

Debugging a Bugcheck 0xF4

Ryan Mangipano

My name is Ryan Mangipano (ryanman) and I am a Sr. Support Escalation Engineer at Microsoft. Today's blog will be a quick walkthrough of the analysis of a bugcheck 0xF4 and how I determined that the action plan going forward should consist of enabling pool tagging on this system.

I began my review with !analyze -v. From the output I can see that a process required for the system to function properly unexpectedly exited or was terminated. The goal of this debugging session will be to determine what failed and why.

```
0: kd> !analyze -v
```

```
*****
*
*                               Bugcheck Analysis                               *
*
*****
```

CRITICAL_OBJECT_TERMINATION (f4)

A process or thread crucial to system operation has unexpectedly exited or been terminated.

Several processes and threads are necessary for the operation of the system; when they are terminated (for any reason), the system can no longer function.

Arguments:

```
Arg1: 00000003, Process                A value of 0x3 in this
parameter indicated that it was a process that terminated, not a thread
Arg2: 8a03ada0, Terminating object    This value is a pointer to the _EPROCESS
object that terminated
Arg3: 8a03af14, Process image file name Process Name
Arg4: 805d1204, Explanatory message (ascii) text message about the problem
```

We shall begin by dumping out all the parameters of the bugcheck. Let's dump out the "Terminating Object" below

```
0: kd> !object 8a03ada0
Object: 8a03ada0 Type: (8a490900) Process
ObjectHeader: 8a03ad88 (old version)
HandleCount: 3 PointerCount: 228
```

First, let's dump out the process image file name from the bugcheck parameter 3 above.

```
0: kd> dc 8a03af14
8a03af14 73727363 78652e73 00000065 00000000 csrss.exe
```

```
0: kd> dt _EPROCESS 8a03ada0 imageFileName
CSRSRV!_EPROCESS
+0x174 ImageFileName : [16] "csrss.exe"
```

Notice that if we add the base of the _EPROCESS object (8a03ada0- Parameter 2) to the offset of the imageFileName field (+0x174) we get parameter 3. The imageFileName field.

```
0: kd> ? 8a03ada0+0x174
Evaluate expression: -1979470060 = 8a03af14
```

```
0: kd> dc 8a03af14
```

Debugging a Bugcheck 0xF4

Ryan Mangipano

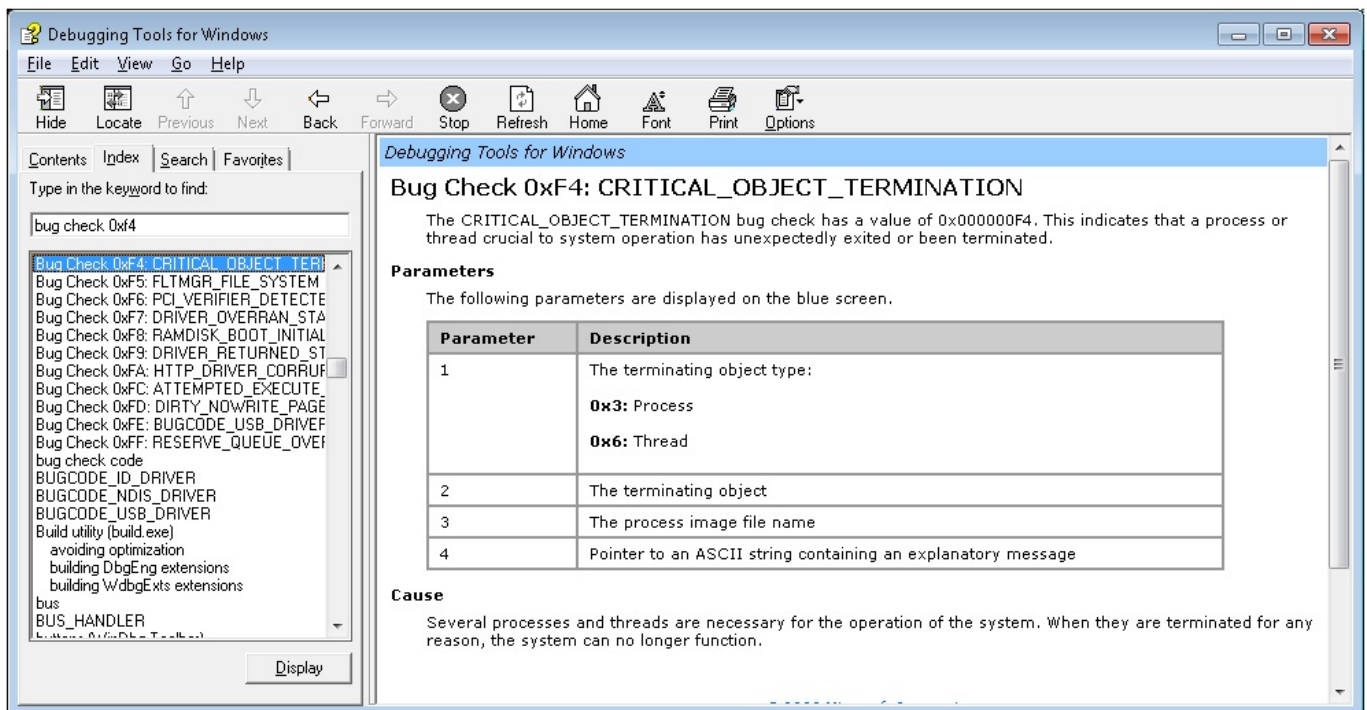
```
8a03af14 73727363 78652e73 00000065 00000000 csrss.exe.....
```

Let's dump out the ASCII message from parameter number 4

```
0: kd> dc 805d1204
805d1204 6d726554 74616e69 20676e69 74697263 Terminating crit
805d1214 6c616369 6f727020 73736563 25783020 ical process
```

Let's review the debugger help file for more information. We can see that this bugcheck occurs when a critical process or thread terminates. "Several processes and threads are necessary for the operation of the system. When they are terminated for any reason, the system can no longer function.

```
0: kd> .hh bug check 0xf4
```



Next, we need to determine why this process terminated. !analyze -v also provided us with an exception record which provides us with an error code:

```
PROCESS_NAME: csrss.exe

EXCEPTION_RECORD: 9a85e9d8 -- (.exr 0xffffffff9a85e9d8)
ExceptionAddress: 7c92c375 (ntdll!RtlFindMessage+0x0000007c)
ExceptionCode: c000006 (In-page I/O error)
ExceptionFlags: 00000000
NumberParameters: 3
Parameter[0]: 00000000
Parameter[1]: 7c99c3d8
Parameter[2]: c000009a
Inpage operation failed at 7c99c3d8, due to I/O error c000009a
```

Debugging a Bugcheck 0xF4

Ryan Mangipano

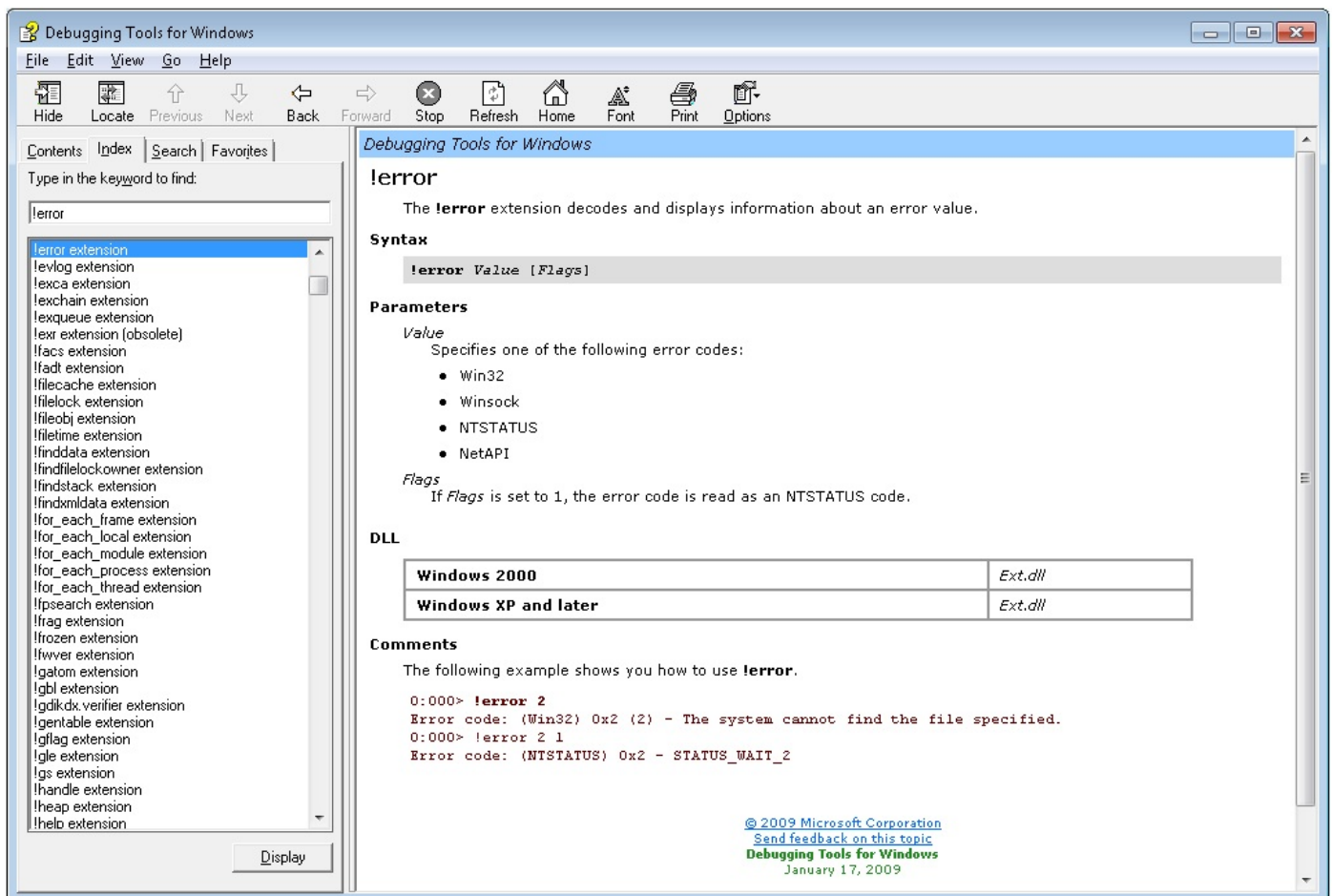
EXCEPTION_CODE: (NTSTATUS) 0xc0000006 - The instruction at 0x%p referenced memory at 0x%p. The required data was not placed into memory because of an I/O error status of 0%x.

Since we have an error code, let's investigate that error code. We can quickly perform this operation from within the debugger using the !error command

```
0: kd> !error c000009a
```

```
Error code: (NTSTATUS) 0xc000009a (3221225626) - Insufficient system resources exist to complete the API.
```

```
0: kd > .hh !error
```



Let's check the output of the !vm command

```
0: kd> !vm 2
```

```
*** Virtual Memory Usage ***
```

```
Physical Memory:      760875 ( 3043500 Kb)
Page File: \??\C:\pagefile.sys
Current:      4190208 Kb Free Space:  4156380 Kb
Minimum:     4190208 Kb Maximum:     4190208 Kb
Available Pages: 579241 ( 2316964 Kb)
ResAvail Pages: 673481 ( 2693924 Kb)
```

Debugging a Bugcheck 0xF4

Ryan Mangipano

```
Locked IO Pages:          69 (          276 Kb)
Free System PTEs:        115226 (      460904 Kb)
Free NP PTEs:             0 (           0 Kb)
Free Special NP:         0 (           0 Kb)
Modified Pages:          221 (          884 Kb)
Modified PF Pages:       219 (          876 Kb)
NonPagedPool Usage:      65534 (      262136 Kb)
NonPagedPool Max:        65536 (      262144 Kb)
***** Excessive NonPaged Pool Usage *****
PagedPool 0 Usage:       24167 (      96668 Kb)
PagedPool 1 Usage:        967 (        3868 Kb)
PagedPool 2 Usage:        967 (        3868 Kb)
PagedPool 3 Usage:        984 (        3936 Kb)
PagedPool 4 Usage:        977 (        3908 Kb)
PagedPool Usage:         28062 (     112248 Kb)
PagedPool Maximum:       92160 (     368640 Kb)

***** 2075 pool allocations have failed *****

Session Commit:          1562 (        6248 Kb)
Shared Commit:           2526 (     10104 Kb)
Special Pool:             0 (           0 Kb)
Shared Process:          4821 (     19284 Kb)
PagedPool Commit:        28062 (     112248 Kb)
Driver Commit:           5138 (     20552 Kb)
Committed pages:         153449 (     613796 Kb)
Commit limit:            1767229 (    7068916 Kb)
```

```
0: kd> !poolused
unable to get PoolTrackTable - pool tagging is disabled, enable it to use this
command
Use gflags.exe and check the box that says "Enable pool tagging".
```

The output above has informed us that pool tagging is disabled. Let's demonstrate how you can verify that it is disabled:

```
0: kd> dd nt!NtGlobalFlag L1
805597ec  00000000

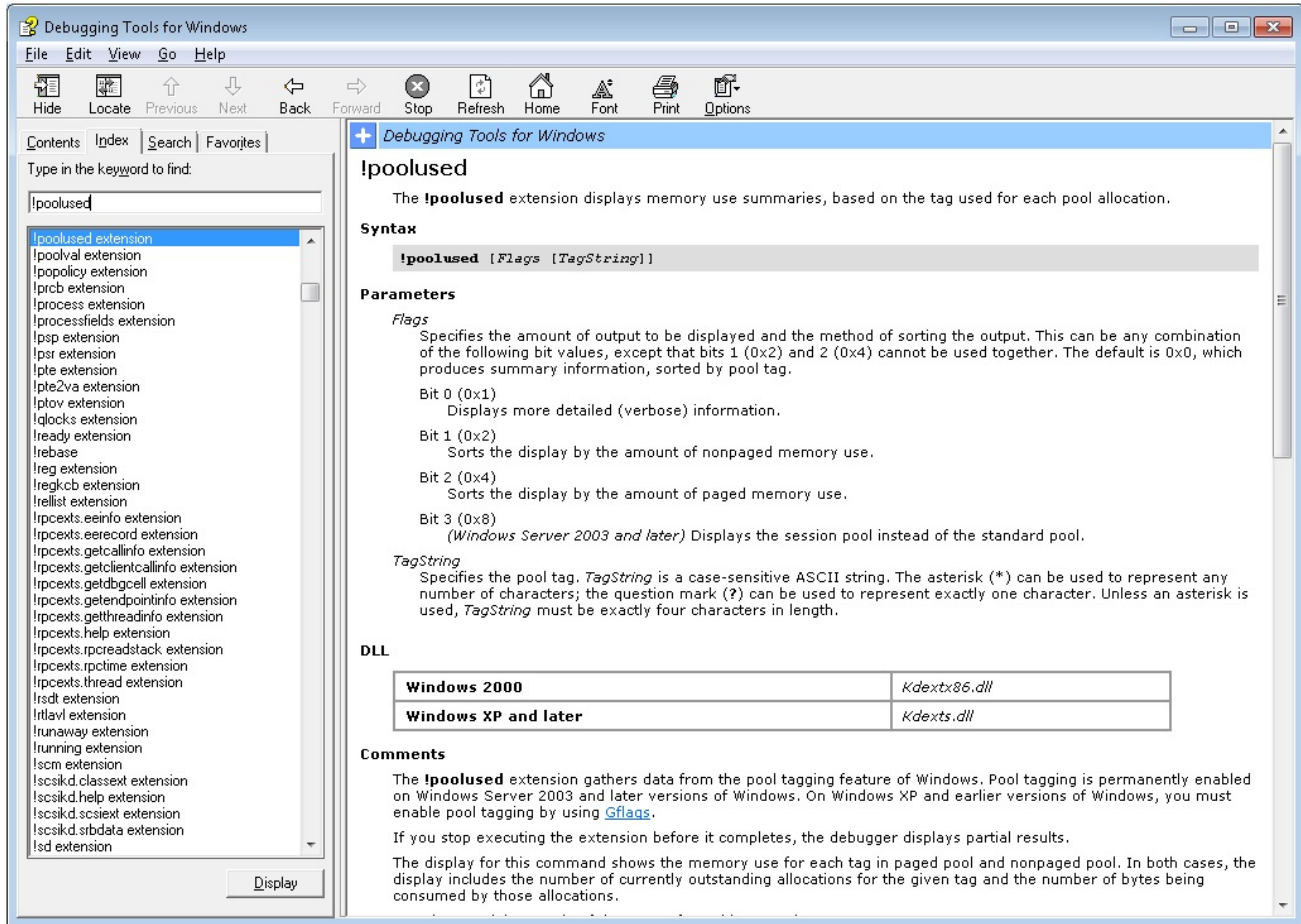
0: kd> !gflag
Current NtGlobalFlag contents: 0x00000000
```

Let's explore the debugging help file entry on the !poolused command

```
0: kd > .hh !poolused
```

Debugging a Bugcheck 0xF4

Ryan Mangipano



Reading the text above, we are informed that “Pool tagging is permanently enabled on Windows Server 2003 and later versions of Windows. On Windows XP and earlier versions of Windows, you must enable pool tagging by using Gflags.”

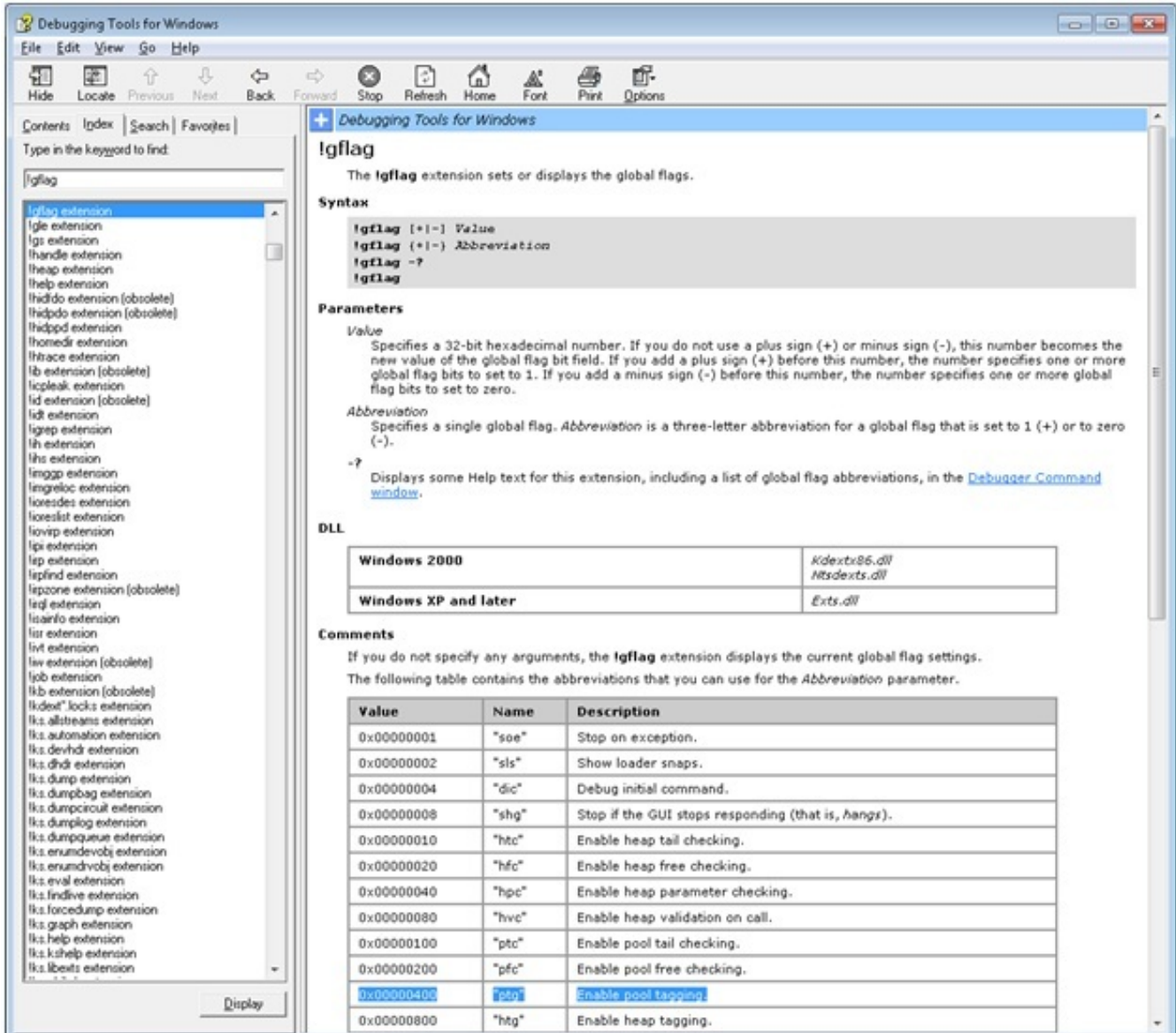
Using the vertarget command, I can see that this system was running Windows XP.

```
0: kd> vertarget
Windows XP Kernel Version 2600 (Service Pack 2) MP (2 procs) Free x86
compatible
```

```
0: kd > .hh !gflag
```

Debugging a Bugcheck 0xF4

Ryan Mangipano



By reviewing the help file entry for the !gflag extension, I was able to determine that if pooltagging was set, the following bit would have been set:

0x400 "ptg" Enable pool tagging.

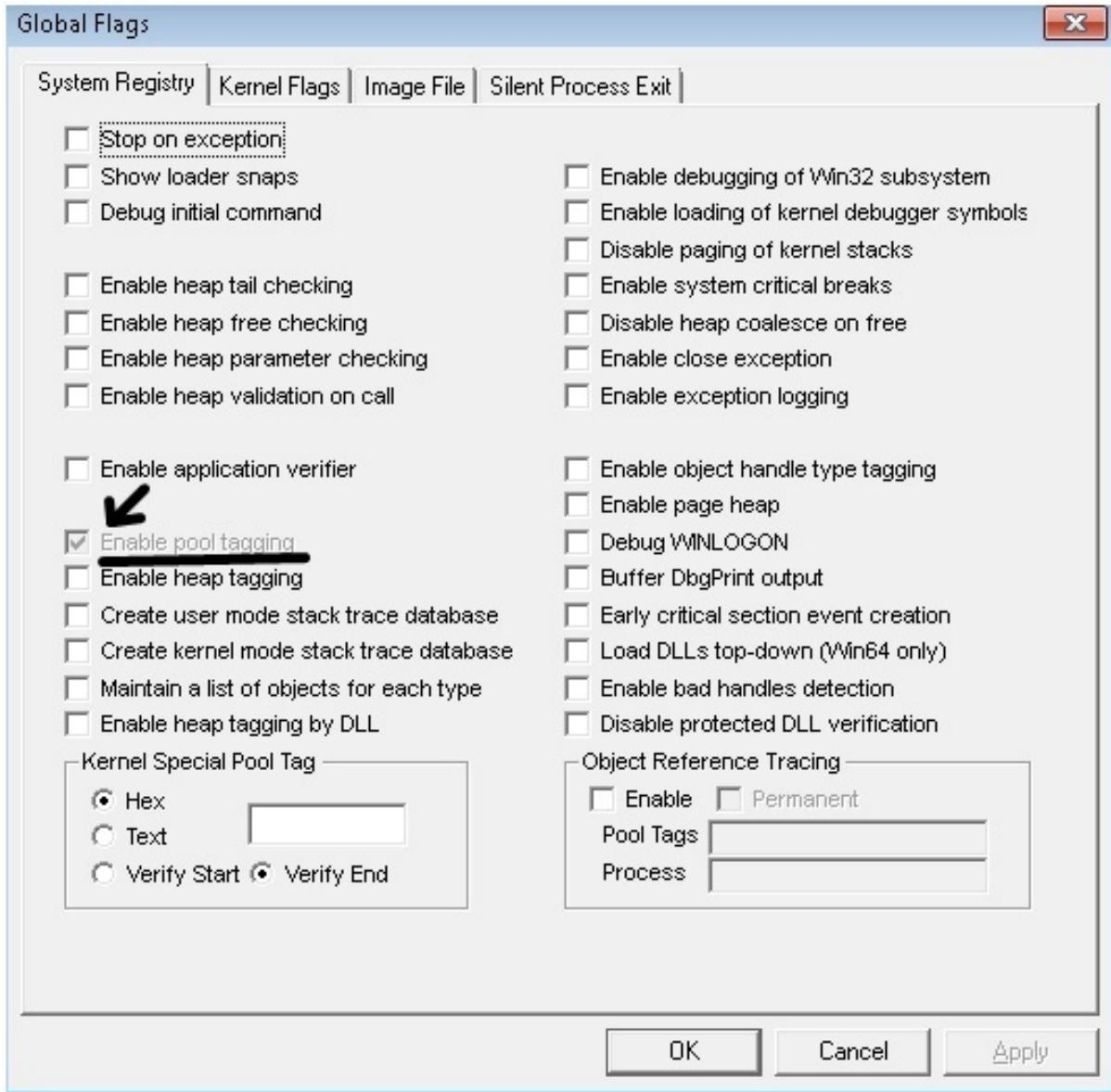
```
0: kd> .formats 0x400
Evaluate expression:
```

```
...
Binary: 00000000 00000000 00000100 00000000                    0x00000400
```

Gflags is included in the Debugging Tools for Windows. The screenshot below is from a Windows 7 system. Notice that Pool Tagging is enabled permanently as described above.

Debugging a Bugcheck 0xF4

Ryan Mangipano



Summary: This system bugchecked when the critical process csrss.exe failed an I/O operation due to insufficient non-paged pool. For an action plan, we recommended the use of gflags to enable pool tagging in order to obtain more information about pool consumption.