Conditional Debug, Radioactive Tracing, and Packet Tracing

Introduction to Conditional Debugging

The Conditional Debugging feature allows you to selectively enable debugging and logging for specific features based on the set of conditions you define. This feature is useful in systems where a large number of features are supported.

The Conditional debug allows granular debugging in a network that is operating at a large scale with a large number of features. It allows you to observe detailed debugs for granular instances within the system. This is very useful when we need to debug only a particular session among thousands of sessions. It is also possible to specify multiple conditions.

A condition refers to a feature or identity, where identity could be an interface, IP Address, or a MAC address and so on.

This is in contrast to the general debug command, that produces its output without discriminating on the feature objects that are being processed. General debug command consumes a lot of system resources and impacts the system performance.

Introduction to Radioactive Tracing

Radioactive tracing (RA) provides the ability to stitch together a chain of execution for operations of interest across the system, at an increased verbosity level. This provides a way to conditionally print debug information (up to DEBUG Level or a specified level) across threads, processes and function calls.

Note

- The radioactive tracing supports First-Hop Security (FHS).

  For more information on First Hop Security features, see *System Management > Wireless Multicast > Information About Wireless Multicast > Information About IPv6 Snooping*.
- The radioactive tracing filter does not work, if the certificate is not valid.
- For effective debugging of issues on mesh features, ensure that you add both Ethernet and Radio MAC address as conditional MAC for RA tracing, while collecting logs.
- To enable debug for wireless IPs, use the **debug platform condition feature wireless ip** *ip-address* command.

Table 1: Components Supporting Radio Active Tracing

Components	Details
SISF or FHS	The first-hop security features, includes IPv6 Address Glean and IPv6 Device Tracking. For more information, see *Information About IPv6 Snooping*.
LISP	Locator or ID Separation Protocol.

Conditional Debugging and Radioactive Tracing

Radioactive Tracing when coupled with Conditional Debugging, enable us to have a single debug CLI to debug all execution contexts related to the condition. This can be done without being aware of the various control flow processes of the feature within the box and without having to issue debugs at these processes individually.

Note Use the **clear platform condition all** command to remove the debug conditions applied to the platform.

Location of Tracefiles

By default the tracefile logs will be generated for each process and saved into either the **/tmp/rp/trace** or **/tmp/fp/trace** directory. In this temp directory, the trace logs are written to files, which are of 1 MB size each. You can verify these logs (per-process) using the **show platform software trace message** *process_name* **chassis active R0** command. The directory can hold up to a maximum of 25 such files for a given process. When a tracefile in the **/tmp** directory reaches its 1MB limit or whatever size was configured for it during the boot time, it is rotated out to an archive location in the **/crashinfo** partition under **tracelogs** directory.

The **/tmp** directory holds only a single tracefile for a given process. Once the file reaches its file size limit it is rotated out to **/crashinfo/tracelogs**. In the archive directory, up to 25 files are accumulated, after which the oldest one is replaced by the newly rotated file from **/tmp**. File size is process dependent and some processes uses larger file sizes (upto 10MB). Similarly, the number of files in the **tracelogs** directory is also decided by the process. For example, WNCD process uses a limit of 400 files per instance, depending on the platform.

The tracefiles in the crashinfo directory are located in the following formats:

1. Process-name_Process-ID_running-counter.timestamp.gz

   Example: IOSRP_R0-0.bin_0.14239.20151101234827.gz

2. Process-name_pmanlog_Process-ID_running-counter.timestamp.bin.gz

   Example: wncmgrd_R0-0.27958_1.20180902081532.bin.gz

Configuring Conditional Debugging (GUI)

Procedure

Step 1	Choose **Troubleshooting** > **Radioactive Trace**.
Step 2	Click **Add**.
Step 3	Enter the **MAC/IP Address**. The MAC address can be either in *xx:xx:xx:xx:xx:xx*, *xx-xx-xx-xx-xx-xx*, or *xxxx.xxxx.xxxx* format.
Step 4	Click **Apply to Device**.
Step 5	Click **Start** to start or **Stop** to stop the conditional debug.
Step 6	Click **Generate** to create a radioactive trace log.
Step 7	Click the radio button to set the time interval.
Step 8	Click the **Download Logs** icon that is displayed next to the trace file name, to download the logs to your local folder.
Step 9	Click the **View Logs** icon that is displayed next to the trace file name, to view the log files on the GUI page. Click **Load More** to view more lines of the log file.
Step 10	Click **Apply to Device**.

Configuring Conditional Debugging

Follow the procedure given below to configure conditional debugging:

Procedure

	Command or Action	Purpose
Step 1	**debug platform condition feature wireless mac** {*mac-address*} **Example:** `Device# debug platform condition feature wireless mac b838.61a1.5433`	Configures conditional debugging for a feature using the specified MAC address. **Note** This is supported with AP or client MAC/IP and also on CMX IP address and mobility peer IP.
Step 2	**debug platform condition start** **Example:** `Device# debug platform condition start`	Starts conditional debugging (this will start radioactive tracing if there is a match on one of the conditions above). **Note** This is supported with AP or client MAC/IP and also on CMX IP address and mobility peer IP.

Step 3	**show platform condition** OR **show debug**
	Example:
	`Device# show platform condition`
	`Device# show debug`
		Displays the current conditions set.

Step 4	**debug platform condition stop** **Example:** `Device# debug platform condition stop`	Stops conditional debugging (this will stop radioactive tracing). **Note** This is supported with AP or client MAC/IP and also on CMX IP address and mobility peer IP.
Step 5	**show logging profile wireless** [**counter** | **[last]**{*x days/hours*} | **filter mac**{<*mac address*>} **[to-file]**{<*destination*>} **Example:** `Device# show logging profile wireless start last 20 minutes to-file bootflash:logs.txt`	Displays the logs from the latest wireless profile. **Note** You can use either the *show logging profile wireless* command or *show logging process* command to collect the logs.
Step 6	**show logging process** <*process name*> **Example:** `Device# show logging process wncd to-file flash:wncd.txt`	Displays the logs collection specific to the process.

	Command or Action	Purpose
Step 7	**clear platform condition all** **Example:** `Device# clear platform condition all`	Clears all conditions.

What to do next

Note The command **request platform software trace filter-binary wireless** {*mac-address*} generates 3 flash files:

- *collated_log_<.date..>*
- *mac_log <..date..>*
- *mac_database .. file*

Of these, *mac_log <..date..>* is the most important file, as it gives the messages for the MAC address we are debugging. The command **show platform software trace filter-binary** also generates the same flash files, and also prints the mac_log on the screen.

Radioactive Tracing for L2 Multicast

To identify a specific multicast receiver, specify the MAC address of the joiner or the receiver client, Group Multicast IP address and Snooping VLAN. Additionally, enable the trace level for the debug. The debug level will provide detailed traces and better visibility into the system.

debug platform condition feature multicast controlplane mac *client-mac-addr* **ip** *group-ip-addr* **vlan** *id* **level** *debug level*

Recommended Workflow for Trace files

The Recommended Workflow for Trace files is listed below:

1. To request the tracelogs for a specific time period.

   EXAMPLE 1 day.

   Use the command:

   Device#**show logging process wncd to-file flash:wncd.txt**

2. The system generates a text file of the tracelogs in the location /flash:
3. Copy the file off the switchdevice. By copying the file, the tracelogs can be used to work offline. For more details on copying files, see section below.
4. Delete the tracelog file (.txt) file from /flash: location. This will ensure enough space on the switchdevice for other operations.

Copying Tracefiles Off the Box

An example of the tracefile is shown below:

```
Device# dir crashinfo:/tracelogs
Directory of crashinfo:/tracelogs/

50664 -rwx 760 Sep 22 2015 11:12:21 +00:00 plogd_F0-0.bin_0.gz
50603 -rwx 991 Sep 22 2015 11:12:08 +00:00 fed_pmanlog_F0-0.bin_0.9558.20150922111208.gz
50610 -rw- 11 Nov 2 2015 00:15:59 +00:00 timestamp
50611 -rwx 1443 Sep 22 2015 11:11:31 +00:00
auto_upgrade_client_sh_pmanlog_R0-.bin_0.3817.20150922111130.gz
50669 -rwx 589 Sep 30 2015 03:59:04 +00:00 cfgwr-8021_R0-0.bin_0.gz
50612 -rwx 1136 Sep 22 2015 11:11:46 +00:00 reflector_803_R0-0.bin_0.1312.20150922111116.gz
50794 -rwx 4239 Nov 2 2015 00:04:32 +00:00 IOSRP_R0-0.bin_0.14239.20151101234827.gz
50615 -rwx 131072 Nov 2 2015 00:19:59 +00:00 linux_iosd_image_pmanlog_R0-0.bin_0
```

The trace files can be copied using one of the various options shown below:

```
Device# copy crashinfo:/tracelogs ?
  crashinfo: Copy to crashinfo: file system
  flash: Copy to flash: file system
  ftp: Copy to ftp: file system
  http: Copy to http: file system
  https: Copy to https: file system
  null: Copy to null: file system
  nvram: Copy to nvram: file system
  rcp: Copy to rcp: file system
  running-config Update (merge with) current system configuration
  scp: Copy to scp: file system
  startup-config Copy to startup configuration
  syslog: Copy to syslog: file system
  system: Copy to system: file system
  tftp: Copy to tftp: file system
  tmpsys: Copy to tmpsys: file system
```

The general syntax for copying onto a TFTP server is as follows:

```
Device# copy source: tftp:
Device# copy crashinfo:/tracelogs/IOSRP_R0-0.bin_0.14239.20151101234827.gz tftp:
Address or name of remote host []? 2.2.2.2
Destination filename [IOSRP_R0-0.bin_0.14239.20151101234827.gz]?
```

Note It is important to clear the generated report or archive files off the switch in order to have flash space available for tracelog and other purposes.

Configuration Examples for Conditional Debugging

The following is an output example of the *show platform condition* command.

```
Device# show platform condition
Conditional Debug Global State: Stop
Conditions Direction
------------------------------------------------------------------------------------------|---------
MAC Address 0024.D7C7.0054 N/A
Feature Condition Type Value
----------------------|----------------------|--------------------------------
Device#
```

The following is an output example of the *show debug* command.

```
Device# show debug
IOSXE Conditional Debug Configs:
Conditional Debug Global State: Start
Conditions Direction
------------------------------------------------------------------------------------------|---------
MAC Address 0024.D7C7.0054 N/A
Feature Condition Type Value
----------------------|----------------------|--------------------------------
Packet Infra debugs:
Ip Address Port
------------------------------------------------------|----------
Device#
```

Verifying Conditional Debugging

The table shown below lists the various commands that can be used to verify conditional debugging:

Command	Purpose
show platform condition	Displays the current conditions set.
show debug	Displays the current debug conditions set.
show platform software trace filter-binary	Displays logs merged from the latest tracefile.
request platform software trace filter-binary	Displays historical logs of merged tracefiles on the system.

Example: Verifying Radioactive Tracing Log for SISF

The following is an output example of the *show platform software trace message ios chassis active R0 | inc sisf* command.

```
Device# show platform software trace message ios chassis active R0 | inc sisf

2017/10/26 13:46:22.104 {IOSRP_R0-0}{1}: [parser]: [5437]:  UUID: 0, ra: 0 (note):  CMD:
'show platform software trace message ios switch active R0 | inc sisf' 13:46:22 UTC Thu Oct
 26 2017
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
   FF8E802918 semaphore system unlocked
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Unlocking, count is now 0
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
   FF8E802918 semaphore system unlocked
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Unlocking, count is now 1
```

```
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Setting State to 2
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Start timer 0
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc  Timer value/granularity for 0 :299998/1000
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Updated Mac Timer : 299998
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Before Timer :  350000
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Timer 0, default value is 350000
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
   Allocating timer wheel for 0
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc No timer running
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Granularity for timer MAC_T1  is 1000
2017/10/26 13:46:10.667 {IOSRP_R0-0}{1}: [sisf]: [5437]:  UUID: 4800000000060, ra: 7 (debug):
  Gi1/0/5 vlan 10 aaaa.bbbb.cccc Current State :MAC-STALE, Req Timer : MAC_T1  Current Timer
 MAC_T1
```

Information About Packet Tracing

The Packet tracing feature cover details on how to perform data plane packet tracing for Cisco Catalyst 9800 Series Wireless Controller for Cloud software.

This feature identifies the following issues:

- Misconfiguration
- Capacity overload
- Software bugs while troubleshooting

This feature identifies what happens to a packet in your system. The conditional debugging packet tracing feature is used for accounting and capturing per-packet processing details for user-defined conditions.

You can trace packets on the controller using the following steps:

1. Enable conditional debugging on selected packets or traffic you want to trace on the controller.
2. Enable packet tracing (per-AP or per-Client).

Note You need to use per AP conditional debugging with MAC address as a filter when AP and controllers are in the same VLAN. If they are not in the same VLAN, the per AP packet tracing with MAC address does not capture packets as MAC address varies.

Limitation of Conditional Debugging Packet Tracing

MAC or IP filter only applies to the outer Ethernet or IP header, so if a packet is CAPWAP encapsulated, the MAC or IP does not apply to the inner 802.11 MAC or IP.

Configuring Conditional Debugging Packet Tracing

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged **EXEC** mode. Enter your password, if prompted.
Step 2	**debug platform packet-trace packet** *packet-count* **circular fia-trace data-size** *data-size* **Example:** `Device# debug platform packet-trace packet 8192 circular fia-trace data-size 2048`	Configures packet tracing to capture the last set of packets. Here, *packet-count*—Valid range is from 16 to 8192. *data-size*—Valid range is from 2048 to 16384 bytes.
Step 3	**debug platform packet-trace copy packet both size** *packet-size* **Example:** `Device# debug platform packet-trace copy packet both size 2048`	Configures packet tracing for a copy of packet data. Here, *packet-size*—Valid range is from 16 to 2048 bytes.
Step 4	**debug platform condition interface** {*intf-name* | **cpp**} {**mac** | **ipv4** | **match**} {**both** | **ingress** | **egress**} **Example:** Enables conditional debugging for TenGigabitEthernet 0/0/0 and match packets whose source and destination MAC is 0001.0001.0001: `Device# debug platform condition interface TenGigabitEthernet 0/0/0 mac 0001.0001.0001 both`	Enables conditional debugging for an interface, MAC, or IP filter. An interface refers to any physical port, port channel, internal vlan, SVI, or wireless client.
Step 5	**debug platform condition start** **Example:** `Device# debug platform condition start`	Starts conditional debugging packet tracing.
Step 6	**debug platform condition stop** **Example:** `Device# debug platform condition stop`	Stops conditional debugging packet tracing.
Step 7	**show platform hardware chassis active qfp feature packet-trace packet all | redirect bootflash:packet_trace.txt**	Redirects all traced packets to bootflash.

	Command or Action	Purpose
	Example: `Device# show platform hardware chassis active qfp feature packet-trace packet all | redirect bootflash:packet_trace.txt`	Converts the packet_trace.txt to pcap and downloads the pcap files. You can do so using the following link: http://wwwin-dharton-dev.cisco.com/pactrac2pcap.html

Configuring Conditional Debugging Packet Tracing per AP

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged **EXEC** mode. Enter your password, if prompted.

Step 2	**debug platform condition interface** {*intf-name* | **cpp**} {**mac** [**mac-address** | **access-list** *acl-name*] | **ipv4** | **match**} {**both** | **ingress** | **egress**}
	Example:
	`Device# debug platform condition interface TenGigabitEthernet 0/0/0 mac 0001.0001.0001 both`
	`Device# debug platform condition interface TenGigabitEthernet 0/0/0 mac access-list mac-acl-name both`
		Enables conditional debugging with MAC filter.
		Herein, the CLI matches the packets whose source or destination MAC address is 0001.0001.0001.

Step 3	**debug platform condition interface TenGigabitEthernet** *intf-number* **match mac** {**H.H.H** | **any** | **host**} {**both** | **ingress** | **egress**} **Example:** `Device# debug platform condition interface TenGigabitEthernet 0/0/0 match mac 0001.0001.0001 both`	Enables conditional debugging with inline MAC ACL.

Step 4	**debug platform condition interface TenGigabitEthernet** *intf-number* **ipv4** {*A.B.C.D/nn* | **access-list** *acl-name* | **both** | **egress** | **ingress**} {**both** | **egress** | **ingress**}
	Example:
	`Device# debug platform condition interface TenGigabitEthernet 0/0/0 ipv4 192.168.1.2/32 both`
	`Device# debug platform condition interface TenGigabitEthernet 0/0/0 ipv4 access-list ip-acl-name both`
		Enables conditional debugging with IP filter.
		Here,
		intf-number—Is the GigabitEthernet interface number.Valid range is from 1 to 32.

	Command or Action	Purpose
	`Device# debug platform condition interface TenGigabitEthernet 0/0/0 match ipv4 192.168.1.2/32 both`	

Configuring Conditional Debugging Packet Tracing per Client (GUI)

Procedure

Step 1 Choose **Troubleshooting** > **Radioactive Trace**.

Step 2 Click **Add**.

Step 3 In the Add MAC/IP Address window, enter the **MAC/IP Address**.

Step 4 Click **Apply to Device**.

Configuring Conditional Debugging Packet Tracing per Client

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enables privileged **EXEC** mode. Enter your password, if prompted.
Step 2	**debug platform condition interface** {*intf-name* | **cpp** *cpp-handle-index*} {**mac** | **ipv4** | **match** [**ipv4** | **ipv6** | **mac**]} {**both** | **ingress** | **egress**} **Example:** `Device# debug platform condition interface cpp 0xa0000001 match ipv4 protocol icmp host 192.168.1.100 host 192.168.1.1 both`	Enables conditional debugging for a wireless client interface. Here, *cpp-handle-index*—Valid range is from 1 to 4294967295.

Verifying Conditional Debugging Packet Tracing Configuration

To view the summary of the traced packet, use the following command:

```
Device# show platform packet-trace summary
```

To view a specific traced packet, use the following command:

```
Device# show platform packet-trace packet packet-number
```

To view the wireless client interface handle, use the following command:

```
Device# show platform hardware chassis active qfp feature wireless wlclient cpp-client
mac-address client-mac details
Device# show platform hardware chassis active qfp feature wireless wlclient cpp-client
mac-address 8825.93b0.b51f details
Client Details for client cpp_if_handle: 0x34
Name : WLCLIENT-IF-0x00a0000001
Mac Addr : 8825.93b0.b51f
pal_if_handle : 0xa0000001
Mobility State : LOCAL
Multicast Action : FORWARD
Auth State : RUN
```

Feature History for Wireless Client Debug Bundle

This table provides release and related information about the feature explained in this section.

This feature is also available in all the releases subsequent to the one in which they are introduced in, unless noted otherwise.

Table 2: Feature History for Client Debug Bundle

Release	Feature	Feature Information
Cisco IOS XE Dublin 17.11.1	Wireless Client Debug Bundle	Client debug bundle includes AP logs along with the existing controller bundle, collected in a tar file through a single **debug** command.

Information About Wireless Client Debug Bundle

The log collection of client radioactive trace, packet capture, and the output of various **show** commands are useful in troubleshooting wireless client issues. In the earlier releases, logs were collected through various individual steps and commands. Now, client debug bundle collates radioactive trace debug logs, packet captures in a control plane, and the output of **show** commands related to clients, collected in a tar file through a single **debug** command. From Cisco IOS XE Cupertino 17.11.1, client debug bundle collates AP logs along with the existing controller bundle.

Note Client debug bundle is not supported on High Availability (HA) with Stateful Switch Over (SSO).

Note When you enable the **all** command option on the AP console, the command activates the debug logging for all clients, which can result in an excessive amount of logs being printed in the console.

Types of Logs Collected

The client debug bundle logs are collected in a tar file format on bootflash, through a single **debug** command.

The following example displays the file formats of logs collected from a client device with MAC address 8cXX.90XX.fdXX:

Example:

The final tar file that is generated is wireless_bundle_123456_UTC_Oct_20_2022.tar.

The following files are extracted from the wireless_bundle_123456_UTC_Oct_20_2022.tar file:

- wireless_bundle_8cXX.90XX.fdXX.tar (client radioactive trace debug log)
- epc_135790_UTC_Oct_20_2022.pcap (packet capture in a control plane)
- ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.tgz (AP logs)

The following files are extracted from wireless_bundle_8cXX.90XX.fdXX.tar client radioactive trace debug log file:

- show_tech_support_wireless_client_before_RA_start_8cXX.90XX.fdXX_134941_UTC_Oct_20_2022.txt
- ra_trace_8cXX.90XX.fdXX_135055_UTC_Oct_20_2022.log
- ra_trace_internal_8cXX.90XX.fdXX_135057_UTC_Oct_20_2022.log
- show_tech_support_wireless_client_after_RA_stop_8cXX.90XX.fdXX_135055_UTC_Oct_20_2022.txt

The following files are extracted from ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.tgz AP log file:

- ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.messages
- ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.syslogs
- ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.tech_cdb_0
- ap_3802_cisco_client_bundle.17.11.0.61.20221020.135154.tech_cdb_1

Collecting Wireless Client Debug Bundle (CLI)

Procedure

	Command or Action	Purpose
Step 1	**debug wireless bundle client mac** *H.H.H* **Example:** `Device# debug wireless bundle client mac aaaa.bbbb.cccc`	Adds client MAC addresses for which debug logs are required. You can add up to 32 client MAC addresses to the command. To delete the MAC addresses, run the **no** form of this command.
Step 2	**debug wireless bundle client start** **Example:**	Starts the collection of the client debug bundle for wireless clients.

	Command or Action	Purpose
	`Device# debug wireless bundle client start`	
Step 3	(Optional) **debug wireless bundle client start ap-archive site-tag** *default-site-tag* **level** {**critical** | **debug** | **error** | **verbose**} **Example:** `Device# debug wireless bundle client start ap-archive site-tag default-site-tag level debug`	Enables the AP archive collection on a site tag. Specifies the AP archive levels as well.
Step 4	(Optional) **debug wireless bundle client start epc** **Example:** `Device# debug wireless bundle client start epc`	Enables embedded packet capture (EPC) in a control plane. **Note** If EPC is already enabled and is active from a different source, debug bundle with EPC cannot be started. To use EPC with debug bundle, stop EPC (enabled from a different source) and restart it with debug bundle.
Step 5	(Optional) **debug wireless bundle client start monitor-time** *monitor-time* **Example:** `Device# debug wireless bundle client start monitor-time 30`	Configures the maximum time, in minutes, to trace the condition. The default time is 30 minutes.
Step 6	(Optional) **debug wireless bundle client stop-all collect** {**all** | **mac** *H.H.H*} **Example:** `Device# debug wireless bundle client stop-all collect all`	Stops the collection of the debug bundle for wireless clients.
Step 7	(Optional) **debug wireless bundle client abort** **Example:** `Device# debug wireless bundle client abort`	Cancels the collection of the debug bundle for wireless clients.