Stream TCP Inspector

Stream TCP Inspector Overview

Type	Inspector (stream)
Usage	Inspect
Instance type	Multiton
Other Inspectors Required	None
Enabled	`true`

Transmission Control Protocol (TCP) is a connection-oriented, stateful transport layer protocol. TCP can reliably transmit an ordered stream of bytes between a client and a server over an IP network. TCP permits only one connection with the same connection parameter values to exist at a time. A host operating system manages the states of a TCP connection.

The `stream_tcp` inspector provides TCP flow tracking, stream normalization, and stream reassembly. Each stream TCP inspector can handle the TCP traffic for one or more hosts in your network. In addition, if you have enough information about the hosts that are sending the TCP traffic to your network, you can configure a `stream_tcp` inspector for those hosts.

In a network analysis policy (NAP), Snort applies each configured `stream_tcp` inspector to the TCP services defined in the `binder` inspector configuration.

You can configure multiple stream TCP inspectors to handle various operating systems and TCP traffic.

The `stream_tcp` inspector configuration includes:

- Operating system on the TCP host
- Operating system options: how overlaps are handled during reassembly

- Traffic handling options: the maximum number of bytes or segments in a session or direction
- TCP stream reassembly options: the maximum reassembled PDU size

Note In inline IPS mode, the `stream_tcp` inspector normalizes the payload stream so that overlaps always resolve to the first copy seen. Each stream TCP inspector handles repeated SYNs, RST validation, and timestamp checks.

Best Practices for Configuring the Stream TCP Inspector

Consider the following best practices when configuring a `stream_tcp` inspector:

- Do not deploy the sensing interfaces on a device so that Snort can only inspect one side of a flow. You can enable the `reassemble_async` parameter in the `stream_tcp` inspector to process asymmetric traffic. However, the stream TCP inspector cannot process asymmetric traffic in all cases. For example, a response to an HTTP HEAD request can cause the HTTP inspector to get out of sync. In IDS mode, the lack of TCP acknowledgements makes evasions much easier.

  For IPS mode, we recommend that you deploy a device only if Snort can inspect both sides of a flow.
- Create a `stream_tcp` inspector for each TCP host operating system that you expect to send or receive TCP traffic. You can have multiple versions of the `stream_tcp` inspector in the same network analysis policy. The TCP policies defined in each `stream_tcp` inspector are applied to the TCP hosts specified in the `binder` inspector.
- To enable IPS mode, set the `normalizer.tcp.ips` parameter in the `normalizer` inspector to `true`.
- In the advanced settings in your network analysis policy (NAP), confirm that the networks which you want to identify in a custom, target-based `stream_tcp` inspector match or are a subset of the networks, zones, and VLANs handled by its parent NAP.
- The system builds a separate network map for each leaf domain. In a multidomain deployment, using literal IP addresses to constrain this configuration can have unexpected results. Using override-enabled objects allows descendant domain administrators to tailor Global configurations to their local environments.
- To generate events and, in an inline deployment, drop offending packets, enable the `stream_tcp` inspector rules (GID 129).

Best Practices for TCP Stream Reassembly

The `stream_tcp` inspector collects and reassembles all packets that are part of a TCP session's server-to-client communication stream, client-to-server communication stream, or both. TCP stream reassembly allows Snort to inspect the stream as a single, reassembled entity, a protocol data unit (PDU), rather than inspecting only the individual packets that are part of a given stream. If the PDU is large, the rules engine splits it into several parts.

Stream reassembly allows Snort to identify stream-based attacks, which it may not detect when inspecting individual packets. You can specify which communication streams to reassemble based on your network

needs. For example, when monitoring traffic on your web servers, you may only want to inspect client traffic because you are less likely to receive malicious traffic from your own web server.

For each `stream_tcp` inspector, you can specify a list of TCP ports in the `binder` configuration. The TCP stream inspector automatically and transparently includes the configured ports to identify and reassemble traffic. If adaptive profiles updates are enabled, you can list services that identify traffic to reassemble, either as an alternative to ports or in combination with ports.

Specify TCP ports in the `binder` configuration for the following Snort inspectors:

- `dnp3`
- `ftp_server`
- `gtp_inspect` (ports provided by default)
- `http_inspect`
- `imap`
- `iec104` (ports provided by default)
- `mms` (ports provided by default)
- `modbus` (ports provided by default)
- `pop`
- `sip`
- `smtp`
- `ssh`
- `ssl`
- `telnet`

Note When you reassemble multiple traffic types (client, server, both), Snort resource demands may increase.

Stream TCP Inspector Parameters

Stream TCP reassembly configuration

The `binder` inspector defines the TCP stream reassembly configuration for the network analysis policy (NAP). You specify the host IP addresses to which you want to apply the TCP stream reassembly policy. The stream TCP inspector is automatically associated with the ports configured in the `binder` for the NAP. For more information, see the Binder Inspector Overview.

Note The system builds a separate network map for each leaf domain. In a multidomain deployment, using literal IP addresses to constrain this configuration can have unexpected results. Using override-enabled objects allows descendant domain administrators to tailor Global configurations to their local environments.

The `default` setting in the default policy specifies all IP addresses on your monitored network segment that are not covered by another target-based policy. Therefore, you cannot and do not need to specify an IP address or CIDR block/prefix length for the default policy, and you cannot leave this setting blank in another policy or use address notation to represent `any` (for example, `0.0.0.0/0` or `::/0`).

policy

Specifies the operating system of the target host or hosts. The operating system determines the appropriate TCP reassembly policy and operating system characteristics. You can define only one `policy` parameter for each stream TCP inspector.

Note If you set the `policy` parameter to `first`, Snort may provide some protection, but miss attacks. You should edit the `policy` parameter of the TCP stream inspector to specify the appropriate operating system.

Type: enum

Valid values: Set a type of operating system for the `policy` parameter.

Table 1: TCP Operating System Policies

Policy	Operating Systems
`first`	unknown OS
`last`	Cisco IOS
`bsd`	AIX FreeBSD OpenBSD
`hpux_10`	HP-UX 10.2 and earlier
`hpux_11`	HP-UX 11.0 and later
`irix`	SGI Irix
`linux`	Linux 2.4 kernel Linux 2.6 kernel
`macos`	Mac OS 10 (Mac OS X)
`old_linux`	Linux 2.2 and earlier kernel

Policy	Operating Systems
`solaris`	Solaris OS SunOS
`vista`	Windows Vista
`windows`	Windows 98 Windows NT Windows 2000 Windows XP
`win_2003`	Windows 2003

Default value: `bsd`

max_window

Specifies the maximum TCP window size permitted by a receiving host. You can specify an integer less than `65535`, or specify `0` to disable inspection of the TCP window size.

Caution The upper limit of `max_window` is the maximum window size permitted by RFC 1323. You can set the upper limit to prevent an attacker from evading detection, however, a significantly large maximum TCP window size may create a self-imposed denial of service.

Type: integer

Valid range: `0` to `1,073,725,440`

Default value: `0`

overlap_limit

Specifies the maximum number of overlapping segments allowed in each TCP session. Specify `0` to permit an unlimited number of overlapping segments. If you set a number between `0` and `255`, segment reassembly stops for the session.

Enable rule 129:7 to generate events and, in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `4,294,967,295 (max32)`

Default value: 0

max_pdu

Specifies the maximum reassembled protocol data unit (PDU) size.

Type: integer

Valid range: `1460` to `32768`

Default value: `16384`

reassemble_async

Ensures that data is queued for reassembly before traffic is seen in both directions. When the monitored network is an asynchronous network, you must enable the `reassemble_async` parameter. An asynchronous network only permits traffic in a single direction and one flow at a time. If the `reassemble_async` parameter is enabled, Snort does not reassemble TCP streams to increase performance.

Note The stream TCP inspector cannot correctly process asymmetric traffic in all cases. For example, a response to an HTTP HEAD request can cause the HTTP inspector to get out of sync. In IDS mode, the lack of TCP acknowledgements makes evasions much easier. For IPS mode, we recommend that you deploy a device only if the rules engine can inspect both sides of a flow.

The `reassemble_async` parameter is ignored for the Secure Firewall Threat Defense routed and transparent interfaces.

Type: boolean

Valid values: `true`, `false`

Default value: `true`

require_3whs

Specifies the number of seconds from start up after which the stream TCP inspector stops tracking midstream sessions. Specify `-1` to track all midstream TCP sessions, no matter when they occur.

Snort does not synchronize most protocol streams. Snort always picks up on SYN if it needs any of the handshake options (timestamps, window scale, or MSS). Typically, IPS efficacy is not improved by allowing midstream pickups.

Type: integer

Valid range: `-1` to `2,147,483,647 (max31)`

Default value: `-1`

queue_limit.max_bytes

Specifies the maximum number of bytes to queue for a session on one side of a TCP connection. Specify `0` to allow an unlimited number of bytes.

Caution We recommend that you contact Cisco TAC before changing the default setting of the `queue_limit.max_bytes` parameter.

Type: integer

Valid range: `0` to `4,294,967,295 (max32)`

Default value: `4,194,304`

queue_limit.max_segments

Specifies the maximum number of data segments to queue for a session on one side of a TCP connection. Specify `0` to allow an unlimited number of data segments.

Caution We recommend that you contact Cisco TAC before changing the default setting of the `queue_limit.max_segments` parameter.

Type: integer

Valid range: `0` to `4,294,967,295 (max32)`

Default value: `3072`

small_segments.count

Specifies a number that is above the expected amount of consecutive small TCP segments. Specify `0` to ignore the count of consecutive small TCP segments.

You must set the `small_segments.count` and `small_segments.maximum_size` parameters with the same type of value. Specify `0` for both parameters or set each parameter to a non-zero value.

Note Snort considers 2000 consecutive segments, even if each segment is `1` byte in length, above the normal amount of consecutive TCP segments.

Snort ignores the `small_segments.count` parameter for threat defense routed and transparent interfaces.

You can enable rule 129:12 to generate events and, in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `2048`

Default value: `0`

small_segments.maximum_size

Specifies the number of bytes which identify a TCP segment as larger than a small TCP segment. A small TCP segment size is in the range of `1` to `2048` bytes. Specify `0` to ignore the maximum size of a small segment.

Snort ignores the `small_segments.maximum_size` parameter for threat defense routed and transparent interfaces.

You must set the `small_segments.maximum_size` and `small_segments.count` parameters with the same type of value. Specify `0` for both parameters or set each parameter to a non-zero value.

Note A `2048` byte TCP segment is larger than a normal `1500` byte Ethernet frame.

You can enable rule 129:12 to generate events and, in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `2048`

Default value: `0`

session_timeout

Specifies the number of seconds that Snort keeps an inactive TCP stream in its state table. If the stream is not reassembled in the specified time, Snort deletes it from the state table. If the session is still alive and more packets appear, Snort handles the stream as a midstream flow.

We recommend that you set the `session_timeout` parameter to greater than or equal to the host TCP session timeout.

Type: integer

Valid range: `0` to `2,147,483,647 (max31)`

Default value: `180`

Stream TCP Inspector Rules

Enable the `stream_tcp` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 2: Stream TCP Inspector Rules

GID:SID	Rule Message
129:1	SYN on established session
129:2	data on SYN packet
129:3	data sent on stream not accepting data
129:4	TCP timestamp is outside of PAWS window
129:5	bad segment, adjusted size <= 0 (deprecated)
129:6	window size (after scaling) larger than policy allows
129:7	limit on number of overlapping TCP packets reached
129:8	data sent on stream after TCP reset sent
129:9	TCP client possibly hijacked, different ethernet address
129:10	TCP server possibly hijacked, different ethernet address
129:11	TCP data with no TCP flags set
129:12	consecutive TCP small segments exceeding threshold
129:13	4-way handshake detected
129:14	TCP timestamp is missing
129:15	reset outside window
129:16	FIN number is greater than prior FIN

GID:SID	Rule Message
129:17	ACK number is greater than prior FIN
129:18	data sent on stream after TCP reset received
129:19	TCP window closed before receiving data
129:20	TCP session without 3-way handshake

Stream TCP Inspector Intrusion Rule Options

stream_reassemble

Specify whether to enable TCP stream reassembly on matching traffic. The `stream_reassemble` rule option includes four parameters: `stream_reassemble.action`, `stream_reassemble.direction`, `stream_reassemble.noalert`, and `stream_reassemble.fastpath`.

Syntax: `stream_reassemble: <enable|disable>, <server|client|both>, noalert, fastpath;`

Examples: `stream_reassemble: disable,client,noalert;`

stream_reassemble.action

Stop or start stream reassembly.

Type: enum

Syntax: `stream_reassemble: <action>;`

Valid values: `disable` or `enable`

Examples: `stream_reassemble: enable;`

stream_reassemble.direction

Action applies to the given directions.

Type: enum

Syntax: `stream_reassemble: <direction>`

Valid values: `client`, `server`, `both`

Examples: `stream_reassemble: both;`

stream_reassemble.noalert

Don't alert when rule matches. The `stream_reassemble.noalert` parameter is optional.

Syntax: `stream_reassemble: noalert;`

Examples: `stream_reassemble: noalert;`

stream_reassemble.fastpath

Optionally trust the remainder of the session. The `stream_reassemble.fastpath` parameter is optional.

Syntax: `stream_reassemble: fastpath;`

Examples: `stream_reassemble: fastpath;`

stream_size

Detection option for stream size checking. Allows a rule to match traffic according to the number of bytes observed, as determined by the TCP sequence numbers. The `stream_size` rule option includes two parameters: `stream_size.direction` and `stream_size.range`.

Syntax: `stream_size: <server|client|both|either>, <operator><number>;`

Examples: `stream_size: client, <6;`

stream_size.direction

Comparison applies to the direction of the flow.

Type: enum

Syntax: `stream_size: <direction>;`

Valid values:

- `either`
- `to_server`
- `to_client`
- `both`

Examples: `stream_size: to_client;`

stream_size.range

Check if the stream size is within the specified range. Specify a `range` operator and one or more positive integers.

Type: interval

Syntax: `stream_size: <range_operator><positive integer>;` or `stream_size: <positive integer><range_operator><positive integer>;`

Valid values: A set of one or more postive integers, and one `range_operator` as specified in Table 3: Range Formats.

Examples: `stream_size: >6;`

Table 3: Range Formats

Range Format	Operator	Description
operator i		
	<	Less than
	>	Greater than
	=	Equal

Range Format	Operator	Description
	≠	Not equal
	≤	Less than or equal
	≥	Greater than or equal
j operator k		
	<>	Greater than j and less than k
	<=>	Greater than or equal to j and less than or equal to k