcfWifADFS051514.IService1
↪   Behaviors          : 0000002be20a03e0
↪   Binding            : 0000002be2089478
↪
↪   Address            : 0000002BE2093FE0
↪   URI                : https://nape-msft.northamerica.corp.microsoft.com/Test/WcfWifADFS051514/Service1.svc
↪   Is Anonymous       : False
↪   Configuration Name : WcfWifADFS051514.IService1
↪   Type Name          : WcfWifADFS051514.IService1
↪   Listening Mode     : Explicit
↪   Class Definition   : 00007ff9b273d110 WcfWifADFS051514.IService1
↪   Behaviors          : 0000002be20a03e0
↪   Binding            : 0000002be208b390
↪
```

7. NOTE: Sometimes it is difficult to know which configuration is in place during runtime, so this command will show the actual values in place. Explaining the internals of WCF is not the scope of this lab, however this configuration is causing sessions leaks: service session mode is false, instance mode is per call (as opposed to per session) but there is a channel (not used directly) with session mode set to true. The session will be created and will only be disposed when the receive timeout is reached (10 min in this case). This is one of the main problems of performance.
8. Run this command to list the current Http running

```
↪  !whttp -order -running
```

```
0:000> !whttp -order -running
HttpContext      Start Time                 Thread Time Out Running  Status Verb     Url
0000002ae2730548 6/19/2014 12:00:22 AM          35 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
00000028e2770c08 6/19/2014 12:00:22 AM          32 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002be2836c08 6/19/2014 12:00:22 AM          63 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002c630ed350 6/19/2014 12:00:22 AM          62 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
00000029e29c69c0 6/19/2014 12:00:22 AM          70 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002b6274df58 6/19/2014 12:00:22 AM          65 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002c63336c40 6/19/2014 12:00:22 AM          31 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002962227068 6/19/2014 12:00:22 AM          61 00:00:00 00:00:09    200 GET      https://nape-msft.northamerica.
0000002be21d63b8 6/19/2014 12:00:24 AM          43 00:00:00 00:00:07    200 GET      https://nape-msft.northamerica.
0000002ae25cbeb8 6/19/2014 12:00:25 AM          45 00:00:00 00:00:06    200 GET      https://nape-msft.northamerica.

10 HttpContext object(s) found matching criteria

18 HttpContext object(s) skipped by filter
```

9. Choose the fourth line. This is equivalent to run this command:

```
↪ !whttp 0000002ae29eca50
```

10. Choose the option to dump all claims for this request, this is equivalent to:

```
↪ !wtoken 0000002ae29eca50
```

```
0:000> !wtoken 0000002ae29eca50
HttpContext     :  0000002ae29eca50 https://nape-msft.northam

0000002b62068c58 System.IdentityModel.Tokens.SessionSecurity

Session Security Token
==================================

Address      : 0000002B62068C58
Endpoint     : /Test/MvcWifADFS051514/
Id           : _a29be11c-48ae-4457-9032-b502fcd94a4b-3DAC2B0D
Current Time: 6/19/2014 12:00:32 AM
Valid From   : 6/19/2014 12:00:08 AM
Valid To     : 6/19/2014 1:00:07 AM
Status       : Valid


Authentication Type: Federation
Name Claim Type     : http://schemas.xmlsoap.org/ws/2005/05/i
Role Claim Type     : http://schemas.microsoft.com/ws/2008/06
Bootstrap Token     : 0000002b62068aa8

Claims
==================================
Type            : http://schemas.xmlsoap.org/ws/2005/05/ident
Issuer          : http://WIN-Lab1.northamerica.corp.microsoft
Original Issuer: http://WIN-Lab1.northamerica.corp.microsoft
Value           : Rodney Viana
============================================================
Type            : http://schemas.xmlsoap.org/ws/2005/05/ident
Issuer          : http://WIN-Lab1.northamerica.corp.microsoft
Original Issuer: http://WIN-Lab1.northamerica.corp.microsoft
Value           : rviana@microsoft.com
============================================================
Type            : http://schemas.microsoft.com/ws/2008/06/ide
Issuer          : http://WIN-Lab1.northamerica.corp.microsoft
Original Issuer: http://WIN-Lab1.northamerica.corp.microsoft
Value           : ESCAL ENG
============================================================
Type            : http://schemas.microsoft.com/ws/2008/06/ide
```

11. In this case, the provider is ADFS, when the provider is ACS things will be similar only changing the issuer (will look like `https://contososso.accesscontrol.windows.net/)` and original issuer (it will depend on the original provider – Live, Facebook, ADFS)

12. This works wonderfully in SharePoint when claims authentication is enabled.
13. The Federation authentication normally survives in the federation cookie. You can see the calls sharing a same federation token by using the command !wcookie
14. Use the command below to list all federation cookies that repeats at least 3 times

```
↪ !wcookie -summary -name FedAuth* -min 3
```

```
0:000> !wcookie -summary -name FedAuth* -min 3
Action Total Finished Cookie=Value
=========================================================================
(list)     3         2 FedAuth1=OEhPdXpKZHJpeGFEeVhac0ZaT0RxOWtpcmw5S
(list)     3         2 FedAuth1=OUQzSzlVNHhWZlB2OHZIMEthdlRBNjgyWFpNW
(list)     3         2 FedAuth2=REM3OFFxVmZKZnRKdGJaT1ZycmpIMXdWdW9QR
(list)     3         2 FedAuth2=UmlETmtObmVKWVdobjZHcHFuUGxZQS9GNi82a
(list)    21        16 FedAuth3=
(list)     7         5 FedAuth3==
(list)     3         2 FedAuth3=TzZLSlpEa3dSamI4Q21tWnh2b0dBN0xpSzNicr
(list)     3         2 FedAuth=77u/PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGl
(list)     3         2 FedAuth=77u/PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGl
```

15. Click on the first item in the list to dump all http request related to that cookie

Task 3 – OPTIONAL: Playing with much complex queries

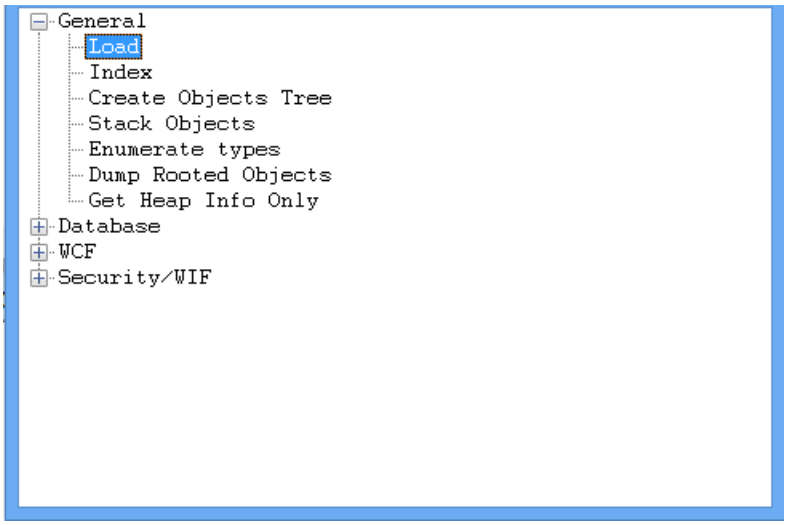
This is a bonus task to show some queries prepared while debugging real life problems. The objective is not to understand all the queries, but rather to see what can be achieved by the tool.

Task instructions

1. Open WinDBG 64-bits
2. After WinDBG opens, select **File | Open Crash Dump…**.
3. Choose file "**TokenService.dmp"** and click **Open**.
4. Instead of loading wcfwifext, let's first load a command tree:

```
↪ .cmdtree netext.tl
```

5. Expand General and double-click on load:



NOTE: This will choose the correct extension version to load without any intervention on user's part.

6. Feel free to open other dump files and explore the available "recipe" queries that are more related to your field
7. Study the queries and try to do changes