How to Identify a Cisco CallManager Restart as a Crash or Service Shutdown

Document ID: 46806

Contents

Introduction

This document describes the difference between a Cisco CallManager crash and a service shutdown. The document also provides the procedure to report a Cisco CallManager crash and enable Cisco Technical Support to troubleshoot the issue.

Prerequisites

Requirements

There are no specific requirements for this document.

Components Used

The information in this document is based on these software versions:

- Cisco CallManager 3.x and 4.0

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

Conventions

Refer to the Cisco Technical Tips Conventions for more information on document conventions.

Difference Between Cisco CallManager Crashes and Shutdowns

Crashes

A bug in the Cisco CallManager code causes CallManager crashes. There are three main types of crashes:

- Access violations
- Divide by zero
- Unknown exceptions

Crashes generate Dr. Watson entries, which are appended to the end of the existing Dr. Watson file. Crashes also generate user.dmp files. The location of these files is C:\Documents and Settings\All Users\Documents\DrWatson.

The name of the Dr. Watson file, which is a text file, is drwtsn32.log.

Choose **drwtsn32** from the Run window to configure the settings.

How to Read a Dr. Watson File

Complete these steps to read the Dr. Watson file:

1. Search for the word when , in lowercase, and find the date and time at which the problem occurred.

   The Dr. Watson file records all application crashes. Some crash records may not be Cisco CallManager crashes. Examples of crash records that are not Cisco CallManager crashes include RisDC.exe and aupair.exe.
2. After you locate the date and time of the problem, locate the process identifier (PID) number, and search the task list to determine which application crashed.

   The task list appears in the example in this step.

   In this example, the application that crashed has a PID of 752, and the name of the application is SCAN32.exe:

```
Application exception occurred:
        App:  (pid=752)
        When: 9/1/2000 @ 10:23:40.836
        Exception number: c0000005 (access violation)

*----> System Information <----*
        Computer Name: CISCO-8VCUWBLUJ
        User Name: SYSTEM
        Number of Processors: 1
        Processor Type: x86 Family 6 Model 8 Stepping 3
        Windows 2000 Version: 5.0
        Current Build: 2195
        Service Pack: None
        Current Type: Uniprocessor Free
        Registered Organization: Cisco Systems Inc.
        Registered Owner: Cisco User

*----> Task List <----*
   0 Idle.exe
   8 System.exe
 144 smss.exe
 168 csrss.exe
 164 winlogon.exe
 216 services.exe
 228 lsass.exe
 336 ibmpmsvc.exe
 380 svchost.exe
 424 svchost.exe
 576 regsvc.exe
 592 MSTask.exe
 924 Explorer.exe
 992 cmd.exe
 972 msiexec.exe
 928 tp4mon.exe
 856 ibmpmsvc.exe
 852 ltmsg.exe
 408 RunDll32.exe
 428 RunDll32.exe
 328 PDirect.exe
 620 TP98.exe
 968 tphkmgr.exe
 948 PRPCUI.exe
 668 AUTOCHK.exe
 744 tponscr.exe
 868 KIX32.exe
 520 spoolsv.exe
1164 Avsynmgr.exe
1136 VsStat.exe
1192 Vshwin32.exe
1224 Mcshield.exe
1024 MCUPDATE.exe
752 SCAN32.exe
1292 drwtsn32.exe
   0 _Total.exe
```

3. If the crash is a Cisco CallManager crash, note the exception number to determine the type of crash.

   Note: Route to the appropriate development team an application crash that is not a Cisco CallManager crash, if necessary.

```
Application exception occurred:

        App:  (pid=752)

        When: 9/1/2000 @ 10:23:40.836

        Exception number: c0000005 (access violation)
```

   In this example, the exception number is `c0000005`, which is an `access violation`. This `access violation` means that the application attempted to access memory outside of the application memory limit that the operating system set.

   Another possible crash type is divide by zero. As this example shows, the exception number for divide by zero is `c0000094`:

```
Application exception occurred:

        App:  (pid=1564)

        When: 1/7/2003 @ 13:16:15.051

        Exception number: c0000094 (divide by zero)
```

   The crash type can also be unknown exception. As this example shows, the exception number for unknown exception is `e06d7363`:

```
Application exception occurred:

        App:  (pid=2784)
```

```
When: 12/10/2002 @ 09:02:58.202

Exception number: e06d7363
```

When you determine if a crash is an access violation, divide by zero, or unknown exception, you can match a crash with an existing Cisco bug. If you find no matches, the development engineer has a good start to determine what occurred.

4. Search under the `when` section of the file for the word `FAULT` to begin to define the "signature" of the crash.

Note: `FAULT` appears in capital letters.

This `FAULT` section of the file contains six pieces of important information, which are:

- The number of the thread that experienced the problem
- The contents of the registers for this thread at the time of the crash
- The function that executed at the time of the crash
- The assembly code statement that resulted in the crash
- The stack back trace that shows the addresses of the functions that executed, in order, just before the crash
- The raw stack dump, which provides more information about what was on the run−time stack at the time of the crash

This code provides an example of a Cisco CallManager crash that is an access violation crash. Boldface text highlights the six critical elements, as well as the word `FAULT`, which marks this section of the file:

```
State Dump for Thread Id 0x998

!−−− This number is the number of the thread that experienced the problem.


eax=00cae82c ebx=02070000 ecx=00e95da0 edx=346984d8 esi=34698970 edi=346984d8
eip=77fcb9b3 esp=05cef34c ebp=05cef358 iopl=0         nv up ei ng nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000283

!−−− This provides the contents of the registers at the time of the crash.

function: RtlSizeHeap

!−−− This function executed at the time of the crash.

        77fcb992 0f87aafeffff     jnbe    RtlFreeHeap+0x20f (77fcb842)
        77fcb998 807d1400         cmp     byte ptr [ebp+0x14],0x0      ss:
         0650c92a=??
        77fcb99c 0f8586300000     jne     RtlZeroHeap+0x327 (77fcea28)
        77fcb9a2 57               push    edi
        77fcb9a3 53               push    ebx
        77fcb9a4 e8646cfbff       call RtlConsoleMultiByteToUnicodeN+0x348
         (77f8260d)
        77fcb9a9 8b4f0c           mov     ecx,[edi+0xc]          ds:
         34eb5aaa=00003781
        77fcb9ac 8b4708           mov     eax,[edi+0x8]          ds:
         34eb5aaa=00003781
        77fcb9af 3bc1             cmp     eax,ecx
        77fcb9b1 8901             mov     [ecx],eax              ds:
         00e95da0=00cae82c
FAULT −>77fcb9b3 894804           mov     [eax+0x4],ecx          ds:
 014cbdfe=ec810000


!−−− This is the assembly code statement that resulted in the crash.
```

```
        77fcb9b6 744d             jz      77fd4405
        77fcb9b8 8a4705           mov     al,[edi+0x5]                ds:
         34eb5aaa=81
        77fcb9bb a804             test    al,0x4
        77fcb9bd 0f8521310000     jne     RtlZeroHeap+0x3e3 (77fceae4)
        77fcb9c3 8a4605           mov     al,[esi+0x5]                ds:
         34eb5f42=d5
        77fcb9c6 2410             and     al,0x10
        77fcb9c8 a810             test    al,0x10
        77fcb9ca 884705           mov     [edi+0x5],al                ds:
         34eb5aaa=81
        77fcb9cd 0f8555030000     jne     RtlSizeHeap+0x3ef (77fcbd28)
        77fcb9d3 0fb70f           movzx   ecx,word ptr [edi]        ds:
         346984d8=0093
        77fcb9d6 8b4510           mov     eax,[ebp+0x10]        ss:
         0650c92a=????????

*----> Stack Back Trace <----*

!--- This shows, in order, the addresses of the functions that executed
!--- just before the crash.

FramePtr ReturnAd Param#1  Param#2  Param#3  Param#4  Function Name
05CEF358 77FCB733 02070000 34698970 05CEF3D0 00000000 ntdll!RtlSizeHeap
05CEF400 7800115C 02070000 00000000 34698978 05CEF454 ntdll!RtlFreeHeap
05CEF448 00C0304F 34698978 00545EC2 34698978 34698978 !free
05CEF460 00B66F85 00000001 00B6626C 033B3D58 025A6720 !<nosymbols>
05CEFF34 018E736B 025A6720 77E964CB 033C6B20 033C6B20 !<nosymbols>
05CEFF80 780060CE 033B3D58 77E964CB 00000018 033C6B20 !ACE_OS_Thread_Adapter::
 invoke
µµ
05CEFFEC 00000000 00000000 00000000 00000000 00000000 kernel32!TlsSetValue

*----> Raw Stack Dump <----*


!--- This provides more information about what was on the run-time stack
!--- at the time of the crash.

05cef34c  00 00 07 02 70 89 69 34 - 00 00 00 00 00 f4 ce 05  ....p.i4........
05cef35c  33 b7 fc 77 00 00 07 02 - 70 89 69 34 d0 f3 ce 05  3..w....p.i4....
05cef36c  00 00 00 00 54 f4 ce 05 - 78 89 69 34 20 67 5a 02  ....T...x.i4 gZ.
05cef37c  44 5b e3 09 94 f3 ce 05 - 30 e6 b5 00 fc f3 ce 05  D[......0.......
05cef38c  38 29 6a 09 40 5b e3 09 - a8 f3 ce 05 65 e5 b5 00  8)j.@[......e...
05cef39c  fc f3 ce 05 38 29 6a 09 - 40 5b e3 09 c4 f3 ce 05  ....8)j.@[......
05cef3ac  39 e2 b5 00 57 92 89 01 - 30 db 55 02 f5 50 5b 00  9...W...0.U..P[.
05cef3bc  e0 f3 ce 05 cc f3 ce 05 - 0f 4f 5b 00 e0 f3 ce 05  .........O[.....
05cef3cc  00 00 07 02 19 00 00 00 - 01 f4 ce 05 f8 2b cf 21  .............+.!
05cef3dc  f8 2b cf 21 01 f1 ce 05 - 28 ff ce 05 70 f3 ce 05  .+.!....(...p...
05cef3ec  98 ef ce 05 38 f4 ce 05 - a7 9d fb 77 90 26 f8 77  ....8......w.&.w
05cef3fc  01 00 00 00 48 f4 ce 05 - 5c 11 00 78 00 00 07 02  ....H...\..x....
05cef40c  00 00 00 00 78 89 69 34 - 54 f4 ce 05 04 fa ce 05  ....x.i4T.......
05cef41c  20 67 5a 02 02 00 00 00 - 64 00 00 00 5c 00 00 00   gZ.....d...\...
05cef42c  fe 08 00 00 00 00 00 00 - 98 ef ce 05 28 ff ce 05  ............(...
05cef43c  b8 ff 00 78 50 32 03 78 - ff ff ff ff 60 f4 ce 05  ...xP2.x....`...
05cef44c  4f 30 c0 00 78 89 69 34 - c2 5e 54 00 78 89 69 34  O0..x.i4.^T.x.i4
05cef45c  78 89 69 34 34 ff ce 05 - 85 6f b6 00 01 00 00 00  x.i44....o......
05cef46c  6c 62 b6 00 58 3d 3b 03 - 20 67 5a 02 98 f6 e6 36  lb..X=;. gZ....6
05cef47c  98 f6 e6 36 00 00 00 00 - 00 00 00 00 00 00 00 00  ...6............
```

These six bits of information constitute part of, but not all of, the crash signature. The rest of the information that defines the signature is:

- The type of crash (access violation, divide by zero, or unknown exception)
- The last Signal Distribution Layer (SDL) trace statements that executed before the crash

  Note: The last SDL file that had use before the crash, as well as the Dr. Watson file, attaches to any crash bug.

This signature information (the last SDL file, the last Cisco CallManager file, and the Dr. Watson file) attaches to the Distributed Defect Tracking System (DDTS) record when you create a new crash DDTS.

If you match the new crash with a DDTS that already exists and has the same root cause, this information is the same:

- The type of exception
- The name of the function that executed at the time of the crash
- The names of the functions in the stack back trace

  Note: These names do not always appear in a Dr. Watson file.
- The assembly code statement that appears next to the `FAULT` marker
- The last SDL trace lines, which should be very similar

The contents of the registers, the memory addresses, and other information can differ from the information in another DDTS that exists, even if the crash has the same root cause. The addresses vary if you run a different version of Cisco CallManager. If you run the same version of Cisco CallManager, the addresses in the assembly code section and in the stack trace section are the same.

5. Collect these files to debug the crash:

   - drwtsn32.log
   - user.dmp
   - The last SDL and Cisco CallManager trace files, from approximately 5 minutes before the crash and 5 minutes after the restart.
   - proglog files

     Note: Collect these files only in Cisco CallManager versions 3.2 and later.
   - Event logs, both System and Application, if available.
   - Performance Monitor (perfmon) logs, if available.

DBLException Error

You see this error message in the application log of both the Cisco CallManager Publisher and Subscriber:

```
Error: DBLException - DBL Exception.
  ErrorCode: 8
  ExceptionString: Invalid parameter
  UNKNOWN_PARAMNAME:Text: addDevice
  App ID: Cisco CallManager
  Cluster ID: XXXX-Cluster
  Node ID: 192.168.0.2
Explanation: Severe database layer interface error occurred.
Recommended Action: Contact TAC..
```

Or:

```
A COM error occurred during processing. (6)

Details:
Error No. -2147219962 (0x80040606):
CDBLException Dump: [COM Error] COM Error Description = []
```

This type of error occurs when an IP phone is rejected from registration or because of a broken subscription between the publisher and subscriber databases. This issue can be solved by using the DBLHelper tool. For more information on DBLHelper, refer to Using DBLHelper to Reestablish a Broken Cisco CallManager Cluster SQL Subscription.

This error can also occur due to the service crash of the Cisco Database Layer Monitor. Perform these steps to resolve the issue:

1. Go to **Start** > **Programs** > **Administrative Tools** > **Component Services**.
2. Expand **Component Services** > **Computers** > **My Computer** > **COM+ Applications**.
3. Start the **MSDTC** (Distributed Transaction Coordinator) service, if it shows stopped.

Shutdowns

The other type of Cisco CallManager restart is a shutdown. A shutdown is when Cisco CallManager is unable to operate effectively and shuts itself down. Shutdowns fall into two categories:

- Initialization Timeouts
- SDL Timer and SDL Router Thread Deaths

If Cisco CallManager shuts itself down, you find a shutdown `Reason code` in the last few trace lines of the CallManager trace. Here is an example:

```
03/22/2003 14:32:16.562 Cisco CallManager|CallManagerFailure - Indicates
some failure in the Cisco CallManager system. Host name of hosting
node.:NEROCM1 IP address of hosting node.:172.27.27.224 Reason code.:4
Additional Text [Optional]: App ID:Cisco CallManager Cluster
ID:NEROCM1-Cluster Node
ID:172.27.27.224|<CLID::NEROCM1-Cluster><NID::172.27.27.224><CT::Alarm>
```

In this example, the reason code is 4. This list provides shutdown reason codes from the Cisco CallManager code:

```
class CallManagerFailureAlarm : public CallManagerAlarmCatalog {
public:
            enum Reason {
                        Unknown = 1,
                        HeartBeatStopped = 2,
                        RouterThreadDied =3 ,
                        TimerThreadDied = 4,
                        CriticalThreadDied = 5,
                        DeviceMgrInitFailed = 6,
                        DigitAnalysisInitFailed = 7,
                        CallControlInitFailed = 8,
                        LinkMgrInitFailed = 9,
                        DbMgrInitFailed = 10,
                        MsgTranslationInitFailed = 11,
                        SupServiceInitFailed = 12,
                        DirectoryInitFailed = 13
            };
```

Reason 1 and Reason 2 are rare cases of internal shutdowns, while the other reasons are more common. Reason 3 indicates that the SDL router thread has stopped response. Reason 4 indicates that the SDL timer thread has stopped response. Reasons 5™3 relate to initialization timer fire.

Initialization Timeouts

When the Cisco CallManager service first starts, the CallManager Process Monitor (CMProcMon) thread starts. Then, the MmmanInit thread starts, which spawns all the other processes. Next, the SDL router thread starts. This thread handles signals that send from one process to another. All three of the threads start at the same time. The CMProcMon thread and the SDL router thread are up and run while the MmmanInit thread starts up other processes.

CMProcmon and SDL need to be up and running while MmmanInit starts up various processes. The MmmanInit thread starts these processes, in this order:

1. Database (ProcessDb)

   Note: ProcessDb is a Cisco CallManager interface to Database Layer (DBL) code.

   At the same time, the MmmanInit code also starts a number of other Cisco CallManager internal, independent processes. These processes include H225Handler, MGCPBhHandler, and LineManager.
2. Regions
3. AARNeighborhood
4. Locations
5. Route Plan
6. Digit Analysis
7. Call Control
8. Supplementary Services

   Features include call park, forward, conference, and transfer.
9. Device
10. Directory
11. Calling Search Space Manager (CSSManager)
12. Time of Day Manager (TODManager)

The accomplishment of these tasks occurs in a series. Each one of the twelve tasks has a timer that associates with the task. This timer starts when the task begins. If the timer fires before the task completes, Cisco CallManager stops and prints an SDL trace that reads:

```
Critical thread death: name of the timer which fired
```

This list shows each of the timers, as well as the SDL signal that starts the timer and the SDL signal that stops the timer. The InitDone signals appear in the SDL trace if you have set trace levels appropriately. (You set SdlTraceTypeFlags at 0x8000CB15.)

These default timers are based on Cisco CallManager version 4.1(2). If you run different version, the might be slightly different.

1. Database Initialization Time (defaults to 900 seconds) − The start signal for this time is the "start" signal sent to the MmmanInit process. You see this in the SDL trace.
2. Regions Initialization Time (defaults to 120 seconds).
3. AARNeighborhoods Initialization Time (defaults to 90 seconds).
4. Locations Initialization Time (defaults to 90 seconds).
5. Route Plan Initialization Time (defaults to 600 seconds).
6. Digit Analysis Initialization Time (defaults to 900 seconds).
7. Call Control Initialization Time (defaults to 90 seconds).
8. Supplementary Services Initialization Time (defaults to 900 seconds) − The start signal is CcInitDone and end signal is SsInitDone.

9. Device Initialization Time (defaults to 360 seconds).
10. Directory Initialization Time (defaults to 90 seconds)
11. CSSManager Initialization Time (defaults to 900 seconds).
12. TODManager Initialization Time   (defaults to 900 seconds).

After all the tasks are complete, Cisco CallManager opens SDL links to the CallManager services that run on other nodes in the network. Cisco CallManager also opens SDL links to Computer Telephony Integration (CTI) Manager services that run on the same node or different nodes in the network.

Then, MmmanInit sends a CMInitComplete signal back to the CMProcMon thread. When the CMProcMon first starts, it starts a 60−minute hard−coded timer for Cisco CallManager initialization. The timer has the name CMInitComplete_WaitTime. (This timer is not a service parameter; the timer is not configurable.) If the CMProcMon thread does not receive the CMInitComplete signal within 60 minutes, Cisco CallManager stops and issues a trace statement that reads:

```
Timed out waiting for CMInitComplete signal
```

If any one of the twelve initialization tasks fails, or if the total time for these tasks exceeds 60 minutes, Cisco CallManager stops.

Note: The CMInitComplete_WaitTime timer was once hard coded to 10 minutes. This hard code has changed to 60 minutes as part of Cisco bug ID CSCdx31622 (registered customers only) . The change entered the 3.1(3) Engineering Special (ES) train, with ES 38 as the start. The change is also in the 3.2(2) ES train, with ES 11 as the start, and in Cisco CallManager 3.3.

If you experience problems with an initialization timer fire, you might only need to increase the timer setting to resolve the startup. If this change does not resolve the problem, the problem can be a slow database response time that causes the operation to time out. Collect detailed DBL traces, as well as SDL and Cisco CallManager traces, if necessary.

Collect these files to debug an initialization problem:

- Detailed Cisco CallManager trace
- SDL trace

  Note: Set the SdlTraceTypeFlags to 0x8000CB15.
- Detailed DBL trace

SDL Timer and SDL Router Thread Deaths

The SDL router thread is the most important thread of execution inside the Cisco CallManager application. It controls the send of call processing signals. The CMProcMon thread checks the health of the SDL router thread once every two seconds. The Cisco CallManager traces show this health check, as you can see in these statements:

```
02/05/2003 00:30:32.790 Cisco CallManager|CMProcMon − −−−−−Entered Router
Verification|<CLID::USNYTSVOIPPB01−Cluster><NID::10.2.40.11>

02/05/2003 00:30:32.790 Cisco CallManager|CMProcMon − −−−−Exited Router
Verification|<CLID::USNYTSVOIPPB01−Cluster><NID::10.2.40.11>
```

If the CMProcMon thread enters and exits router verification, the SDL router thread responded to the health check and is fine.

However, if the SDL router thread does not respond, you see `While loop` in the Cisco CallManager trace, as this shows:

```
CMProcMon − −−−−Entered While loop ++++ TimeAtWhileEntry: [some number here],
TimeBeforeSleep: [another number], TimeAfterSleep: [a third number], sleepTimeWas :
[4th number"
```

In this emergency situation, the SDL router thread receives checks once every second for a period of 20 seconds. If the thread responds at any time during the 20−second period, normal operation resumes, and the health of the SDL router thread once again receives verification every 2 seconds. If, however, the SDL router thread does not respond to the checks over the 20 seconds, the Cisco CallManager application shuts down. This statement appears in the SDL trace:

```
000177508| 01/12/31 07:28:40.389| 001| AlarmErr  |
 |                  |                  |                  |
| AlarmClass: CallManager, AlarmName: CallManagerFailure, AlarmSeverity: Error
AlarmMessage: , AlarmDescription: Indicates some failure in the Cisco CallManager
 system.,
AlarmParameters:  HostName:CCM−PUB, IPAddress:10.5.162.180, Reason:3, Text:,
AppID:Cisco CallManager, ClusterID:CCM−PUB−Cluster, NodeID:10.5.162.180,
```

Notice the reason code `3` in the text of this trace statement. The code means that the SDL router thread has stopped response, so Cisco CallManager has shut down.

The most likely cause of an SDL router thread shutdown is a lack of system resources. Another application has used most or all of the CPU for a long period of time, at least 20 seconds. This activity is why performance monitors are vital to debug this type of shutdown.

The other type of shutdown to explore is the SDL timer thread shutdown. An SDL timer thread shutdown occurs when the differential between the internal Cisco CallManager clock and the external operating system clock exceeds 16 seconds. When SDL timer thread shutdown happens, this trace appears in the Cisco CallManager trace:

```
03/22/2003 14:32:16.562 Cisco CallManager|CallManagerFailure − Indicates
some failure in the Cisco CallManager system. Host name of hosting
node.:NEROCM1 IP address of hosting node.:172.27.27.224 Reason code.:4
Additional Text [Optional]: App ID:Cisco CallManager Cluster
ID:NEROCM1−Cluster Node
ID:172.27.27.224|<CLID::NEROCM1−Cluster><NID::172.27.27.224><CT::Alarm>
```

The Cisco CallManager generally checks the timer threads once every second. Cisco CallManager adds 1 second to the current operating system time and stores that value away as expected time. Then, Cisco CallManager sleeps for 1 second. After Cisco CallManager awakes, it checks the new operating system time and subtracts the expected time. If the difference between these two times is more than 1 second, this warning statement appears in the Cisco CallManager trace:

```
CMProcMon::star_sdlVerification − Test Timer exceeded minimum timer latency
threshold of  1000 milliseconds, Actual latency: 1630 milliseconds
```

`Actual latency` in this statement shows that the internal Cisco CallManager SDL timer thread runs slow. Here, the difference between the Cisco CallManager expected time and the actual operating system time is 1.63 seconds.

If this difference exceeds 16 seconds, the Cisco CallManager shuts down and provides the shutdown reason code of `4`.

The most likely cause of an SDL timer thread shutdown is a lack of CPU time for Cisco CallManager.

Another application, such as VirusScan or STI Backup, has used most of the CPU resources for at least 16 seconds. Perfmon logs are vital to determine the root cause of this type of shutdown.

If the Cisco IP Telephony Applications Backup runs for a long period of time at high CPU utilization, a system crash can occur. For information on how to avoid this system crash, refer to:

- Check Settings on Backup Utility to Avoid High CPU section of the document Cisco CallManager Service Crash

Collect these files in the case of an SDL router thread or SDL timer thread shutdown:

- Detailed Cisco CallManager trace
- SDL trace

  Note: Set the SdlTraceTypeFlags to 0x8000CB15.
- Perfmon traces, if available, that show the percent CPU utilization of all processes that run on the box

  Note: You can capture these traces remotely to reduce performance impact on the CPU of the Cisco CallManager server.

How to Report Cisco CallManager Crashes to Cisco Technical Support

Diagnosis of the actual cause of a Cisco CallManager crash is difficult. In order to determine the cause and expedite a solution, Cisco Technical Support requires you to collect traces and Dr. Watson logs and upload the information to Cisco Technical Support case notes. You send the case notes to attach@cisco.com and provide the case number in the e−mail subject line. The procedure is:

1. Collect Cisco CallManager trace files from 30 minutes before and 15 minutes after the crash.

   The location of the traces is C:\Program Files\Cisco\Trace\CCM.
2. Collect SDL trace files from 30 minutes before and 15 minutes after the crash.

   The location of the traces is C:\Program Files\Cisco\Trace\SDL\CCM.
3. Collect the user.dmp and drwtsn32.log files.

   The location of the files is C:\Documents and Settings\All Users\Documents\DrWatson.
4. Select **Start > Programs > Administrative Tools > Event Viewer** to gather System and Application event log files from the Event Viewer.

   You can skip this step, if the event log data is not necessary. But, dump the System and Application events and filter out only the events from approximately 30 minutes before the crash. Investigate these events before you send them to Cisco Technical Support. You can find an event that warrants more attention.

   Caution: Be careful if you use Event Viewer, a built−in Microsoft utility, to dump these events to a text file. In a system that has high CPU utilization, this use of Event Viewer can easily starve all other processes from the CPU. These processes include the Cisco CallManager KeepAlive process that maintains phone registrations.

   You can use a shareware utility with the name elogdmp.exe to dump all the entries in the individual logs to a text file. The CPU implications are negligible when you use the elogdmp.exe tool. Issue this command from a DOS prompt:

```
elogdmp COMPUTER_NAME Application > AppEvents.txt

elogdmp COMPUTER_NAME System > SysEvents.txt
```

5. Compress all files as zip files in the order that this step shows before you e−mail and copy the files.

   Use WinZip version 8 to compress the files. (Cisco has a site license for this utility.) In general, files copy to a local machine for faster evaluation. Files that you compress use less space, and you can move these files much more quickly than raw file formats.

   a. Compress the user.dmp and drwtsn32.log files together.

      Immediately send and copy this zip file. Provide a descriptive symptom definition and include the exact Cisco CallManager version, appropriate device loads, and Cisco IOS® Software versions. If any special patches are in use, ensure that you make this fact clear.
   b. Compress the Cisco CallManager and SDL trace files together.

      Send and copy this zip file while you wait for contact.
   c. Compress the perfmon logs together.

      Send and copy this zip file while you wait for contact.
   d. Compress the event log entries together.

      Send and copy this zip file while you wait for contact.
6. After you have gathered all the traces and logs, compress the files and send the zip file to attach@cisco.com. Provide the case number in the e−mail subject line.

Related Information

- **Cisco CallManager Service Crash**
- **Troubleshooting Cisco CallManager Crashes**
- **Voice Technology Support**
- **Voice and Unified Communications Product Support**
- **Troubleshooting Cisco IP Telephony**
- **Technical Support & Documentation − Cisco Systems**

Updated: Feb 12, 2010 Document ID: 46806