Chunk Validation During Scheduler Heapcheck

First Published: May 16, 2006
Last Updated: May 16, 2006

The Chunk Validation During Scheduler Heapcheck feature enables you to check the memory chunk structures in a router and acquire the latest crash information in order to detect the processes that are corrupting the chunks. This crash information is also reliable and helpful in effectively debugging the processes that are corrupting the chunk structures.

Finding Feature Information in This Module

Your Cisco IOS software release may not support all of the features documented in this module. To reach links to specific feature documentation in this module and to see a list of the releases in which each feature is supported, use the "Feature Information for Chunk Validation During Scheduler Heapcheck" section on page 11.

Finding Support Information for Platforms and Cisco IOS Catalyst OS Software Images

Use Cisco Feature Navigator to find information about platform support and Cisco IOS and Catalyst OS software image support. To access Cisco Feature Navigator, go to http://www.cisco.com/go/cfn. An account in Cisco.com is not required.

Contents

CISCO SYSTEMS

Corporate Headquarters:
Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

Restrictions for Chunk Validation During Scheduler Heapcheck

This feature has the following restrictions:

- We recommend you not use this feature on any live router.
- We recommended you use this feature when the network traffic is low.

Note This feature should be used only as a last resort after all other possible methods to detect processes that are corrupting the chunks have failed.

Information About Chunk Validation During Scheduler Heapcheck

This feature is useful in detecting and deducing the memory corruption problems on chunk structures thereby facilitating in debugging the applications that are corrupting the router memory. With this feature, once you detect the process that is corrupting the memory chunk structures on a router, you can easily obtain the latest crash information. The crash information contains data about the number of chunks destroyed, number of chunks created, and values of the previous, current, and next chunk headers.

Heapchecker is a function in the Scheduler Algorithm. The Scheduler Algorithm is used to allocate space in memory for running processes. When the Scheduler switches to a different process, the heapchecker checks either every chunk structure or the one you specify for detecting chunk corruption. This feature can have a significant impact on the performance of the router. Based on the Scheduler Algorithm, the Scheduler allocates the CPU to different processes and decides which process gets to use the CPU and when. This decision is based on the priority of each process. This feature is an enhancement over the checkheap process because processes corrupting chunk structures are also detected.

The checkheap process has a lower priority than the Scheduler Heapcheck and by the time it gets the chance to execute, the corruption would have occurred much earlier. The latest crash information is not obtained. This feature enables chunk corruption detection immediately as it validates the chunks for every process switch. You can invoke the **scheduler heapcheck process** command during the interval between two processes running in the allocated memory and detect possible memory corruption threats.

Benefits of Chunk Validation During Scheduler Heapcheck

The Chunk Validation During Scheduler Heapcheck feature is an enhancement over the Scheduler Heapcheck process because the crash information you receive is the latest, and hence reliable and accurate. By employing this feature you can monitor and detect the processes responsible for corruption in the chunk structures, and in the input/output (I/O) memory block.

With the former checkheap process you can check for memory corruption. However, the crash information you receive by using the checkheap process is not reliable and not useful for debugging purposes. The inaccurate and unreliable crash information result because by the time the checkheap process assumes priority to get executed, the memory chunk structures and blocks get corrupted a number of times. So the crash information that you receive is not the latest.

This feature has a distinct advantage over the checkheap process because it can detect any application corrupting the Cisco IOS chunk structures immediately before other processes scheduled to run. You can configure this feature to execute in the time interval between any two processes running in the allocated

memory, unlike the checkheap process, which takes time to assume priority to get executed. With this feature you have the option to validate the chunk elements belonging to a single chunk block if the allocator program counter of the chunk block is specified during configuration. This option reduces the overhead of validating all chunks.

How to Configure Chunk Validation During Scheduler Heapcheck

This section contains the following procedures:

Configuring Checking of Memory Chunk Structures

Perform this task to enable the checking of memory chunk structures.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **scheduler heapcheck process** [**memory**] [**checktype chunks** *allocator-pc*]
4. **exit**

DETAILED STEPS

	Command	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.

	Command	Purpose
Step 3	**scheduler heapcheck process** [**memory**] [**checktype chunks** *allocator-pc*] **Example:** `Router(config)# scheduler heapcheck process memory checktype chunks 0x429496727`	Configures the memory chunks and specifies the chunk structure that needs to be monitored.
Step 4	**exit** **Example:** `Router(config)# exit`	Exits from the global configuration mode.

Configuring Malloc Lite-Chunks

Perform this task to check for corruption of Malloc lite-chunks in the memory.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **scheduler heapcheck process** [**memory**] [**checktype lite-chunks**]
4. **exit**

DETAILED STEPS

	Command	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**scheduler heapcheck process**[**memory**] [**checktype lite-chunks**] **Example:** `Router(config)# scheduler heapcheck process memory checktype lite-chunks`	Checks the Malloc lite-chunks in the memory for corruption.
Step 4	**exit** **Example:** `Router(config)# exit`	Exits from the global configuration mode.

Configuration Examples for Chunk Validation During Scheduler Heapcheck

This section provides the following configuration examples:

Configuring Chunk Structures: Example

Chunk validation is not enabled by default when the router boots and cannot be configured to startup-config. You must explicitly configure the **chunks** keyword in the CLI once the router boots up. The following example shows how to configure the **chunks** keyword and check for a given PC (sibling as well as root):

```
Router(config)# scheduler heapcheck process memory checktype chunks 0x607F7B98
Checktype value 0x0
mempool_check_type = 0x20, mempool_check_pc = 0x607F7B98
```

Sample Output from the show chunk Command

In this example the output is displayed for the **show chunk** command. The **show chunk** command checks for the number of chunks and siblings created and destroyed.

```
Router(config)# show chunk

Chunk Manager:
  532 chunks created, 69 chunks destroyed
  261 siblings created, 68 siblings trimmed

Chunk element  Block Maximum  Element Element Total
cfgsize Ohead   size element    inuse   freed Ohead    Name
   16        0    20052      995       28     967    4080   Managed Chunk
0x64A27040
   16        4    10052      412      412       0    3408   List Elements
0x64A2BE94
   16        4    11028      453       78     375    3728   (sibling)
0x6593ED74

0x6593F6F0: EF4321CD 00000000 00000000 00000000  oC!M............
0x6593F700: 0x6593ED74 EF4321CD 00000000 00000000  e.mtoC!M........
```

Configuring Malloc Lite-Chunks: Example

The following example shows how to configure the Malloc **lite-chunks** keyword and check for the Malloc lite-chunk structures in the router memory:

```
Router(config)# scheduler heapcheck process memory processor checktype lite-chunks

Router(config)# show run | in sched
scheduler heapcheck process memory processor checktype magic lite-chunks
+
```

Sample Output from the show chunk Command

The following is the sample output from the **show chunk** command when the chunk name is MallocLite:

```
Router(config)# show chunk | in Malloc

   20       16    65588     1636       77    1559   32816   MallocLite
0x65EFC1C0
   96       16    65588      564       24     540   11392   MallocLite
0x65F0C1F4
   44       16    65588     1022       53     969   20568   MallocLite
0x65F1C228
   68       16    65588      743        9     734   15012   MallocLite
0x656436C0
  128       16    65588      442       18     424    8960   MallocLite
0x65F4C290
    8       16    65588     2337       19    2318   46840   MallocLite
0x65FBC298
0x65F4CA60: 00000000 15A3C78B 00000000 60843474  .....#G.....`.4t
0x65F4CA70: 65F4C290 EF4321CD 00000000 00000000  etB.oC!M........
0x65F4CA80: 00000000 00000000 00000000 00000000  ................
0x65F4CA90: 00000000 00000000 00000000 00000000  ................
0x65F4CAA0: 00000000 00000000 00000000 00000000  ................
0x65F4CAB0: 00000000 00000000 00000000 00000000  ................
0x65F4CAC0: 00000000 00000000 00000000 00000000  ................
0x65F4CAD0: 00000000 00000000 00000000 00000000  ................
```

Sample Output from the scheduler heapcheck process Command

To check for the memory corruption of chunk structures, configure the **scheduler heapcheck process** command with the **chunks** keyword. The following examples show the sample output for the **scheduler heapcheck process** command with the **checktype chunks** as keywords:

```
Router(config)# scheduler heapcheck process memory checktype chunks 0x64A2BE60
```

From the **show chunk** command output you can select the chunk that needs to be configured:

```
Router# show chunk

Chunk Manager:
532 chunks created, 69 chunks destroyed
261 siblings created, 68 siblings trimmed
Chunk element  Block Maximum  Element Element Total
cfgsize Ohead   size element    inuse   freed Ohead    Name
   16        0    20052      995       28     967    4080   Managed Chunk 0x64A27040
   16        4    10052      412      412       0    3408   List Elements 0x64A2BE94
   16        4    11028      453       78     375    3728   (sibling) 0x6593ED74
```

By dumping the show memory output of the selected chunk you can get the *allocator-pc* value in the block header of the chunk that needs to be configured on the CLI.

```
Router# show memory 0x64A2BE60 0x64A2BEFF

64A2BE60: FD0110DF AB1234CD FFFE0000 00000000  }.._+.4M.~......
64A2BE70: 62D0D0A8 607F7B98 64A2E5A8 64A27024  bPP(`.{.d"e(d"p$
64A2BE80: 8000138A 00000001 60862A24 00000001  ........`.*$....
64A2BE90: 64A57F04 64A2E5D8 6593ED74 00000000  d%..d"eXe.mt....
64A2BEA0: 64A2BE48 64AB85A0 0000012C 0000019C  d">Hd+. ...,....
64A2BEB0: 00100014 00000000 0000084D 0000085C  ...........M...\
64A2BEC0: 00000184 00000014 00C10000 644F9EC0  .........A..dO.@
64A2BED0: 4C697374 20456C65 6D656E74 73000000  List Elements...
```

Additional References

The following sections provide references related to Chunk Validation During Scheduler Heapcheck feature.

Related Documents

Related Topic	Document Title
Router memory commands: complete command syntax, command mode, defaults, usage guidelines, and examples	"Router Memory Commands" chapter in the *Cisco IOS Configuration Fundamentals Command Reference,* Release 12.4T

Standards

Standards	Title
None	—

MIBs

MIB	MIBs Link
None	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFCs	Title
None	—

Technical Assistance

Description	Link
The Cisco Technical Support & Documentation website contains thousands of pages of searchable technical content, including links to products, technologies, solutions, technical tips, and tools. Registered Cisco.com users can log in from this page to access even more content.	http://www.cisco.com/techsupport

Command Reference

This section documents one modified command only.

- **scheduler heapcheck process**

scheduler heapcheck process

To perform a check for corruption in memory blocks and in chunk structures when a process switch occurs, use the **scheduler heapcheck process** command in global configuration mode. To disable this function, use the **no** form of this command.

scheduler heapcheck process [**memory**] [**fast**] [**io**] [**multibus**] [**pci**] [**processor**] [**checktype** {**all** | **chunks** *allocator-pc* | **lite-chunks** | **magic** | **pointer** | **refcount**}]

no scheduler heapcheck process [**memory**] [**fast**] [**io**] [**multibus**] [**pci**] [**processor**] [**checktype** {**all** | **chunks** *allocator-pc* | **lite-chunks** | **magic** | **pointer** | **refcount**}]

Syntax Description

memory	(Optional) Specifies checking all memory blocks and memory pools.
fast	(Optional) Specifies checking the fast memory block.
io	(Optional) Specifies checking the I/O memory block.
multibus	(Optional) Specifies checking the multibus memory block.
pci	(Optional) Specifies checking the process control information memory block.
processor	(Optional) Specifies checking the processor memory block.
checktype	(Optional) Specifies checking the checktype memory block.
all	(Optional) Specifies checking the value of the block and chuck **magic**, **red zone**, **size**, **refcount**, and **pointers** (next and previous).
chunks	(Optional) Specifies checking only the chunk memory structures.
allocator-pc	(Optional) Specifies allocator program counter (PC) of the root chunk in hexadecimal value.
lite-chunks	(Optional) Specifies checking only the Malloc lite chunks in the memory.
magic	(Optional) Specifies checking the block **magic**, **red zone**, and **size**.
pointer	(Optional) Specifies checking the value of next and previous pointers.
refcount	(Optional) Specifies checking the value of the block magic, chunk magic and refcount.

Command Default

This command is disabled by default. If no keywords are specified, a check for corruption will be performed on all the memory blocks and memory pools. Chunk validation does not occur in the default mode.

Command Modes

Global configuration

Command History

Release	Modification
12.2(15)T	This command was introduced.
12.4(8)	The **chunks** and **lite-chunks** keywords were added.

Usage Guidelines

While configuring this command, you can choose none or all memory block keywords (**fast**, **io**, **multibus**, **pci**, **processor**, and **checktype**). Enabling this command has a significant impact on router performance. By configuring the **chunks** keyword you can check for any corruption in the chunk structures of the memory. By configuring the **lite-chunks** keyword you can detect corruption of the Malloc lite-chunks in the memory structures. Chunk validation is not enabled by default and cannot be configured to startup-config. You must explicitly configure the chunks keyword in the CLI once the router boots up.

Examples

Check for Corruption in the I/O Memory Block

The following example shows how to check for corruption in the I/O memory block when a process switch occurs. In this example, the values of only the block magic, red zone, and size will be checked.

```
Router# scheduler heapcheck process memory io checktype magic
```

Check for Corruption in the Processor Memory Block

The following example shows how to sanity check for corruption in the processor memory block when a process switch occurs. In this example, the values of only the next and previous pointers will be checked.

```
Router# scheduler heapcheck process memory processor checktype pointer refcount
```

Check for Corruption in the Memory Chunk Structures

The following example shows how to check for corruption in the memory chunk structures when a process switch occurs. You must specify the value for *allocator-pc* argument.

```
Router# scheduler heapcheck process memory processor checktype chunks 4294967293
```

Check for Corruption in the Malloc Lite-Chunk structures

The following example shows how to check for corruption in the Malloc lite-chunk structures when a process switch occurs.

```
Router# scheduler heapcheck process memory processor checktype lite-chunks
```

Related Commands

Command	Description
memory sanity	Performs sanity check for corruption in buffers and queues.

Feature Information for Chunk Validation During Scheduler Heapcheck

Table 1 lists the release history for this feature.

Not all commands may be available in your Cisco IOS software release. For release information about a specific command, see the command reference documentation.

Use Cisco Feature Navigator to find information about platform support and software image support. Cisco Feature Navigator enables you to determine which Cisco IOS and Catalyst OS Software images support a specific release, feature set, or platform. To access Cisco Feature Navigator go to http://www.cisco.com/go/cfn. An account on Cisco.com is not required.

Note Table 1 lists only the Cisco IOS software release that introduced support for a given feature in a given Cisco IOS software release train. Unless noted otherwise, subsequent releases of that Cisco IOS software release train also support that feature.

Table 1 Feature Information for Chunk Validation During Scheduler Heapcheck

Feature Name	Releases	Feature Information
Chunk Validation During Scheduler Heapcheck	12.4(8)	In Cisco IOS Release 12.4(8) this feature was introduced. The following sections provide information about this feature: • Information About Chunk Validation During Scheduler Heapcheck, page 2 • Configuration Examples for Chunk Validation During Scheduler Heapcheck, page 5 • Command Reference, page 8