Snort 3 Inspector Reference

First Published: 2021-05-26

Last Modified: 2024-08-27

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

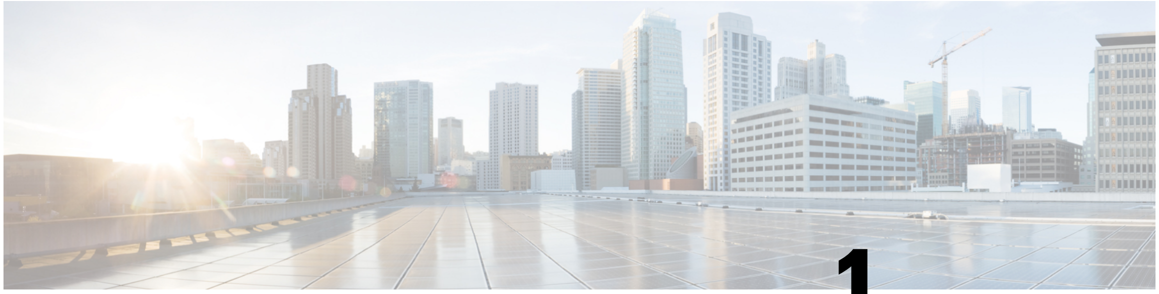
IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

CONTENTS



CHAPTER 1

Introduction

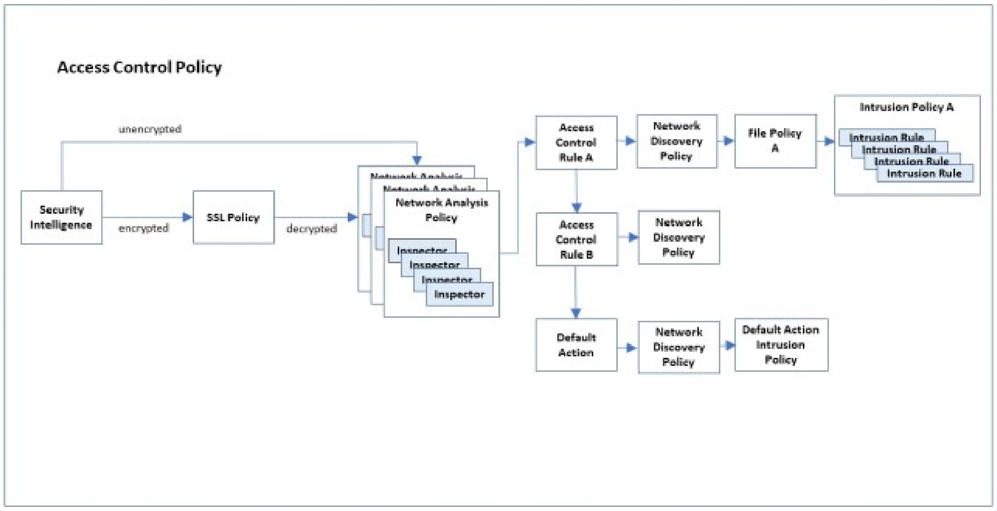
About Snort 3 Inspection

The Snort Intrusion Prevention System (IPS) analyzes network traffic in real time to provide deep packet inspection. Snort can detect and block traffic anomalies, and network probes and attacks. Snort 3 is the latest version of Snort. For more information, see https://snort.org/snort3.

Snort is designed for high performance and scalability. Snort includes a set of configurable plugins called *inspectors*. A Snort inspector can detect and analyze traffic for a certain type of network protocol or probe, normalize messages to enhance packet analysis, and inspect specific types of files embedded in a message. You configure the Snort inspectors in a Network Analysis Policy (NAP) and enable intrusion rules in an Intrusion policy.

Access Control Policies

Access control policies process traffic in several stages. The following diagram represents an example of a policy deployment. The elements addressed in this document are the Snort 3 inspectors and rule options used in intrusion rules, both highlighted in blue.



Network analysis policies enable you to configure Snort 3 inspectors to determine the traffic protocol and extract and normalize data. You can configure multiple network analysis policies, each using a uniquely configured collection of Snort 3 inspectors to normalize the data. Inspectors can alert when they detect irregularities in the data stream, but their main purpose is to prepare the data for the intrusion rules. The intrusion policies apply their configured intrusion rules to examine the data for signs of evasions, intrusions, or attacks.

Within a network analysis policy, you can customize inspection behavior for data using a given protocol by setting configuration parameters specific to the inspector that handles that protocol. For example, to configure inspection behavior for POP data, set the configuration parameters for the `pop` inspector.

You can also customize the intrusion policy for some protocols by writing custom intrusion rules using rule options specific to those protocols.

If you establish a complex configuration using multiple network analysis policies and multiple intrusion policies, the system first chooses the network analysis policy to handle the data. After the network analysis policy has applied the appropriate inspectors to perform its analysis, the data does not automatically get handed off to the corresponding intrusion policy for that protocol. The access control policy performs additional tests to determine which intrusion policy gets the data. For this reason, when configuring your access control, network analysis, and intrusion policies, ensure that data is analyzed by the correct network analysis and intrusion policy pair. For more information, see the Cisco Secure Firewall Management Center Snort 3 Configuration Guide.

Intrusion Rule Updates

Cisco periodically issues intrusion rule updates in the form of Lightweight Security Packages (LSPs). These updates may change the default values of a Snort 3 inspector's configuration parameters and intrusion rule options.

Inspector Configuration

You can enable and disable Snort inspectors as well as view and change their configurations through the Secure Firewall Management Center web interface. The Secure Firewall Management Center web interface uses the JSON format to describe the inspector configurations. For more information, see the Cisco Secure Firewall Management Center Snort 3 Configuration Guide.

To use an inspector, you must enable it through the management center web interface. In addition, for service inspectors, you must configure an entry for the service inspector in the `binder` inspector. For more information, see Binder Inspector Overview, on page 13.

The Snort 3 Inspector Reference reflects the default settings for Snort 3 inspector parameters and built-in intrusion rule options. Your system may use different defaults depending upon LSP updates, or the base network access policies provided with the system. To get the most accurate understanding of inspector settings for your network access policies, view the settings in the management center web interface.

Introduction to Snort 3 Inspectors

Snort 3 inspectors are plugins that analyze and normalize packets, similar to the Snort 2 preprocessors. The list of inspectors and settings in Snort 3 does not directly correspond to the list of preprocessors and settings in Snort 2.

Snort 3 Inspectors

- ARP Spoof Inspector, on page 11
- Binder Inspector, on page 13
- CIP Inspector, on page 19
- DCE SMB Inspector, on page 25
- DCE TCP Inspector, on page 37
- DNP3 Inspector, on page 45
- FTP Client Inspector, on page 51
- FTP Server Inspector, on page 55
- GTP Inspect Inspector, on page 63
- HTTP Inspect Inspector, on page 79
- IEC104 Inspector, on page 109
- IMAP Inspector, on page 115
- MMS Inspector, on page 119
- Modbus Inspector, on page 123
- Normalizer Inspector, on page 127
- POP Inspector, on page 135
- Port Scan Inspector, on page 139
- Rate Filter, on page 155
- S7CommPlus Inspector, on page 159
- SIP Inspector, on page 163
- SMTP Inspector, on page 171

For each Snort 3 inspector, this document describes:

- General information about the purpose and function of the inspector.
- The type of inspector:
    - Service: Inspectors that analyze protocol data units (PDUs) used in internet service protocols (HTTP, FTP, TCP, or UDP). Examples include: `http_inspect`, `ftp_server`.
    - Passive: Inspectors that provide only configuration (`ftp_client`, `ftp_server`) or facilitate other processing (`binder`).
    - Packet: Inspectors that perform processing on raw packets before other inspectors do their processing. Examples include: `normalizer`.
    - Probe: Inspectors that perform processing on all packets after all detection has completed. Examples include: `port_scan`.
    - Stream: Inspectors that perform flow tracking, internet protocol defragmentation, and TCP reassembly. Examples include: `stream_tcp`, `stream_ip`.
    - Basic module: A configurable, built-in Snort 3 component which provides functionality to support the inspection process for multiple types of traffic. Examples include: `rate_filter`.
- Usage:
    - Inspect: Configure these inspectors within a network analysis policy (NAP). Examples include: `imap`, `ssh`.
    - Global, Context: Configure these inspectors once. Examples include: `port_scan, rate_filter`.
- Instance Type:
    - Singleton: Configure these inspectors for a single instance within a network access policy. For more details, see Singleton Inspectors, on page 5.
    - Multiton: Configure these inspectors for multiple instances within a network access policy (NAP). A NAP can contain multiple instances differentiated by network, port, or VLAN. Each instance is uniquely configured to process a specific traffic segment. For more details, see Multiton Inspectors, on page 6.
- Other inspectors required: Many inspectors depend upon other inspectors to fully process the data stream. When an inspector requires that you configure other inspectors, the documentation identifies those additional inspectors.

- Best practices for configuring the inspector: These are recommendations for optimal performance specific to each inspector.
- Configuration parameters for the inspector: You can set configuration parameters in the management center web interface under **Policies>Access Control>Network Analysis Policy>*Policy Name*>Snort 3 Version>*Inspector Name***.

Note Before modifying inspector parameters, we recommend that you understand the interaction between the inspector and enabled intrusion rules.

- Rules: The Snort 3 inspectors use rules to generate events. The built-in rules may contain classtype, references, and other metadata.
- Intrusion rule options: Customize intrusion rules by defining intrusion rule options for the data type handled by the inspector. See the Cisco Secure Firewall Management Center Snort 3 Configuration Guide for information on managing custom intrusion rules.

Note Writing custom intrusion rules is an advanced activity and must be undertaken with care. You may need to use inspectors and rule options not described in this documentation. Using some of the inspectors and intrusion rule options described in this document require specific settings in inspectors and rule options documented in the Snort open-source documentation. Some rule options have an impact on the Snort fast pattern matcher or placement of the detection cursor. For more information, see the Snort 3 open-source documentation, available at https://www.snort.org/snort3.

Singleton Inspectors

A network access policy (NAP) can use only one instance of a singleton inspector.

- A singleton inspector does not support multiple instances per NAP like multiton inspectors.
- A singleton inspector may not apply to some specific flows.

For example:

```
{
   "normalizer":{
      "enabled":true,
      "type":"singleton",
      "data":{
         "ip4":{
            "df":true
         }
      }
   }
}
```

Multiton Inspectors

A network access policy may use one or more instances of multiton inspectors, which you can configure as needed. A multiton inspector supports configuring settings based on specific conditions, including network, port, and VLAN. One set of supported settings comprises an instance. A multiton provides a default instance, and you can define additional instances based on specific conditions. If the traffic matches the conditions in an customized instance, the settings from that instance are applied. Otherwise, the settings from the default instance are applied. The name of the default instance is the same as the inspector's name.

For a multiton inspector, when you upload the overridden inspector configuration, you also need to define a matching `binder` configuration for each instance in the JSON file, otherwise, the upload results in an error. You can also create new instances, but make sure that you include a `binder` condition for every new instance that you create to avoid errors.

For example:

- Multiton inspector where the default instance is modified:

```
{
   "http_inspect":{
      "instances":[
         {
            "name":"http_inspect",
            "data":{
               "response_depth":5000
            }
         }
      ]
   }
}
```

- Multiton inspector where the default instance and default `binder` is modified:

```
{
   "http_inspect":{
      "instances":[
         {
            "name":"http_inspect",
            "data":{
               "response_depth":5000
            }
         }
      ]
   },
   "binder":{
      "rules":[
         {
            "use":{
               "type":"http_inspect"
            },
            "when":{
               "role":"any",
               "ports":"8080",
               "proto":"tcp",
               "service":"http"
            }
         }
      ]
   }
}
```

- Multiton inspector where a custom instance and a custom `binder` is added:

```
{
   "http_inspect":{
      "instances":[
         {
            "name":"http_inspect1",
            "data":{
               "response_depth":5000
            }
         }
      ]
   },
   "binder":{
      "rules":[
         {
            "use":{
               "type":"http_inspect",
               "name":"http_inspect1"
            },
            "when":{
               "role":"any",
               "ports":"8080",
               "proto":"tcp",
               "service":"http"
            }
         }
      ]
   }
}
```

Protocol and Service Identification in Snort 3

The `binder` inspector performs a unique function that affects all Snort service inspectors. Along with the Snort `wizard` module, the `binder` determines which stream or service inspector can inspect the network traffic. The configurations in the `binder` inspector include the ports, hosts, CIDRs, and services that define when another inspector in the same network analysis policy needs to inspect traffic.

The `wizard` supports port-independent configuration of services which allows for the detection of malware command and control channels.

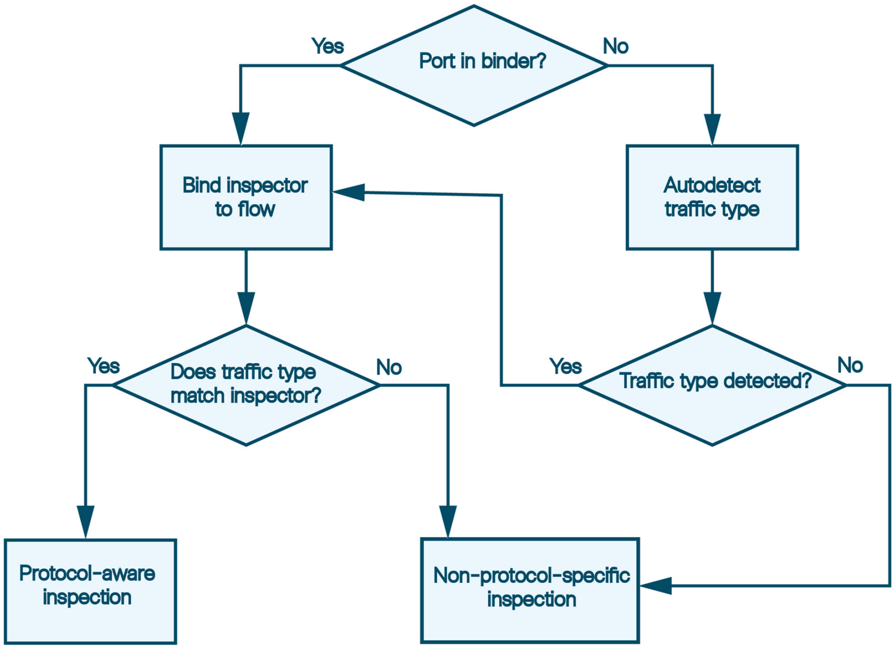
Note You cannot configure the `wizard` through the Secure Firewall Management Center web interface.

When traffic arrives at a firewall device, the `binder` inspector searches for intrusion policies and selects the appropriate network access policy (NAP) to apply. Within a NAP, the `binder` determines the appropriate stream and service inspectors to use for the data flow. Later, if the service associated with a flow changes, the NAP uses the `binder` to select a different service inspector.

The `binder` inspector configuration includes `when` parameters that describe traffic characteristics, and `use` parameters that specify which inspector to apply to traffic matching those characteristics. When determining which inspector to apply to a data flow, the `binder` inspector compares traffic against its `when` clauses in order, from the top down, and applies the `use` clause that corresponds to the first `when` clause that matches the traffic.

If no specific `binder` criteria match a data flow, the `wizard` analyzes the data flow to determine the service. The `wizard` invokes the `binder` to select the appropriate inspector for that service. If no service can be identified, the `binder` typically binds a stream inspector to the flow, and the system performs non-protocol-specific reassembly of the data packets without regard to payload content.

The following diagram illustrates how inspectors perform protocol-specific or non-protocol-specific inspection. Service inspection depends on how you configure `port`, `host`, `service`, and `CIDR` parameters in the `binder` inspector:



You can customize the inspector selection criteria by defining the `use` and `when` parameters in the `binder` inspector for a NAP in the management center web interface. For more information on the `binder` parameters, see Binder Inspector Overview, on page 13. For information on navigating the management center web interface to configure inspectors, see the Cisco Secure Firewall Management Center Snort 3 Configuration Guide.

If you configure the `binder` incorrectly, it cannot detect the service for the flow or bind an inspector to it. If the rules engine and autodetection cannot understand and identify the traffic, configuring a `when` criteria such as the port in the `binder` inspector does not force inspection. For example, if you configure port 88 in the `binder` as an HTTP port, the `binder` binds the `http_inspect` inspector to any flow on that port. But if the flow is not HTTP, the rules engine does not inspect the data as HTTP, but instead performs port-based detection.

Autodetection and Enabled or Disabled Inspectors in the Network Analysis Policy

The behavior of autodetection changes, depending upon whether the targeted inspector is enabled or disabled in the network analysis policy. If the targeted inspector is enabled in the network analysis policy, autodetection works as described above.

If the targeted inspector is disabled in the network analysis policy, typically, autodetection still binds a stream inspector, such as stream TCP or stream UDP, to the flow. However, the rules engine does not perform service inspection or detection. For a TCP flow, the stream TCP inspector performs reassembly.

PART I

Snort 3 Inspectors

CHAPTER 2

ARP Spoof Inspector

ARP Spoof Inspector Overview

Type	Inspector (network)
Usage	Inspect
Instance Type	Singleton
Other Inspectors Required	None
Enabled	`true`

Address Resolution Protocol (ARP) is a stateless, communication protocol used within a single network for address resolution. When exchanging requests and responses, ARP does not provide authentication between hosts.

ARP spoof is a type of man-in-the-middle attack using ARP within a Local Area Network (LAN). An attacker alters the communication to a host by intercepting messages intended for a specific host media access control (MAC) address.

The `arp_spoof` inspector analyzes ARP packets and detects unicast ARP requests. To detect ARP cache overwrite attacks, the ARP Spoof inspector identifies inconsistent Ethernet-to-IP mapping.

If enabled, the `arp_spoof` inspector:

- Inspects Ethernet addresses and the addresses in the ARP packets. When an inconsistency occurs, the inspector uses rule 112:2 or rule 112:3 to generate alerts, and in an inline deployment, drop offending packets.
- Checks for unicast ARP requests. If a unicast ARP request is detected, the inspector uses rule 112:1 to generate alerts, and in an inline deployment, drop offending packets.

- If the `hosts[]` parameter is specified, the inspector uses that information to detect ARP cache overwrite attacks. If such an attack is detected, the inspector uses rule 112:4 to generate alerts, and in an inline deployment, drop offending packets.

ARP Spoof Inspector Parameters

The `arp_spoof` inspector does not provide default configuration parameter values in the Secure Firewall Management Center web interface.

ARP Spoof Inspector Rules

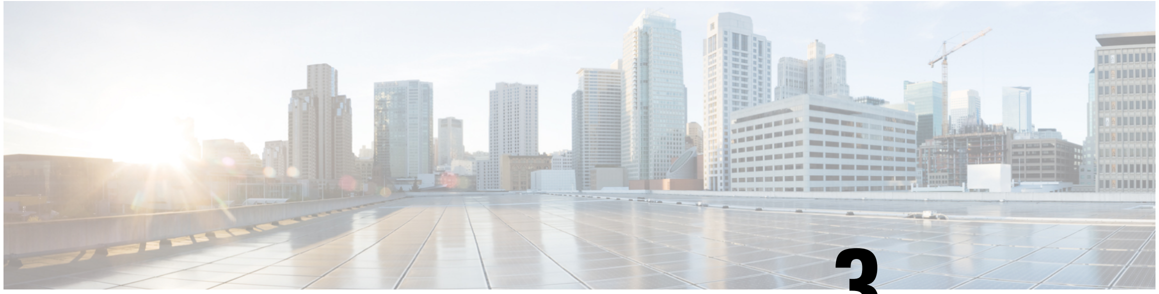
Enable the `arp_spoof` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 1: ARP Spoof Inspector Rules

GID:SID	Rule Message
112:1	unicast ARP request
112:2	ethernet/ARP mismatch request for source
112:3	ethernet/ARP mismatch request for destination
112:4	attempted ARP cache overwrite attack

ARP Spoof Inspector Intrusion Rule Options

The `arp_spoof` inspector does not have any intrusion rule options.



CHAPTER 3

Binder Inspector

Binder Inspector Overview

Type	Inspector (passive)
Usage	Inspect
Instance Type	Singleton
Other Inspectors Required	Depends upon bindings established
Enabled	`true`

Each Network Analysis Policy (NAP) has one `binder` inspector. The `binder` determines when to use a certain service inspector to inspect traffic. The configurations in the `binder` inspector include the ports, hosts, CIDRs, and services that define when another inspector in the same NAP needs to inspect traffic. When a `binder` rule matches a new flow, the targeted inspector is bound to the flow.

The `binder` inspector can work with the autodetection `wizard` to perform port-independent configuration of services and detection of malware command and control channels. For more information, see Protocol and Service Identification in Snort 3, on page 7.

Bindings are evaluated when a session starts and then again if and when an appropriate service is identified in the session. The bindings are a list of when-use rules evaluated from top to bottom. Snort uses the first matching network and service configurations to inspect traffic.

Example

For example, if you want to configure a NAP to inspect CIP traffic:

- In the `binder` inspector for the NAP, update the `"type":"cip"` section with the correct ports, role, and protocol information for the traffic that you want to inspect.
- Review the default values in the `cip` inspector for that same NAP and make any adjustments required to inspect the CIP traffic.

The following is an example of the `cip` configuration and binding. This example uses options described in Binder Inspector Parameters, on page 16.

```
{
   "use": {
     "type":"cip"
   },
   "when": {
     "proto":"udp",
     "ports":"22222 33333",
     "role":"server"
   }
 },
 {
   "use": {
     "type":"cip"
   },
   "when": {
     "role":"server",
     "ports":"44818",
     "proto":"tcp"
   }
 },
```

Autodetection of Services for Portless Configuration

The autodetection `wizard` enables port-independent configuration of services and the detection of malware command and control channels. When traffic arrives, the `binder` inspector attaches the autodetection `wizard` to the flow at the outset and it checks the initial payload to determine the service the traffic is using. For example, `GET` would indicate HTTP and `HELO` would indicate SMTP. After the service is determined, Snort bounds the the appropriate service inspector to the flow and detaches the autodetection `wizard` from the flow.

Note You cannot configure the autodetection `wizard` through the Secure Firewall Management Center web interface.

If the rules engine and autodetection `wizard` cannot understand and identify the traffic, configuring a port in the `binder` inspector does not force inspection.

Autodetection and Binder Configuration

The `binder` inspector matches intrusion rules in order, from the top down, and applies the first rule to match the traffic. If you haven't configured the `binder` inspector for the service detected in the flow, the autodetection wizard can still bind the flow to the relevant inspector. For example:

- If the payload is `GET` and the autodetection wizard identifies the traffic type as HTTP, the `binder` inspector binds the HTTP inspector to that flow.
- If the traffic type cannot be identified, the rules engine performs a non-protocol specific inspection.

If you configure a port incorrectly, the `binder` inspector cannot autodetect the service for that flow nor can it bind an inspector to it. For example, if you configure port 88 into the binder as an HTTP port, the `binder` inspector will bind the HTTP inspector to any flow on that port. However, if the flow is not HTTP, the rules engine will not inspect it as HTTP. Instead, the inspection and detection will timeout.

Autodetection and Enable or Disable of Inspectors in the Network Analysis Policy

The behavior of autodetection changes, depending upon whether the targeted inspector is enabled or disabled in the network analysis policy. If the targeted inspector is enabled in the network analysis policy, autodetection works as expected.

If the targeted inspector is disabled in the network analysis policy, typically, autodetection still binds a stream inspector, such as stream TCP or stream UDP, to the flow. However, the rules engine does not perform service inspection or detection. For a TCP flow, the stream TCP inspector performs reassembly.

Best Practices for Configuring the Binder Inspector

Consider the following best practices when you configure the binder inspector:

- Do not configure ports in the binder inspector unless it's required for that inspector. The port configuration does not improve efficacy if the rules engine can autodetect the traffic. However, an incorrect port configuration can lead to failure to detect evasions.
- Configure a port for only one inspector. If a port is configured twice in the binder for different protocols and inspectors, it will automatically trigger the first inspector.
- Add the configuration for a service inspector to the `binder` inspector if you do not see it in the default `binder` inspector configuration. For example, if you want to use the `cip` inspector, add the `use` and `when` options for the `cip` inspector to the binder.
- For the stream TCP inspector, configure networks to custom bind operating system configurations. The network configurations apply to all ports.
- For service inspectors, avoid hard port bindings if the binder can autodetect the protocol in the flow. If the protocol is not detectable, a hard port binding does not ensure detection and inspection.

Inspectors that Require Port Configuration

Configure ports in the binder inspector for the following inspectors, because autodetection does not work for the related protocols:

- `cip`
- `gtp_inspect`
- `iec104`
- `modbus`
- `s7commplus`

Inspectors that Do Not Require Port Configuration

Do not configure ports in the binder inspector for the following inspectors, because autodetection does work for the related protocols:

- `arp_spoof`
- `dce_smb`
- `dce_tcp`
- `dnp3`
- `ftp_client`
- `ftp_server`
- `http_inspect`
- `imap`
- `normalizer`
- `pop`
- `port_scan`
- `sip`
- `smtp`
- `ssh`
- `stream_icmp`
- `stream_ip`
- `stream_tcp`
- `stream_udp`
- `telnet`

Binder Inspector Parameters

binder[]

A binder includes an array of rules defined as a pair of `when` and `use` objects.

Type: array

Example:

```
{
    binder: {
        rules: [
            {
                "when": {
                    ...
                },
                "use": {
                    ...
                 }
            },
            {
                "when": {
                    ...
                },
                "use": {
                    ...
                }
            }
        ]
    }
}
```

binder[].use.type

Specifies the inspector to bind to the data flow when the criteria in the `when` parameter matches. For example, to inspect CIP traffic, add `use.type` with a value of `cip`.

Type: string

Valid values: The name of any Snort 3 inspector described in this document.

Default value: The `binder` inspector includes a `use.type` parameter for each supported inspector.

binder[].when.proto

Specifies the protocol that the traffic must match to bind the data flow to the inspector specified in `use.type`. For example, if the network analysis policy is configured to inspect TCP traffic, the `binder` inspector must have this parameter set to `tcp`.

Type: enum

Valid values: `any`, `ip`, `icmp`, `tcp`, `udp`, `user`, `file`

Default value: The `binder` inspector includes a `when.proto` parameter for each protocol.

binder[].when.ports

Specifies the ports that the traffic must match to bind the data flow to the inspector specified in `use.type`. For example, to inspect traffic on TCP port 80, set `when.proto` to `tcp` and `when.ports` to `80`.

Specify a list of one or more ports represented as decimal or hex integers. Separate multiple ports with a space and enclose the list with double quotes.

Type: string

Valid range: `1 - 65535`

Default value: `65535` (This value may vary depending upon the value of `when.proto`.)

binder[].when.role

Specifies the roles that the traffic must match to bind the flow to the inspector specified in `use.type`.

Type: enum

Valid values: `client`, `server`, `any`

Default value: `any`

Specifies the service that the traffic must match to bind the flow to the inspector specified in `use.type`.

Type: string

Valid values: A name of a service that may encapsulate incoming data, for example: `netbios-ssn` or `dcerpc`.

Default value: None

Binder Inspector Rules

The `binder` inspector does not have any associated rules.

Binder Inspector Intrusion Rule Options

The `binder` inspector does not have any intrusion rule options.

CHAPTER 4

CIP Inspector

CIP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

The Common Industrial Protocol (CIP) is an application protocol that supports industrial automation applications. EtherNet/IP (ENIP) is an implementation of CIP that is used on Ethernet-based networks.

The `cip` inspector detects CIP and ENIP traffic running on TCP or UDP and sends it to the intrusion rules engine. You can use CIP and ENIP keywords in custom intrusion rules to detect attacks in CIP and ENIP traffic.

Note In Snort 3, the `cip` inspector does not support CIP application detectors. To implement CIP application detection, you can create and import custom CIP intrusion rules and enable the appropriate IPS rules. For more information, see the Snort 3 configuration documentation for your management application.

Best Practices for Configuring the CIP Inspector

Consider the following best practices when configuring the `cip` inspector:

- You must add the default CIP detection port 44818 and any other CIP ports in the `binder` inspector.
- We recommend that you use an intrusion prevention action as the default action for your access control policy.
- To detect CIP and ENIP applications, you must enable the `cip` inspector in the corresponding custom network analysis policy.
- To block CIP or ENIP application traffic using access control rules, ensure that the normalizer inspector and its inline mode option are enabled (the default setting) in the corresponding network analysis policy.
- To drop traffic that triggers `cip` inspector rules and CIP intrusion rules, ensure that **Drop when Inline** is enabled in the corresponding intrusion policy.
- The `cip` inspector does not support an access control policy default action of either of the following:
  - **Access Control: Trust All Traffic**
  - **Access Control: Block All Traffic**
- The `cip` inspector does not support application visibility for CIP applications, including network discovery.

CIP Inspector Parameters

CIP TCP port configuration

The `binder` inspector defines the CIP TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
                    "role": "server",
                    "proto": "tcp",
                    "ports": "44818"
        },
        "use": {
                    "type": "cip"
        }
    }
]
```

embedded_cip_path

Determines whether the inspector checks the embedded CIP connection path.

Type: string

Valid values:

- `"false"`
- CIP path enclosed in double quotation marks, for example, `"0x2 0x36"`.

Default value: `"false"`

unconnected_timeout

Sets the default unconnected timeout in seconds. When a CIP request message does not contain a protocol-specific timeout value and the maximum number of concurrent unconnected requests per TCP connection is reached, the system times the message for the number of seconds specified by this parameter. When the timer expires, the message is removed to make room for future requests.

When you specify 0, all traffic that does not have a protocol-specific timeout configured times out first.

Type: integer

Valid range: `0` to `360`

Default value: `300`

max_unconnected_messages

Sets the maximum number of concurrent unconnected CIP messages per TCP connection. If the system reaches the maximum number of concurrent requests that can go unanswered, the system closes the connection.

Type: integer

Valid range: `1` to `10000`

Default value: `100`

max_cip_connections

Sets the maximum number of simultaneous CIP connections allowed by the system per TCP connection.

Type: integer

Valid range: `1` to `10000`

Default value: `100`

CIP Inspector Rules

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

Table 2: CIP Inspector Rules

GID:SID	Rule Message
148:1	CIP data is malformed
148:2	CIP data is non-conforming to ODVA standard
148:3	CIP connection limit exceeded. Least recently used connection removed
148:4	CIP unconnected request limit exceeded. Oldest request removed

CIP Inspector Intrusion Rule Options

cip_attribute

Detection parameter to match the CIP attribute.

Type: interval

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

Valid values: A set of one or more integers between 0 and 65535, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_attribute: <100;`

cip_class

Detection parameter to match the CIP class.

Type: interval

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

Valid values: A set of one or more integers between 0 and 65535, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_class: <25;`

cip_conn_path_class

Detection parameter to match the CIP connection path class.

Type: interval

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

Valid values: A set of one or more integers between 0 and 65535, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_conn_path_class: <85;`

cip_instance

Detection parameter to match the CIP instance.

Type: interval

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

Valid values: A set of one or more integers between 0 and 65535, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_instance: <15;`

cip_req

Detection parameter to match the CIP request.

Syntax: `cip_req;`

Examples: `cip_req;`

cip_rsp

Detection parameter to match the CIP response.

Syntax: `cip_rsp;`

Examples: `cip_rsp;`

cip_service

Detection parameter to match the CIP service.

Type: interval

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

Valid values: A set of one or more integers between 0 and 127, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_service: <50;`

cip_status

Detection parameter to match the CIP response status.

Type: interval

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

Valid values: A set of one or more integers between 0 and 255, and a `range_operator` as specified in the Table 3: Range Formats.

Examples: `cip_status: <250;`

Table 3: Range Formats

Range Format	Operator	Description
operator i		
	<	Less than
	>	Greater than
	=	Equal
	≠	Not equal
	≤	Less than or equal

Range Format	Operator	Description
	≥	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

CHAPTER 5

DCE SMB Inspector

DCE SMB Inspector Overview

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

The DCE/RPC protocol allows processes on separate network hosts to communicate as if the processes were on the same host. These inter-process communications are commonly transported between hosts over TCP and UDP. Within the TCP transport, DCE/RPC might also be further encapsulated in the Windows Server Message Block (SMB) protocol or in Samba, an open-source SMB implementation used for inter-process communication in a mixed environment comprised of Windows, and UNIX or Linux operating systems.

Although most DCE/RPC exploits occur in DCE/RPC client requests targeted for DCE/RPC servers, which could be practically any host on your network that is running Windows or Samba, exploits can also occur in server responses.

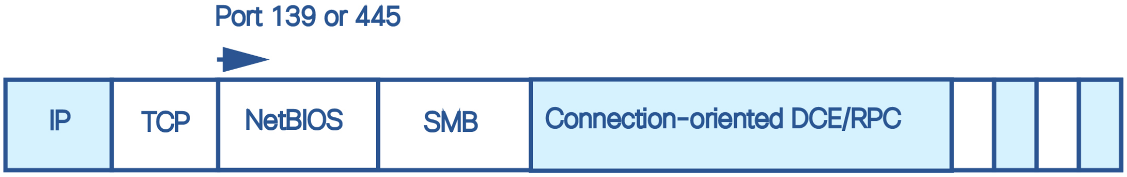
IP encapsulates all DCE/RPC transports. TCP transports all connection-oriented DCE/RPC, such as SMB.

The `dce_smb` inspector detects connection-oriented DCE/RPC in the SMB protocol and uses protocol-specific characteristics including header length and data fragment order to:

- Detect DCE/RPC requests and responses encapsulated in SMB transports.
- Analyze DCE/RPC data streams and detect anomalous behavior and evasion techniques in DCE/RPC traffic.
- Analyze SMB data streams and detect anomalous SMB behavior and evasion techniques.

- Desegment SMB and defragment DCE/RPC.
- Normalize DCE/RPC traffic for processing by the rules engine.

The following diagram illustrates the point at which the DCE SMB inspector begins processing traffic for the SMB transport.



The dce_smb inspector typically receives SMB traffic on the well-known TCP port 139 for the NetBIOS Session Service or the similarly implemented well-known Windows port 445. Because SMB has many functions other than transporting DCE/RPC, the inspector first tests whether the SMB traffic is carrying DCE/RPC traffic and stops processing if it is not, or continues processing if it is.

Descriptions of the dce_smb inspector parameters and functionality include the Microsoft implementation of DCE/RPC known as Microsoft Remote Procedure Call (MSRPC), as well as both SMB and Samba.

Target-Based Policies

Windows and Samba DCE/RPC implementations differ significantly. For example, all versions of Windows use the DCE/RPC context ID in the first fragment when defragmenting DCE/RPC traffic, and all versions of Samba use the context ID in the last fragment. As another example, Windows Vista uses the opnum (operation number) header field in the first fragment to identify a specific function call, and Samba and all other Windows versions use the opnum field in the last fragment.

There are significant differences in Windows and Samba SMB implementations. For example, Windows recognizes the SMB OPEN and READ commands when working with named pipes, but Samba does not recognize these commands.

For this reason, the dce_smb inspector uses a target-based approach. When you configure a dce_smb inspector instance, the policy parameter specifies an implementation of the DCE/RPC SMB protocol. This in combination with the host information establishes a default target-based server policy. Optionally, you can configure additional inspectors that target other hosts and DCE/RPC SMB implementations. The DCE/RPC SMB implementation specified by the default target-based server policy applies to any host not targeted by another dce_smb inspector instance.

DCE/RPC SMB implementations which the dce_smb inspector can target with the policy parameter are:

- WinXP (default)
- Win2000
- WinVista
- Win2003
- Win2008
- Win7
- Samba
- Samba-3.0.37

- Samba-3.0.22
- Samba-3.0.20

File Inspection

The dce_smb inspector supports file inspection for SMB versions 1, 2, and 3.

The dce_smb inspector examines normal SMB file transfers. This includes checks of the file type and signature through the file processing as well as setting a pointer for the file_data rule option. The dce_smb inspector supports inspection of normal SMB file transfers for SMB version 1, 2, and 3 when used in coordination with the file_id inspector (described in Snort 3 open source documentation, available at https://www.snort.org/snort3). To enable file inspection, configure the file_id inspector as needed, and set the dce_smb smb_file_inspection and smb_file_depth parameters. The smb_file_depth parameter indicates the number of file data bytes the file_id inspector examines beginning at the pointer indicated by the file_data IPS rule option. For more information, see the Snort 3 open source documentation, available at https://www.snort.org/snort3).

Defragmentation

The dce_smb inspector supports reassembling fragmented data packets. This feature is useful in inline mode to catch exploits early in the inspection process before full defragmentation is done, or to catch exploits that take advantage of fragmentation to conceal themselves. Be aware that disabling defragmentation may result in a large number of false negatives.

DCE SMB Inspector Parameters

DCE SMB port configuration

The binder inspector defines the DCE SMB port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
   {
       "when": {
           "role":"any",
           "service":"netbios-ssn",
           "ports": ""
       },
        "use": {
            "type":"dce_smb"
       }
   }
]
```

max_frag_len

Specifies the maximum fragment length in bytes allowed for defragmentation. For processing larger fragments the inspector considers packet content to this size before defragmenting.

Note The value specified in this parameter must be greater than or equal to the depth to which the rules need to examine the data to ensure detection. To ensure that all data goes through detection, use the default value.

Type: integer

Valid range: `1514` to `65535`

Default value: `65535`

smb_max_compound

Specifies the maximum number of commands to process in one SMB request.

Type: integer

Valid range: `0` to `255`

Default value: `3`

smb_max_chain

Specifies the maximum number of chained SMB AndX commands allowed. Typically, more than a few chained AndX commands represent anomalous behavior and could indicate an evasion attempt. Specify `1` to permit no chained commands or `0` to disable detecting the number of chained commands.

The `dce_smb` inspector first counts the number of chained commands and generates an event if accompanying SMB inspector rules are enabled and the number of chained commands equals or exceeds the configured value. It then continues processing.

You can enable rule 133:20 to generate events and, in an inline deployment, drop offending packets for this parameter.

Type: integer

Valid range: `0` to `255`

Default value: `3`

disable_defrag

Specifies whether to defragment fragmented DCE/RPC traffic. When enabled, the `dce_smb` inspector detects anomalies and sends DCE/RPC data to the rules engine, but at the risk of missing exploits in fragmented DCE/RPC data.

Although `disable_defrag` provides the flexibility of not defragmenting traffic and can speed processing, most DCE/RPC exploits attempt to take advantage of fragmentation to hide the exploit. Enabling this parameter would bypass most known exploits, resulting in a large number of false negatives.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

limit_alerts

Specifies whether to limit the DCE alerts to at most one per signature per flow.

Type: boolean

Valid values: `true`, `false`

Default value: `true`

reassemble_threshold

Specifies the minimum number of bytes in the DCE/RPC desegmentation and defragmentation buffers to queue before sending a reassembled packet to the rules engine. This parameter is useful in inline mode so as to potentially catch an exploit early before full defragmentation is done.

Note that a low value increases the likelihood of early detection but could have a negative impact on performance. You should test for performance impact if you enable this parameter.

A value of `0` disables reassembly.

Type: integer

Valid range: `0` to `65535`

Default value: `0`

policy

Specifies the Windows or Samba DCE/RPC implementation used by the targeted host or hosts on your monitored network segment.

Type: enum

Valid values: A string selected from the following list: `Win2000`, `WinXP`, `WinVista`, `Win2003`, `Win2008`, `Win7`, `Samba`, `Samba-3.0.37`, `Samba-3.0.22`, `Samba-3.0.20`

Default value: `WinXP`

smb_max_credit

Specifies the maximum number of outstanding requests.

Type: integer

Valid range: `1` to `65536`

Default value: `8192`

smb_file_depth

Specifies the number of bytes inspected when a file is detected in SMB traffic, beginning at the location specified by the `file_data` IPS rule option (described in Snort 3 open source documentation, available at https://www.snort.org/snort3).

Specify `-1` to disable file inspection.

Specify `0` to indicate unlimited file inspection.

Type: integer

Valid range: `-1` to `32767`

Default value: `16384`

When a file is detected in SMB traffic, the `smb_file_depth` parameter indicates the number of file data bytes the inspector will examine beginning at the pointer set in the `file_data` IPS rule option. To inspect the file type and signature, `dce_smb` uses the `enable_type`, `type_depth`, `enable_signature`, and `signature_depth` parameters set in the `file_id` inspector. For more information on the `file_id` inspector, see the Snort open source documentation, available at https://www.snort.org/snort3.

memcap

Specifies the maximum memory limit in bytes allocated to the inspector. When the maximum memory cap is reached or exceeded, the `dce_smb` inspector deletes the least recently used data to create more space.

Type: integer

Valid range: `512` to `9,007,199,254,740,992 (maxSZ)`

Default value: `8,388,608`

smb_fingerprint_policy

Causes the inspector to detect the Windows or Samba version that is identified in SMB `Session Setup AndX` requests and responses. When the detected version is different from the Windows or Samba version configured for the `policy` inspector parameter, the detected version overrides the configured version for that session only.

For example, if you set `policy` to Windows XP and the inspector detects Windows Vista, the inspector uses a Windows Vista policy for that session. Other settings remain in effect.

Type: enum

Valid values: `none`, `client`, `server`, or `both`

- Use `client` to inspect server-to-client traffic for the policy type.
- Use `server` to inspect client-to-server traffic for the policy type.
- Use `both` to inspect server-to-client and client-to-server traffic for the policy type.
- Use `none` to disable Windows or Samba version inspection.

Default value: `none`

smb_legacy_mode

When `smb_legacy_mode` is `true`**,** the system applies SMB intrusion rules only to SMB Version 1 traffic, and applies DCE/RPC intrusion rules to DCE/RPC traffic using SMB Version 1 as a transport.

When `smb_legacy_mode` is `false`, the system applies SMB intrusion rules to traffic using SMB Versions 1, and 2, and:

- For Versions 7.0 and 7.0.x the system applies DCE/RPC intrusion rules to DCE/RPC traffic using SMB as a transport only for SMB Version 1.
- For Versions 7.1+ the system applies DCE/RPC intrusion rules to DCE/RPC traffic using SMB as a transport for SMB Versions 1 and 2.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

valid_smb_versions

Specifies which SMB versions to inspect. Separate multiple SMB versions with a space character.

Type: string

Valid values: `v1`, `v2`, `v3`, `all`

Default value: `all`

DCE SMB Inspector Rules

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

Table 4: DCE SMB Inspector Rules

GID:SID	Rule Message
133:2	SMB - bad NetBIOS session service session type
133:3	SMB - bad SMB message type
133:4	SMB - bad SMB Id (not \xffSMB for SMB1 or not \xfeSMB for SMB2)
133:5	SMB - bad word count or structure size
133:6	SMB - bad byte count
133:7	SMB - bad format type
133:8	SMB - bad offset
133:9	SMB - zero total data count
133:10	SMB - NetBIOS data length less than SMB header length
133:11	SMB - remaining NetBIOS data length less than command length
133:12	SMB - remaining NetBIOS data length less than command byte count
133:13	SMB - remaining NetBIOS data length less than command data size
133:14	SMB - remaining total data count less than this command data size
133:15	SMB - total data sent (STDu64) greater than command total data expected
133:16	SMB - byte count less than command data size (STDu64)
133:17	SMB - invalid command data size for byte count
133:18	SMB - excessive tree connect requests with pending tree connect responses
133:19	SMB - excessive read requests with pending read responses
133:20	SMB - excessive command chaining

GID:SID	Rule Message
133:21	SMB - multiple chained login requests
133:22	SMB - multiple chained tree connect requests
133:23	SMB - chained/compounded login followed by logoff
133:24	SMB - chained/compounded tree connect followed by tree disconnect
133:25	SMB - chained/compounded open pipe followed by close pipe
133:26	SMB - invalid share access
133:44	SMB - invalid SMB version 1 seen
133:45	SMB - invalid SMB version 2 seen
133:46	SMB - invalid user, tree connect, file binding
133:47	SMB - excessive command compounding
133:48	SMB - zero data count
133:50	SMB - maximum number of outstanding requests exceeded
133:51	SMB - outstanding requests with same MID
133:52	SMB - deprecated dialect negotiated
133:53	SMB - deprecated command used
133:54	SMB - unusual command used
133:55	SMB - invalid setup count for command
133:56	SMB - client attempted multiple dialect negotiations on session
133:57	SMB - client attempted to create or set a file’s attributes to readonly/hidden/system
133:58	SMB - file offset provided is greater than file size specified
133:59	SMB - next command specified in SMB2 header is beyond payload boundary

DCE Inspectors Intrusion Rule Options

dce_iface

Specifies the following comma-separated elements:

- The UUID for a service interface.
- The interface version (optional). The default setting matches any version.

- An indicator of whether a rule should match on any fragment in a request (optional). The default setting matches only the first fragment.

In the DCE/RPC protocol, a client must bind to a service before making a call to it. When a client sends a bind request to the server, it can specify one or more service interfaces to bind to. Each interface is represented by a UUID, and each interface UUID is paired with a unique index (or context ID) that future requests can use to reference the service that the client is calling. The server responds with the interface UUIDs it accepts as valid and allows the client to make requests to those services. When a client makes a request, it will specify the context ID so the server knows to what service the client is making a request.

Using the `dce_iface` rule option, a rule can ask the inspector whether the client has bound to a specific interface UUID and whether this client request is making a request to that interface. This can eliminate false positives where more than one service is bound to successfully since the inspector can correlate the bind UUID to the context id used in the request.

The `dce_iface` option requires tracking client Bind and Alter Context requests as well as server Bind Ack and Alter Context responses for connection-oriented DCE/RPC in the inspector. For each Bind and Alter Context request, the client specifies a list of interface UUIDs along with a handle (or context id) for each interface UUID that will be used during the DCE/RPC session to reference the interface. The server response indicates which interfaces it allows the client to make requests to—it either accepts or rejects the client's wish to bind to a certain interface. This tracking is required so that when a request is processed, the context id used in the request can be correlated with the interface UUID for which it is a handle.

The `dce_iface` rule option matches if:

- the specified interface UUID matches the interface UUID (as referred to by the context ID) of the DCE/RPC request

and

- the `version` argument is not supplied, or the `version` argument is supplied and matches the interface UUID of the DCE/RPC request

and

- the `any_frag` argument is supplied, or the `any_frag` argument is absent and the `dce_iface` option matches the UUID and version criteria on the initial request fragment

Examples:

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, <2;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188,any_frag;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, =1, any_frag;
```

dce_iface.uuid

A DCE/RPC request can specify whether UUID numbers are represented as big endian or little endian. The representation of the interface UUID in a request is different depending on the endianness specified in the request. The `dce_rpc` inspector normalizes the UUID. This means that the UUID specification in the `dce_iface` rule option must be written the same way regardless of the endianness of the request.

For example, a little endian Bind request would represent a UUID as follows:

```
|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|
```

and a big-endian Bind request would represent the same UUID as follows:

```
|5a 7b 91 f8 ff 00 11 d0 a9 b2 00 c0 4f b6 e6 fc|
```

In Snort 3 rules using the `dce_iface` option, the UUID must be represented in a string using big endian order regardless of the endian-ness of the request:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

Type: string

Syntax: `dce_iface: <UUID>;`

Valid values: Where: *`UUID`* is 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form: `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

Examples: `dce_iface: 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc;`

dce_iface.version

A service interface has a version associated with it. Some versions of an interface may not be vulnerable to certain exploits. For this reason you can specify one or more version numbers in the `dce_iface` option, to determine whether it is necessary to check for a particular exploit.

Type: interval

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

Valid values: A set of one or more positive version numbers and a `range_operator` as specified in the Table 5: Range Formats.

Examples: `dce_iface: =6;`

dce_iface.any_frag

A DCE/RPC request can be broken up into one or more fragments. Flags are set in the DCE/RPC header to indicate whether the current fragment is the first, a middle, or the last fragment of the request. Many checks for data in the DCE/RPC request are relevant only if the DCE/RPC request is a first fragment (or full request). Thus, fragments that follow the first fragment contain data deeper into the DCE/RPC request. For example, a rule that looks for data in the first five bytes of the request (for example, a length field), finds the wrong data on a fragment other than the first. The start of subsequent fragments is offset some length from the beginning of the request. This can be a source of false positives in fragmented DCE/RPC traffic.

For this reason, by default the DCE_RPC inspector matches only the initial fragment in a request. To force the inspector to examine all fragments in a request for a match, add `any_frag` to the `dce_iface` rule option. Note that a defragmented DCE/RPC request is considered a full request.

Syntax: `dec_iface: any_frag;`

Examples: `dce_iface: any_frag;`

dce_opnum

Match a DCE RPC operation number, range of operation number, or list of operation number. This option represents one or more a specific function calls that can be made to an interface. After a client has bound to a specific service interface and makes a request to it, rules need to determine what function call the client is making to the service, to check for exploits that may lie within the function call. The functions call(s) are specified as a double-quote-enclosed list of operation numbers (opnums)

Type: string

Syntax: `dce_opnum: <opnum_list>;`

Where *`opnum_list`* is one of the following:

- A single integer.
- A comma-separated list of integers.
- A range of integers specified with a hyphen separating the lowest and highest numbers in the range.
- A combination of the above.

Valid values: A list of DCE/RPC request opnums.

Examples:

```
dce_opnum: "15";

dce_opnum: "15-18";

dce_opnum: "15, 18-20";

dce_opnum: "15, 17, 20-22";
```

dce_stub_data

This option places the detection cursor (used to traverse the packet payload in rules processing) at the beginning of the DCE/RPC stub data, regardless of preceding rule options. This option matches if there is DCE/RPC stub data present.

Syntax: `dce_stub_data;`

Examples: `dce_stub_data;`

Table 5: Range Formats

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

CHAPTER 6

DCE TCP Inspector

DCE TCP Inspector Overview

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

The DCE/RPC protocol allows processes on separate network hosts to communicate as if the processes were on the same host. These inter-process communications are commonly transported between hosts over TCP. Within the TCP transport, DCE/RPC might also be further encapsulated in the Windows Server Message Block (SMB) protocol or in Samba, an open-source SMB implementation used for inter-process communication in a mixed environment comprised of Windows, and UNIX or Linux operating systems.

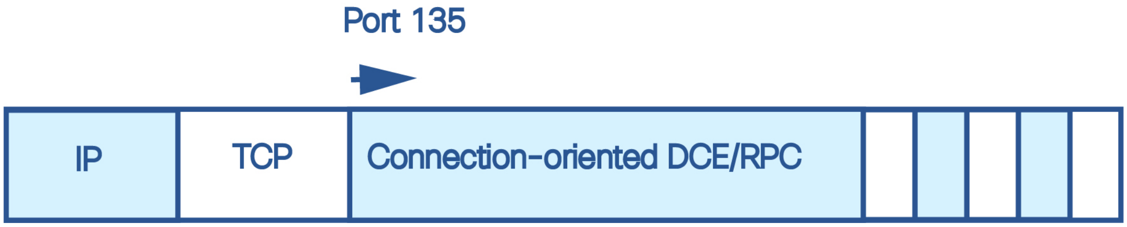
Although most DCE/RPC exploits occur in DCE/RPC client requests targeted for DCE/RPC servers, which could be practically any host on your network that is running Windows or Samba, exploits can also occur in server responses.

IP encapsulates all DCE/RPC transports. TCP transports all connection-oriented DCE/RPC. The DCE TCP inspector detects connection-oriented DCE/RPC and uses protocol-specific characteristics (such as header length and data fragment order) to:

- Detect DCE/RPC requests and responses encapsulated in TCP transports, including TCP-transported DCE/RPC using version 1 RPC over HTTP.
- Analyze DCE/RPC data streams and detect anomalous behavior and evasion techniques in DCE/RPC traffic.
- Defragment DCE/RPC.

- Normalize DCE/RPC traffic for processing by the rules engine.

The following diagram illustrates the point at which the DCE TCP inspector begins processing traffic for the TCP transport.



The well-known TCP port 135 identifies DCE/RPC traffic in the TCP transport. The figure does not include RPC over HTTP. For RPC over HTTP, connection-oriented DCE/RPC is transported directly over TCP as shown in the figure after an initial setup sequence over HTTP.

Target-Based Policies

Windows and Samba DCE/RPC implementations differ significantly. For example, all versions of Windows use the DCE/RPC context ID in the first fragment when defragmenting DCE/RPC traffic, and all versions of Samba use the context ID in the last fragment. As another example, Windows Vista uses the opnum (operation number) header field in the first fragment to identify a specific function call, and Samba and all other Windows versions use the opnum field in the last fragment.

For this reason, the `dce_tcp` inspector uses a target-based approach. When you configure a `dce_tcp` inspector instance, the `policy` parameter specifies a specific implementation of the DCE/RPC TCP protocol. This in combination with the host information establishes a default target-based server policy. Optionally, you can configure additional inspectors that target other hosts and DCE/RPC TCP implementations. The DCE/RPC TCP implementation specified by the default target-based server policy applies to any host not targeted by another `dce_tcp` inspector instance.

The DCE/RPC implementations the DCE TCP inspector can target with the `policy` parameter are:

- WinXP (default)
- Win2000
- WinVista
- Win2003
- Win2008
- Win7
- Samba
- Samba-3.0.37
- Samba-3.0.22
- Samba-3.0.20

Defragmentation

The DCE TCP inspector supports reassembling fragmented data packets before sending them to the detection engine. This feature is useful in inline mode to catch exploits early in the inspection process before full defragmentation is done, or to catch exploits that take advantage of fragmentation to conceal themselves. Be aware that disabling defragmentation may result in a large number of false negatives.

DCE TCP Inspector Parameters

DCE TCP port configuration

The `binder` inspector defines the DCE TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
   {
        "when": {
            "role": "any",
            "proto": "tcp",
            "service": "dcerpc",
            "ports": ""
        },
        "use": {
            "type": "dce_tcp"
        }
    }
]
```

max_frag_len

Specifies the maximum fragment length in bytes allowed for defragmentation. For processing larger fragments the inspector considers packet content to the specified size before defragmenting.

Note The value specified in this parameter must be greater than or equal to the depth to which the rules need to examine the data to ensure detection. To ensure that all data goes through detection, use the default value.

Type: integer

Valid range: `1514` to `65535`

Default value: `65535`

disable_defrag

Specifies whether to defragment fragmented DCE/RPC traffic. When this parameter is `true`, the inspector still detects anomalies and sends DCE/RPC data to the rules engine, but at the risk of missing exploits in fragmented DCE/RPC data.

Although this parameter provides the flexibility of not defragmenting traffic and can speed processing, most DCE/RPC exploits attempt to take advantage of fragmentation to hide the exploit. Enabling this parameter would bypass most known exploits, resulting in a large number of false negatives.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

limit_alerts

Specifies whether to limit DCE alerts to at most one per signature per flow.

Type: boolean

Valid values: `true`, `false`

Default value: `true`

reassemble_threshold

Specifies the minimum number of bytes in the DCE/RPC desegmentation and defragmentation buffers to queue before sending a reassembled packet to the rules engine. This parameter is useful in inline mode so as to potentially catch an exploit early before full defragmentation is done.

A low value increases the likelihood of early detection but could have a negative impact on performance. You should test for performance impact if you enable this parameter.

A value of `0` disables reassembly.

Type: integer

Valid range: `0` to `65535`

Default value: `0`

policy

Specifies the Windows or Samba DCE/RPC implementation used by the targeted host or hosts on your monitored network segment.

Type: enum

Valid values: A string selected from the following list: `Win2000`, `WinXP`, `WinVista`, `Win2003`, `Win2008`, `Win7`, `Samba`, `Samba-3.0.37`, `Samba-3.0.22`, `Samba-3.0.20`

Default value: `WinXP`

DCE TCP Inspector Rules

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

Table 6: DCE TCP Inspector Rules

GID:SID	Rule Message
133:27	connection oriented DCE/RPC - invalid major version
133:28	connection oriented DCE/RPC - invalid minor version
133:29	connection-oriented DCE/RPC - invalid PDU type

GID:SID	Rule Message
133:30	connection-oriented DCE/RPC - fragment length less than header size
133:31	connection-oriented DCE/RPC - remaining fragment length less than size needed
133:32	connection-oriented DCE/RPC - no context items specified
133:33	connection-oriented DCE/RPC -no transfer syntaxes specified
133:34	connection-oriented DCE/RPC - fragment length on non-last fragment less than maximum negotiated fragment transmit size for client
133:35	connection-oriented DCE/RPC - fragment length greater than maximum negotiated fragment transmit size
133:36	connection-oriented DCE/RPC - alter context byte order different from bind
133:37	connection-oriented DCE/RPC - call id of non first/last fragment different from call id established for fragmented request
133:38	connection-oriented DCE/RPC - opnum of non first/last fragment different from opnum established for fragmented request
133:39	connection-oriented DCE/RPC - context id of non first/last fragment different from context id established for fragmented request

DCE Inspectors Intrusion Rule Options

dce_iface

Specifies the following comma-separated elements:

- The UUID for a service interface.
- The interface version (optional). The default setting matches any version.
- An indicator of whether a rule should match on any fragment in a request (optional). The default setting matches only the first fragment.

In the DCE/RPC protocol, a client must bind to a service before making a call to it. When a client sends a bind request to the server, it can specify one or more service interfaces to bind to. Each interface is represented by a UUID, and each interface UUID is paired with a unique index (or context ID) that future requests can use to reference the service that the client is calling. The server responds with the interface UUIDs it accepts as valid and allows the client to make requests to those services. When a client makes a request, it will specify the context ID so the server knows to what service the client is making a request.

Using the `dce_iface` rule option, a rule can ask the inspector whether the client has bound to a specific interface UUID and whether this client request is making a request to that interface. This can eliminate false positives where more than one service is bound to successfully since the inspector can correlate the bind UUID to the context id used in the request.

The `dce_iface` option requires tracking client Bind and Alter Context requests as well as server Bind Ack and Alter Context responses for connection-oriented DCE/RPC in the inspector. For each Bind and Alter Context request, the client specifies a list of interface UUIDs along with a handle (or context id) for each interface UUID that will be used during the DCE/RPC session to reference the interface. The server response indicates which interfaces it allows the client to make requests to—it either accepts or rejects the client's wish to bind to a certain interface. This tracking is required so that when a request is processed, the context id used in the request can be correlated with the interface UUID for which it is a handle.

The `dce_iface` rule option matches if:

- the specified interface UUID matches the interface UUID (as referred to by the context ID) of the DCE/RPC request

and

- the `version` argument is not supplied, or the `version` argument is supplied and matches the interface UUID of the DCE/RPC request

and

- the `any_frag` argument is supplied, or the `any_frag` argument is absent and the `dce_iface` option matches the UUID and version criteria on the initial request fragment

Examples:

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, <2;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188,any_frag;
```

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, =1, any_frag;
```

dce_iface.uuid

A DCE/RPC request can specify whether UUID numbers are represented as big endian or little endian. The representation of the interface UUID in a request is different depending on the endianness specified in the request. The `dce_rpc` inspector normalizes the UUID. This means that the UUID specification in the `dce_iface` rule option must be written the same way regardless of the endianness of the request.

For example, a little endian Bind request would represent a UUID as follows:

```
|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|
```

and a big-endian Bind request would represent the same UUID as follows:

```
|5a 7b 91 f8 ff 00 11 d0 a9 b2 00 c0 4f b6 e6 fc|
```

In Snort 3 rules using the `dce_iface` option, the UUID must be represented in a string using big endian order regardless of the endian-ness of the request:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

Type: string

Syntax: `dce_iface: <UUID>;`

Valid values: Where: *`UUID`* is 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form: `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

Examples: `dce_iface: 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc;`

dce_iface.version

A service interface has a version associated with it. Some versions of an interface may not be vulnerable to certain exploits. For this reason you can specify one or more version numbers in the `dce_iface` option, to determine whether it is necessary to check for a particular exploit.

Type: interval

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

Valid values: A set of one or more positive version numbers and a `range_operator` as specified in the Table 7: Range Formats.

Examples: `dce_iface: =6;`

dce_iface.any_frag

A DCE/RPC request can be broken up into one or more fragments. Flags are set in the DCE/RPC header to indicate whether the current fragment is the first, a middle, or the last fragment of the request. Many checks for data in the DCE/RPC request are relevant only if the DCE/RPC request is a first fragment (or full request). Thus, fragments that follow the first fragment contain data deeper into the DCE/RPC request. For example, a rule that looks for data in the first five bytes of the request (for example, a length field), finds the wrong data on a fragment other than the first. The start of subsequent fragments is offset some length from the beginning of the request. This can be a source of false positives in fragmented DCE/RPC traffic.

For this reason, by default the DCE_RPC inspector matches only the initial fragment in a request. To force the inspector to examine all fragments in a request for a match, add `any_frag` to the `dce_iface` rule option. Note that a defragmented DCE/RPC request is considered a full request.

Syntax: `dec_iface: any_frag;`

Examples: `dce_iface: any_frag;`

dce_opnum

Match a DCE RPC operation number, range of operation number, or list of operation number. This option represents one or more a specific function calls that can be made to an interface. After a client has bound to a specific service interface and makes a request to it, rules need to determine what function call the client is making to the service, to check for exploits that may lie within the function call. The functions call(s) are specified as a double-quote-enclosed list of operation numbers (opnums)

Type: string

Syntax: `dce_opnum: <opnum_list>;`

Where *`opnum_list`* is one of the following:

- A single integer.
- A comma-separated list of integers.
- A range of integers specified with a hyphen separating the lowest and highest numbers in the range.
- A combination of the above.

Valid values: A list of DCE/RPC request opnums.

Examples:

```
dce_opnum: "15";
dce_opnum: "15-18";
dce_opnum: "15, 18-20";
dce_opnum: "15, 17, 20-22";
```

dce_stub_data

This option places the detection cursor (used to traverse the packet payload in rules processing) at the beginning of the DCE/RPC stub data, regardless of preceding rule options. This option matches if there is DCE/RPC stub data present.

Syntax: `dce_stub_data;`

Examples: `dce_stub_data;`

Table 7: Range Formats

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

CHAPTER 7

DNP3 Inspector

DNP3 Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`, `stream_udp`
Enabled	`false`

Distributed Network Protocol (DNP3) is a Supervisory Control and Data Acquisition (SCADA) protocol that was originally developed to provide consistent communication between electrical stations. DNP3 is widely used in the water, waste, and transportation industries.

The `dnp3` inspector detects anomalies in DNP3 traffic and analyzes the DNP3 protocol. The `dnp3` intrusion rule options access certain DNP3 protocol fields.

DNP3 Inspector Parameters

DNP3 TCP port configuration

The `binder` inspector defines the DNP3 TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
   {
       "when": {
           "role": "any",
           "service": "dnp3"
    },
       "use": {
           "type": "dnp3"
       }
    }
]
```

check_crc

Specifies whether to validate the checksums contained in DNP3 Link-Layer Frames. The `dnp3` inspector ignores frames with invalid checksums. If intrusion rule 145:1 is enabled, Snort generates alerts for invalid checksums.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

DNP3 Inspector Rules

Enable the `dnp3` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 8: DNP3 Inspector Rules

GID:SID	Rule Message
145:1	DNP3 link-layer frame contains bad CRC
145:2	DNP3 link-layer frame was dropped
145:3	DNP3 transport-layer segment was dropped during reassembly
145:4	DNP3 reassembly buffer was cleared without reassembling a complete message
145:5	DNP3 link-layer frame uses a reserved address
145:6	DNP3 application-layer fragment uses a reserved function code

DNP3 Inspector Intrusion Rule Options

dnp3_data

The `dnp3_data` keyword positions the detection cursor to the beginning of the DNP3 data in an application layer fragment, regardless of preceding rule options. With this option you can write rules based on the data within fragments without splitting up the data and adding CRCs every 16 bytes.

Syntax: `dnp3_data;`

Examples: `dnp3_data;`

dnp3_func

This option matches against the function code inside a DNP3 application layer request/response header. The code may be a decimal number or a string from the list below.

Type: string

Syntax: `dnp3_func: <DNP3_function>;`

Valid values: *`DNP3_function`* is one of the following:

- An integer from `0` to `255`
- `confirm` (Corresponds to function code `0`.)
- `read` (Corresponds to function code `1`.)
- `write` (Corresponds to function code `2`.)
- `select` (Corresponds to function code `3`.)
- `operate` (Corresponds to function code `4`.)
- `direct_operate` (Corresponds to function code `5`.)
- `direct_operat_nr` (Corresponds to function code `6`.)
- `immed_freeze` (Corresponds to function code `7`.)
- `immed_freeze_nr` (Corresponds to function code `8`.)
- `freeze_clear` (Corresponds to function code `9`.)
- `freeze_clear_nr` (Corresponds to function code `10`.)
- `freeze_at_time` (Corresponds to function code `11`.)
- `freeze_at_time_nr` (Corresponds to function code `12`.)
- `cold_restart` (Corresponds to function code `13`.)
- `warm_restart` (Corresponds to function code `14`.)
- `initialize_data` (Corresponds to function code `15`.)
- `initialize_appl` (Corresponds to function code `16`.)
- `start_appl` (Corresponds to function code `17`.)
- `stop_appl` (Corresponds to function code `18`.)
- `save_config` (Corresponds to function code `19`.)
- `enable_unsolicited` (Corresponds to function code `20`.)
- `disable_unsolicited` (Corresponds to function code `21`.)
- `assign_class` (Corresponds to function code `22`.)
- `delay_measure` (Corresponds to function code `23`.)
- `record_current_time` (Corresponds to function code `24`.)
- `open_file` (Corresponds to function code `25`.)

- `close_file` (Corresponds to function code `26`.)
- `delete_file` (Corresponds to function code `27`.)
- `get_file_info` (Corresponds to function code `28`.)
- `authenticate_file` (Corresponds to function code `29`.)
- `abort_file` (Corresponds to function code `30`.)
- `activate_config` (Corresponds to function code `31`.)
- `authenticate_req` (Corresponds to function code `32`.)
- `authenticate_err` (Corresponds to function code `33`.)
- `response` (Corresponds to function code `129`.)
- `unsolicited_response` (Corresponds to function code `130`.)
- `authenticate_resp` (Corresponds to function code `131`.)

Examples:

```
dnp3_func: 1;
```

```
dnp3_func: delete_file;
```

dnp3_ind

Provide a list of Internal Indicator flags to match against the Internal Indicator flags in a DNP3 application layer response header. If you provide multiple flags in one option, the rule fires if any one of the flags is set. To alert on multiple flags, use multiple rule options.

Type: string

Syntax: `dnp3_ind: "<flag> <flag>";`

Valid values: One or more DNP3 Internal Indicator flags where `flag` is one of the following:

- `all_stations`
- `class_1_events`
- `class_2_events`
- `class_3_events`
- `need_time`
- `local_control`
- `device_trouble`
- `device_restart`
- `no_func_code_support`
- `object_unknown`
- `parameter_error`

- `event_buffer_overflow`
- `already_executing`
- `config_corrupt`
- `reserved_2`
- `reserved_1`

Examples:

Alert on device restart OR on initiation of time synchronization:

```
dnp3_ind:"device_restart need_time";
```

Alert on class_1 AND class_2 AND class_3 events:

```
dnp3_ind:class_1_events; dnp3_ind:class_2_events; dnp3_ind:class_3_events;
```

dnp3_obj

Matches on DNP3 object header groups and variations.

Type: integer

Syntax: `dnp3_obj:<groupnum>,<varnum>;`

Valid values: DNP3 object group identifiers and variation identifiers, where:

- *`groupnum`* is an integer from `0` to `255` specifying a DNP3 object group.
- *`varnum`* is an integer from `0` to `255` specifying a variation within the object group.

Examples:

Alert on DNP3 `Date and Time` object:

```
dnp3_obj:50,1;
```

CHAPTER 8

FTP Client Inspector

FTP Client Inspector Overview

Type	Inspector (passive)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`ftp_server`, `stream_tcp`
Enabled	`true`

File Transfer Protocol (FTP) is a network protocol used to transfer files between clients and servers over TCP/IP. Once a client and server establish a connection, the client issues commands to the server to upload files to or download files from the server, and interprets responses from the server.

The `ftp_client` inspector examines and normalizes responses on the FTP command channel.

Given an FTP command channel buffer, the `ftp_client` inspector interprets FTP response codes and messages. The `ftp_client` inspector enforces correctness of the parameters, determines when an FTP command connection is encrypted and when an FTP data channel is opened.

FTP Client Inspector Parameters

bounce

Specifies whether to check for FTP bounces by examining the host information in `ftp port` commands issued by the client. When `bounce` is `true`, if the host information in an `ftp port` command does not match the configured client IP address or host information , and rule 125:8 is enabled, the system generates an alert, and in an inline deployment drops offending packets. This can be used to prevent FTP bounce attacks and permit FTP connections where the FTP data channel destination is different from the client.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ignore_telnet_erase_cmds

Specifies whether to ignore the telnet escape sequences for the erase character (TNC EAC) and the erase line character (TNC EAL) when normalizing the FTP command channel. You should set this parameter to match how the FTP client handles telnet erase commands. Newer FTP clients typically ignore these telnet escape sequences, while legacy clients typically process them. When the `ignore_telnet_erase_cmds` parameter is `false`, the inspector uses rule 125:1 to generate alerts, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

max_resp_len

Specifies the maximum length for all response messages accepted by the client in bytes. If the message for an FTP response (everything after the 3 digit return code) exceeds that length, and rule 125:6 is enabled, the system generates an alert, and, in an inline deployments, drop offending packets. This is used to check for buffer overflow exploits within FTP clients.

Type: integer

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

Default value: `4,294,967,295`

telnet_cmds

Specifies whether to check for telnet commands on the FTP command channel. The presence of such commands could indicate an evasion attempt on the FTP command channel.

You can enable rule 125:1 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

FTP Client Inspector Rules

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

Table 9: FTP Client Inspector Rules

GID:SID	Rule Message
125:1	TELNET cmd on FTP command channel
125:6	FTP response message was too long
125:8	FTP bounce attempt

FTP Client Inspector Intrusion Rule Options

The `ftp_client` inspector does not have any intrusion rule options.

CHAPTER 9

FTP Server Inspector

FTP Server Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	ftp_client, stream_tcp
Enabled	true

File Transfer Protocol (FTP) is a network protocol used to transfer files between clients and servers over TCP/IP. Once a client and server establish a connection, the client issues commands to the server to upload files to or download files from the server, and interprets responses from the server.

The ftp_server inspector examines and normalizes commands on the FTP command channel.

Given an FTP command channel buffer, the ftp_server inspector identifies the FTP commands and parameters, and enforces correctness of the parameters. The ftp_server determines when an FTP command connection is encrypted and when an FTP data channel is opened.

FTP Server Inspector Parameters

FTP Server port configuration

The binder inspector defines the FTP Server configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "role":"any",
            "service":"ftp",
            "ports": ""
        },
        "use": {
            "type":"ftp_server"
        }
    }
]
```

chk_str_fmt

Specifies a list of FTP commands to check for string format attacks. You can enable rule 125:5 to generate an alert, and in an inline deployment, drop offending packets when the inspector detects this condition. Separate multiple commands with a space character.

Type: string

Valid values: A list of valid FTP commands.

Default value: None

data_chan_cmds

Specifies a list of FTP commands to check for correct formatting. Separate multiple commands with a space character.

Type: string

Valid values: A list of one or more of the following commands: `PORT PASV LPRT LPSV EPRT EPSV`.

Default value: None

data_xfer_cmds

Specifies a list of data transfer commands. Check for correct formatting of the commands. Separate multiple commands with a space character.

Type: string

Valid values: A list of one or more of the following commands: `RETR STOR STOU APPE LIST NLST`.

Default value: None

file_put_cmds

Specifies a list of `PUT` commands. Check for correct formatting of the commands. Separate multiple commands with a space character.

Type: string

Valid values: A list of one or more of the following commands: `STOR STOU APPE`.

Default value: None

⚠

Caution Do not change the `file_put_cmds` parameter unless directed to do so by Support.

file_get_cmds

Specifies a list of `GET` commands. Check for correct formatting of the commands. Separate multiple commands with a space character.

Type: string

Valid values: A list of `GET` command, such as `RETR`.

Default value: None

Caution Do not change the `file_get_cmds` parameter unless directed to do so by Support.

encr_cmds

Specifies a list of commands related to secure connections. Check for correct formatting of the commands. Separate multiple commands with a space character.

Type: string

Valid values: A list of commands related to secure connections, for example: `AUTH`.

Default value: None

login_cmds

Specifies a list of commands related to the login process. Check for correct formatting of the commands. Separate multiple commands with a space character.

Type: string

Valid values: Specify a list of one or more commands: `USER`, `PASS`.

Default value: None

check_encrypted

Specifies whether to check an encrypted session for a command to end encryption. Use with the `encrypted_traffic` parameter.

You can enable rule 125:7 to generate events and, in an inline deployment, drop offending packets for this parameter.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

cmd_validity[]

An array of FTP commands and the criteria the inspector uses to validate them. These validity checks override the default checks performed by the `ftp_server` inspector (RFC 959).

You can enable rules 125:2 and 125:4 to generate events, and, in an line deployment, drop offending packets for this parameter.

Type: array (object)

Example:

```
{
   "cmd_validity": [
       {
           "command": "CWD",
           "format": "abc",
           "length": 250
       }
   ]
}
```

cmd_validity[].command

Specifies the name of an FTP command to validate.

Type: string

Valid values: A valid FTP command enclosed in double quotation marks.

Default value: None

cmd_validity[].format

Describes the valid format for `cmd_validity[].command`

Type: string

Valid values: One of the following formats:

- `int` – The parameter must be an integer
- `number` – The parameter much be an integer between 1 and 255
- `char` *`chars`* – The parameter must be a single character from *`chars`*, a list of one or more characters with no separators between them.
- `date` *`datefmt`* – The parameter follows the format specified, where *`datefmt`* is constructed using the following elements:
    - `#` = Number
    - `C` = Char
    - `[]` = Optional format enclosed
    - `|` = OR
    - `{}` = Choice of formats enclosed
    - `.+-` literal characters
- `string` – The parameter is an unlimited string.
- `host_port` – The parameter must be a host port specifier, per RFC 959.
- `long_host_port` – The parameter must be a long host port specifier per RFC 1639.
- `extended_host_port` – The parameter must be an extended host port specifier per RFC 2428.
- `{},|` – The parameter must be one of the choices enclosed within the braces, separated by |.

- `{}`, `[]` – The parameter must be one of the choices enclosed within the braces. Optional values are enclosed within the brackets.

Default value: None

cmd_validity[].length

Specifies the maximum length in bytes for the `cmd_validity[].command` parameter, overriding the default value defined in `def_max_param_len`. If the parameter for the FTP command exceeds the `cmd_validity[].length`, and rule 125:3 is enabled, Snort generates an alert. Use `cmd_validity[].length` to restrict specific commands to small parameter values.

Specify `0` to indicate unlimited length.

Type: integer

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

Default value: `0`

def_max_param_len

Specifies the default maximum length in bytes that the inspector permits for all FTP commands handled by the server. Use `def_max_param_len` for basic buffer overflow detection. (This can be overridden for individual commands using `cmd_validity[].length`.) You can enable rule 125:3 to generate events and, in an inline deployment, drop offending packets for this parameter.

Specify `0` to indicate unlimited length.

Type: integer

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

Default value: `100`

encrypted_traffic

Specifies whether to check for encrypted FTP traffic. Use with the `check_encrypted` parameter. You can enable rule 125:7 to generate events and, in an inline deployment, drop offending packets for this parameter.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ftp_cmds

A list of FTP commands the server supports beyond those described in RFC 959. (If your installation uses the "X" commands specified in RFC 775, for instance, you can add them to the inspector using this parameter.)

Type: string

Valid values: Space-separated list of valid FTP commands, enclosed in double quotation marks.

Default value: None

ignore_data_chan

Specifies whether to ignore the FTP data channels.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ignore_telnet_erase_cmds

Specifies whether to ignore the telnet escape sequences for the erase character (TNC EAC) and the erase line character (TNC EAL) when normalizing the FTP command channel. Set `ignore_telnet_erase_cmds` to match how your FTP server handles telnet erase commands. Newer FTP clients typically ignore these telnet escape sequences, while legacy clients typically process them.

If telnet erase commands are not ignored, and rule 125:1 is enabled, Snort generates an event, and, in an inline deployment, drops offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

print_cmds

Specifies whether to print the configuration for each FTP command for this server on initialization.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

telnet_cmds

Specifies whether to check for telnet commands on the FTP command channel. The presence of such commands could indicate an evasion attempt on the FTP command channel.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

FTP Server Inspector Rules

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

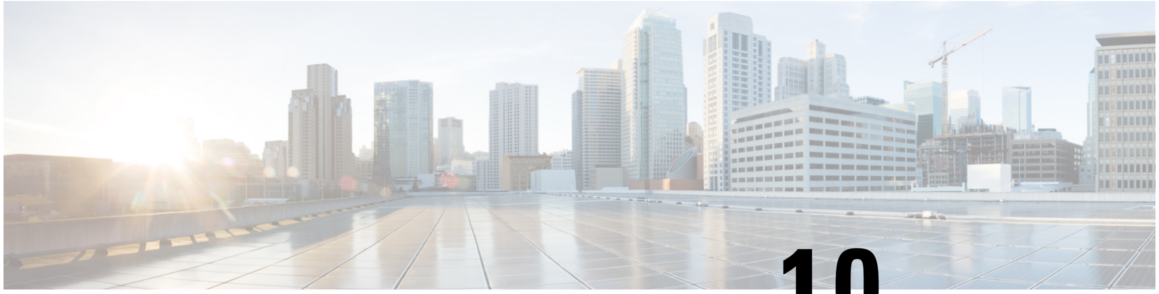
Table 10: FTP Server Inspector Rules

GID:SID	Rule Message
125:1	TELNET command on FTP command channel
125:2	invalid FTP command
125:3	FTP command parameters were too long
125:4	FTP command parameters were malformed

GID:SID	Rule Message
125:5	FTP command parameters contained potential string format
125:7	FTP traffic encrypted
125:9	evasive (incomplete) TELNET cmd on FTP command channel

FTP Server Inspector Intrusion Rule Options

The `ftp_server` inspector does not have any intrusion rule options.



CHAPTER 10

GTP Inspect Inspector

GTP Inspect Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_udp`
Enabled	`false`

The General Service Packet Radio (GPRS) Tunneling Protocol (GTP) provides communication over a GTP core network.

The `gtp_inspect` inspector detects anomalies in GTP traffic and forwards command channel signaling messages to the rules engine for inspection.

GTP Inspect Inspector Parameters

GTP Inspect service and ports configuration

The `binder` inspector defines the GTP Inspect `service` and `ports` configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
   {
       "when": {
           "service": "gtp_inspect",
           "role": any
```

```
        },
        "use": {
            "type": "gtp_inspect"
        }
    },
    {
        "when": {
            "proto": "tcp",
            "role": "server",
            "ports": "2123 2152 3386"
        },
        "use": {
            "type": "gtp_inspect"
        }
    }
]
```

version

Specifies a valid GTP version.

Type: integer

Valid values: `0`, `1`, `2`

Default value: `2`

messages[]

Specifies an array of information about valid GTP messages.

Type: array (object)

Example:

```
{
    messages: [
         {
             "type": 0,
             "name": ""
         }
    ]
}
```

messages[].type

Specifies a valid GTP message type. See Table 12: GTP Message Types table.

Type: integer

Valid range: `0` to `255`

Default value: None

messages[].name

Specifies a valid GTP message name. See Table 12: GTP Message Types table.

Type: string

Valid values: A valid GTP message name

Default value: None

infos[]

Specifies an array of GTP information elements.

Type: array (object)

Example:

```
{
    infos: [
        {
            "type": 0,
            "name": "echo_request",
            "length": 0
        }
    ]
}
```

infos[].type

Specifies a valid GTP element type code. See Table 13: GTP Information Elements table.

Type: integer

Valid range: `0` to `255`

Default value: `0`

infos[].name

Specifies a valid GTP element name.

Type: string

Valid values: Valid GTP information element names. See Table 13: GTP Information Elements table.

infos[].length

Specifies the length of a valid GTP information element.

Type: integer

Valid range: `0` to `255`

Default value: `0`

GTP Inspect Inspector Rules

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

Table 11: GTP Inspector Rules

GID:SID	Rule Message
143:1	message length is invalid
143:2	information element length is invalid

GID:SID	Rule Message
143:3	information elements are out of order
143:4	TEID is missing

GTP Inspect Inspector Intrusion Rule Options

The `gtp_inspect` inspector intrusion rule options allow you to inspect the GTP command channel for the GTP version, message type, and information elements.

You cannot use GTP options in combination with `content` or `byte_jump`. You must use `gtp_version` in each rule that uses `gtp_info` or `gtp_type`.

gtp_version

Check the specified GTP version against the version of the GTP control messages.

Type: integer

Syntax: `gtp_version: <version>;`

Valid values: `0`, `1`, `2`

Examples: `gtp_version: 1;`

gtp_type

Each GTP message is identified by a message type, which is comprised of both a numeric value and a string. Check the specified GTP types against the type of the GTP messages.

You can specify a defined decimal value for a message type, a defined string, or a comma-separated list of either or both in any combination, as seen in the following example:

Type: string

Syntax: `gtp_type: <message_type>;`

Valid values: Listed in the GTP Message Types table. See Table 12: GTP Message Types table.

Examples: `gtp_type: "10, 11, echo_request";`

The system uses an OR operation to match each value or string that you list. The order in which you list values and strings does not matter. Any single value or string in the list matches the keyword. The system generates an error if you attempt to save a rule that includes an unrecognized string or an out-of-range value.

Note that different GTP versions sometimes use different values for the same message type. For example, the `sgsn_context_request` message type has a value of `50` in GTPv0 and GTPv1, but a value of `130` in GTPv2.

The `gtp_type` option matches different values depending on the version number in the packet. For instance the `sgsn_context_request` message matches the value `50` in a GTPv0 or GTPv1 packet and the value `130` in a GTPv2 packet. The option does not match a packet when the message type value in the packet is not a known value for the version specified in the packet.

If you specify an integer for the message type, the option matches if the message type matches the value in the GTP packet, regardless of the version specified in the packet.

`gtp_message_type` is a numeric value or keyword from the Table 12: GTP Message Types table.

Table 12: GTP Message Types

Type	Name for Version 0	Name for Version 1	Name for Version 2
1	echo_request	echo_request	echo_request
2	echo_response	echo_response	echo_response
3	version_not_supported	version_not_supported	version_not_supported
4	node_alive_request	node_alive_request	N/A
5	node_alive_response	node_alive_response	N/A
6	redirection_request	redirection_request	N/A
7	redirection_response	redirection_response	N/A
16	create_pdp_context_request	create_pdp_context_request	N/A
17	create_pdp_context_response	create_pdp_context_response	N/A
18	update_pdp_context_request	update_pdp_context_request	N/A
19	update_pdp_context_response	update_pdp_context_response	N/A
20	delete_pdp_context_request	delete_pdp_context_request	N/A
21	delete_pdp_context_response	delete_pdp_context_response	N/A
22	create_aa_pdp_context_request	init_pdp_context_activation_request	N/A
23	create_aa_pdp_context_response	init_pdp_context_activation_response	N/A
24	delete_aa_pdp_context_request	N/A	N/A
25	delete_aa_pdp_context_response	N/A	N/A
26	error_indication	error_indication	N/A
27	pdu_notification_request	pdu_notification_request	N/A
28	pdu_notification_response	pdu_notification_response	N/A
29	pdu_notification_reject_request	pdu_notification_reject_request	N/A
30	pdu_notification_reject_response	pdu_notification_reject_response	N/A
31	N/A	supported_ext_header_notification	N/A
32	send_routing_info_request	send_routing_info_request	create_session_request
33	send_routing_info_response	send_routing_info_response	create_session_response
34	failure_report_request	failure_report_request	modify_bearer_request

Type	Name for Version 0	Name for Version 1	Name for Version 2
35	failure_report_response	failure_report_response	modify_bearer_response
36	note_ms_present_request	note_ms_present_request	delete_session_request
37	note_ms_present_response	note_ms_present_response	delete_session_response
38	N/A	N/A	change_notification_request
39	N/A	N/A	change_notification_response
48	identification_request	identification_request	N/A
49	identification_response	identification_response	N/A
50	sgsn_context_request	sgsn_context_request	N/A
51	sgsn_context_response	sgsn_context_response	N/A
52	sgsn_context_ack	sgsn_context_ack	N/A
53	N/A	forward_relocation_request	N/A
54	N/A	forward_relocation_response	N/A
55	N/A	forward_relocation_complete	N/A
56	N/A	relocation_cancel_request	N/A
57	N/A	relocation_cancel_response	N/A
58	N/A	forward_srns_contex	N/A
59	N/A	forward_relocation_complete_ack	N/A
60	N/A	forward_srns_contex_ack	N/A
64	N/A	N/A	modify_bearer_command
65	N/A	N/A	modify_bearer_failure_indication
66	N/A	N/A	delete_bearer_command
67	N/A	N/A	delete_bearer_failure_indication
68	N/A	N/A	bearer_resource_command
69	N/A	N/A	bearer_resource_failure_indication
70	N/A	ran_info_relay	downlink_failure_indication
71	N/A	N/A	trace_session_activation
72	N/A	N/A	trace_session_deactivation
73	N/A	N/A	stop_paging_indication

Type	Name for Version 0	Name for Version 1	Name for Version 2
95	N/A	N/A	create_bearer_request
96	N/A	mbms_notification_request	create_bearer_response
97	N/A	mbms_notification_response	update_bearer_request
98	N/A	mbms_notification_reject_request	update_bearer_response
99	N/A	mbms_notification_reject_response	delete_bearer_request
100	N/A	create_mbms_context_request	delete_bearer_response
101	N/A	create_mbms_context_response	delete_pdn_request
102	N/A	update_mbms_context_request	delete_pdn_response
103	N/A	update_mbms_context_response	N/A
104	N/A	delete_mbms_context_request	N/A
105	N/A	delete_mbms_context_response	N/A
112	N/A	mbms_register_request	N/A
113	N/A	mbms_register_response	N/A
114	N/A	mbms_deregister_request	N/A
115	N/A	mbms_deregister_response	N/A
116	N/A	mbms_session_start_request	N/A
117	N/A	mbms_session_start_response	N/A
118	N/A	mbms_session_stop_request	N/A
119	N/A	mbms_session_stop_response	N/A
120	N/A	mbms_session_update_request	N/A
121	N/A	mbms_session_update_response	N/A
128	N/A	ms_info_change_request	identification_request
129	N/A	ms_info_change_response	identification_response
130	N/A	N/A	sgsn_context_request
131	N/A	N/A	sgsn_context_response
132	N/A	N/A	sgsn_context_ack
133	N/A	N/A	forward_relocation_request
134	N/A	N/A	forward_relocation_response

Type	Name for Version 0	Name for Version 1	Name for Version 2
135	N/A	N/A	forward_relocation_complete
136	N/A	N/A	forward_relocation_complete_ack
137	N/A	N/A	forward_access
138	N/A	N/A	forward_access_ack
139	N/A	N/A	relocation_cancel_request
140	N/A	N/A	relocation_cancel_response
141	N/A	N/A	configuration_transfer_tunnel
149	N/A	N/A	detach
150	N/A	N/A	detach_ack
151	N/A	N/A	cs_paging
152	N/A	N/A	ran_info_relay
153	N/A	N/A	alert_mme
154	N/A	N/A	alert_mme_ack
155	N/A	N/A	ue_activity
156	N/A	N/A	ue_activity_ack
160	N/A	N/A	create_forward_tunnel_request
161	N/A	N/A	create_forward_tunnel_response
162	N/A	N/A	suspend
163	N/A	N/A	suspend_ack
164	N/A	N/A	resume
165	N/A	N/A	resume_ack
166	N/A	N/A	create_indirect_forward_tunnel_request
167	N/A	N/A	create_indirect_forward_tunnel_response
168	N/A	N/A	delete_indirect_forward_tunnel_request
169	N/A	N/A	delete_indirect_forward_tunnel_response
170	N/A	N/A	release_access_bearer_request
171	N/A	N/A	release_access_bearer_response
176	N/A	N/A	downlink_data

Type	Name for Version 0	Name for Version 1	Name for Version 2
177	N/A	N/A	downlink_data_ack
179	N/A	N/A	pgw_restart
180	N/A	N/A	pgw_restart_ack
200	N/A	N/A	update_pdn_request
201	N/A	N/A	update_pdn_response
211	N/A	N/A	modify_access_bearer_request
212	N/A	N/A	modify_access_bearer_response
231	N/A	N/A	mbms_session_start_request
232	N/A	N/A	mbms_session_start_response
233	N/A	N/A	mbms_session_update_request
234	N/A	N/A	mbms_session_update_response
235	N/A	N/A	mbms_session_stop_request
236	N/A	N/A	mbms_session_stop_response
240	data_record_transfer_request	data_record_transfer_request	N/A
241	data_record_transfer_response	data_record_transfer_response	N/A
254	N/A	end_marker	N/A
255	pdu	pdu	N/A

gtp_info

A GTP message can include multiple information elements, each of which is identified by both a defined numeric value and a defined string. You can use the `gtp_info` option to start inspection at the beginning of a specified information element, and restrict inspection to that information element.

You can specify either the defined decimal value or the defined string for an information element. You can specify a single value or string, and you can use multiple `gtp_info` options in a rule to inspect multiple information elements.

When a message includes multiple information elements of the same type, all are inspected for a match. When information elements occur in an invalid order, only the last instance is inspected.

Depending on the version, a GTP message can use different values for the same information element. For example, the `cause` information element has a value of `1` in GTPv0 and GTPv1, but a value of `2` in GTPv2.

The `gtp_info` option matches different values depending on the version number in the packet. In the example above, the keyword matches the information element value `1` in a GTPv0 or GTPv1 packet and the value `2` in a GTPv2 packet. The option does not match a packet when the information element value in the packet is not a known value for the version specified in the packet.

If you specify an integer for the information element, the option matches if the message type matches the value in the GTP packet, regardless of the version specified in the packet.

Type: string

Syntax: `gtp_info: <identifier>;`

Valid values: Listed in the Table 13: GTP Information Elements table.

Examples: `gtp_info: "qos";`

Table 13: GTP Information Elements

Type	Name for Version 0	Name for Version 1	Name for Version 2
1	cause	cause	imsi
2	imsi	imsi	cause
3	rai	rai	recovery
4	tlli	tlli	N/A
5	p_tmsi	p_tmsi	N/A
6	qos	N/A	N/A
8	recording_required	recording_required	N/A
9	authentication	authentication	N/A
10	N/A	N/A	N/A
11	map_cause	map_cause	N/A
12	p_tmsi_sig	p_tmsi_sig	N/A
13	ms_validated	ms_validated	N/A
14	recovery	recovery	N/A
15	selection_mode	selection_mode	N/A
16	flow_label_data_1	teid_1	N/A
17	flow_label_signalling	teid_control	N/A
18	flow_label_data_2	teid_2	N/A
19	ms_unreachable	teardown_ind	N/A
20	N/A	nsapi	N/A
21	N/A	ranap	N/A
22	N/A	rab_context	N/A
23	N/A	radio_priority_sms	N/A

Type	Name for Version 0	Name for Version 1	Name for Version 2
24	N/A	radio_priority	N/A
25	N/A	packet_flow_id	N/A
26	N/A	charging_char	N/A
27	N/A	trace_ref	N/A
28	N/A	trace_type	N/A
29	N/A	ms_unreachable	N/A
71	N/A	N/A	apn
72	N/A	N/A	ambr
73	N/A	N/A	ebi
74	N/A	N/A	ip_addr
75	N/A	N/A	mei
76	N/A	N/A	msisdn
77	N/A	N/A	indication
78	N/A	N/A	pco
79	N/A	N/A	paa
80	N/A	N/A	bearer_qos
80	N/A	N/A	flow_qos
82	N/A	N/A	rat_type
83	N/A	N/A	serving_network
84	N/A	N/A	bearer_tft
85	N/A	N/A	tad
86	N/A	N/A	uli
87	N/A	N/A	f_teid
88	N/A	N/A	tmsi
89	N/A	N/A	cn_id
90	N/A	N/A	s103pdf
91	N/A	N/A	s1udf
92	N/A	N/A	delay_value

Type	Name for Version 0	Name for Version 1	Name for Version 2
93	N/A	N/A	bearer_context
94	N/A	N/A	charging_id
95	N/A	N/A	charging_char
96	N/A	N/A	trace_info
97	N/A	N/A	bearer_flag
99	N/A	N/A	pdn_type
100	N/A	N/A	pti
101	N/A	N/A	drx_parameter
103	N/A	N/A	gsm_key_tri
104	N/A	N/A	umts_key_cipher_quin
105	N/A	N/A	gsm_key_cipher_quin
106	N/A	N/A	umts_key_quin
107	N/A	N/A	eps_quad
108	N/A	N/A	umts_key_quad_quin
109	N/A	N/A	pdn_connection
110	N/A	N/A	pdn_number
111	N/A	N/A	p_tmsi
112	N/A	N/A	p_tmsi_sig
113	N/A	N/A	hop_counter
114	N/A	N/A	ue_time_zone
115	N/A	N/A	trace_ref
116	N/A	N/A	complete_request_msg
117	N/A	N/A	guti
118	N/A	N/A	f_container
119	N/A	N/A	f_cause
120	N/A	N/A	plmn_id
121	N/A	N/A	target_id
123	N/A	N/A	packet_flow_id

Type	Name for Version 0	Name for Version 1	Name for Version 2
124	N/A	N/A	rab_contex
125	N/A	N/A	src_rnc_pdcp
126	N/A	N/A	udp_src_port
127	charge_id	charge_id	apn_restriction
128	end_user_address	end_user_address	selection_mode
129	mm_context	mm_context	src_id
130	pdp_context	pdp_context	N/A
131	apn	apn	change_report_action
132	protocol_config	protocol_config	fq_csid
133	gsn	gsn	channel
134	msisdn	msisdn	emlpp_pri
135	N/A	qos	node_type
136	N/A	authentication_qu	fqdn
137	N/A	tft	ti
138	N/A	target_id	mbms_session_duration
139	N/A	utran_trans	mbms_service_area
140	N/A	rab_setup	mbms_session_id
141	N/A	ext_header	mbms_flow_id
142	N/A	trigger_id	mbms_ip_multicast
143	N/A	omc_id	mbms_distribution_ack
144	N/A	ran_trans	rfsp_index
145	N/A	pdp_context_pri	uci
146	N/A	addi_rab_setup	csg_info
147	N/A	sgsn_number	csg_id
148	N/A	common_flag	cmi
149	N/A	apn_restriction	service_indicator
150	N/A	radio_priority_lcs	detach_type
151	N/A	rat_type	ldn

Type	Name for Version 0	Name for Version 1	Name for Version 2
152	N/A	user_loc_info	node_feature
153	N/A	ms_time_zone	mbms_time_to_transfer
154	N/A	imei_sv	throttling
155	N/A	camel	arp
156	N/A	mbms_ue_context	epc_timer
157	N/A	tmp_mobile_group_id	signalling_priority_indication
158	N/A	rim_routing_addr	tmgi
159	N/A	mbms_config	mm_srvcc
160	N/A	mbms_service_area	flags_srvcc
161	N/A	src_rnc_pdcp	nmbr
162	N/A	addi_trace_info	N/A
163	N/A	hop_counter	N/A
164	N/A	plmn_id	N/A
165	N/A	mbms_session_id	N/A
166	N/A	mbms_2g3g_indicator	N/A
167	N/A	enhanced_nsapi	N/A
168	N/A	mbms_session_duration	N/A
169	N/A	addi_mbms_trace_info	N/A
170	N/A	mbms_session_repetition_num	N/A
171	N/A	mbms_time_to_data	N/A
173	N/A	bss	N/A
174	N/A	cell_id	N/A
175	N/A	pdu_num	N/A
177	N/A	mbms_bearer_capab	N/A
178	N/A	rim_routing_disc	N/A
179	N/A	list_pfc	N/A
180	N/A	ps_xid	N/A
181	N/A	ms_info_change_report	N/A

Type	Name for Version 0	Name for Version 1	Name for Version 2
182	N/A	direct_tunnel_flags	N/A
183	N/A	correlation_id	N/A
184	N/A	bearer_control_mode	N/A
185	N/A	mbms_flow_id	N/A
186	N/A	mbms_ip_multicast	N/A
187	N/A	mbms_distribution_ack	N/A
188	N/A	reliable_inter_rat_handover	N/A
189	N/A	rfsp_index	N/A
190	N/A	fqdn	N/A
191	N/A	evolved_allocation1	N/A
192	N/A	evolved_allocation2	N/A
193	N/A	extended_flags	N/A
194	N/A	uci	N/A
195	N/A	csg_info	N/A
196	N/A	csg_id	N/A
197	N/A	cmi	N/A
198	N/A	apn_ambr	N/A
199	N/A	ue_network	N/A
200	N/A	ue_ambr	N/A
201	N/A	apn_ambr_nsapi	N/A
202	N/A	ggsn_backoff_timer	N/A
203	N/A	signalling_priority_indication	N/A
204	N/A	signalling_priority_indication_nsapi	N/A
205	N/A	high_bitrate	N/A
206	N/A	max_mbr	N/A
251	charging_gateway_addr	charging_gateway_addr	N/A
255	private_extension	private_extension	private_extension

CHAPTER 11

HTTP Inspect Inspector

HTTP Inspect Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`true`

Hypertext Transfer Protocol (HTTP) is an application layer protocol that enables the exchange of hypermedia (audio, video, images, and text) between a client and server. HTTP is a stateless protocol that requires reliable message transmission. Communication between a client and a server is in the form of HTTP requests and responses.

An HTTP/1.1 server typically uses port 80 over TCP/IP. The secure version of HTTP (HTTP/TLS or HTTPS) uses port 443. HTTP defines access control and authentication mechanisms in the protocol.

HTTP/2 contains improvements to increase speed and push more information than the client requested, but operates over the same ports and protocols as HTTP/1.1. HTTP/2-specific rules are configured with `service:http2`.

HTTP/3 is connection-less, using the QUIC (Quick UDP Internet Connections) protocol rather than TCP, and can support more active streams with better loss recovery. HTTP/3 uses the same messaging as prior versions of HTTP. HTTP/3-specific rules are configured with `service:http3`.

The HTTP inspector supports all three versions of HTTP in an identical fashion.

The `http_inspect` inspector detects and analyzes the protocol data unit (PDU) of the HTTP message. `http_inspect` receives the TCP payload from the TCP stream and examines the encapsulated HTTP message.

The HTTP inspector can detect the following HTTP message sections:

- Request line
- Status line
- Headers
- Content-Length message body (message body defined by Content-length header)
- Chunked message body
- Previous message body (message body with no Content-Length header
- Trailers

The `http_inspect` inspector detects and normalizes all HTTP header fields and the components of the HTTP URI. The `http_inspect` inspector does not normalize the TCP port.

The `http_inspect` inspector can detect four types of URI:

- Asterisk (`*`): not normalized
- Authority: a URI used with the HTTP CONNECT method
- Origin: a URI that begins with a slash (no scheme or authority present)
- Absolute: a URI that includes a scheme, host, and an absolute path

An HTTP URI can include:

- Scheme (ftp, http, or https)
- Host (domain name of server)
- TCP port
- Path (directory and file)
- Query (request parameters)
- Fragment (part of the file)

You can configure the `http_inspect` inspector to alert on the sections of the HTTP message. For example:

- Specify the amount of bytes to read from the HTTP request or response body
- Enable JavaScript detection and normalization
- Handle various types of file decompression
- Customize the decoding of the HTTP URI

Note The `http_inspect` inspector can partially inspect the stream TCP payload.

Best Practices for Configuring the HTTP Inspect Inspector

Consider the following best practices when configuring the `http_inspect` inspector:

- Set the `request_depth` and `response_depth` parameters if your HTTP traffic includes large video files.
- Use the default settings for the HTTP URI inspection parameters:

  ```
  "utf8": "true"

  "plus_to_space": "true"

  "percent_u": "true"

  "utf8_bare_byte": "true"

  "iis_unicode": "true"

  "iis_double_decode": "true"
  ```

HTTP Inspect Inspector Parameters

HTTP service configuration

The `binder` inspector defines the HTTP service configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
             "service": "http",
             "role": any
        },
        "use": {
            "type": "http_inspect"
        }
    }
]
```

request_depth

Specifies the number of bytes to read from the HTTP message request body.

Specify `-1` to place no limit on the number of bytes to inspect. We recommend that you specify the `request_depth` and `response_depth` parameters to limit the amount of HTTP body data to analyze.

To inspect only the HTTP headers, set `request_depth` to `0`.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default Value: `-1`

response_depth

Specifies the number of bytes to read from the HTTP message response body.

Specify `-1` to place no limit on the number of bytes to inspect. We recommend that you specify the `request_depth` and `response_depth` parameters to limit the amount of HTTP body data to analyze.

To inspect only the HTTP headers, set `response_depth` to `0`.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default Value: `-1`

unzip

Specifies whether to decompress `gzip` files and deflate message bodies before inspecting them. When you turn off decompression, the HTTP inspector is unable to process all parts of the HTTP message body. The `http_inspect` inspector can process the HTTP headers.

Type: boolean

Valid values: `true, false`

Default value: `true`

maximum_host_length

Specifies the maximum number of bytes allowed in the `Host` HTTP header value.

Specify `-1` to place no limit on the header value length.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

maximum_chunk_length

Specifies the maximum number of bytes allowed in an HTTP message body chunk.

Specify `-1` to place no limit on the number of bytes in an HTTP chunk.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

normalize_utf

Specifies whether to normalize UTF encodings (UTF-8, UTF-7, UTF-16LE, UTF-16BE, UTF-32LE, and UTF-32BE) found in the HTTP response body. The `http_inspect` inspector determines the UTF character encoding from the HTTP `Content-Type` header.

Type: boolean

Valid values: `true, false`

Default value: `true`

decompress_pdf

Specifies whether to decompress the deflate-compatible compressed portions of the `application/pdf` (PDF) files found in the HTTP response body. The `http_inspect` inspector decompresses PDF files with the `/FlateDecode` stream filter.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_swf

Specifies whether to decompress `application/vnd.adobe.flash-movie` (SWF) files found in the HTTP response body.

Note You can only decompress the compressed portions of files found in the HTTP GET responses.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_vba

Specifies whether to decompress Microsoft Office Visual Basic for Applications macro files found in the HTTP response body.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_zip

Specifies whether to decompress `application/zip` (ZIP) files found in the HTTP response body.

Type: boolean

Valid values: `true, false`

Default value: `false`

script_detection

Specifies whether to inspect the JavaScript content after detecting the script end element (<\script>). When `http_inspect` detects the end of a script, it immediately forwards the partially read message body for early detection. Script detection enables Snort to quickly block response messages that may contain malicious JavaScript.

Type: boolean

Valid values: `true, false`

Default value: `false`

normalize_javascript

Specifies whether to use the legacy mechanism to normalize JavaScript in the HTTP response body. This option configures the legacy JavaScript normalizer. The `http_inspect` inspector normalizes obfuscated JavaScript data including the `unescape` and `decodeURI` functions, and the `String.fromCharCode` method. The HTTP inspector normalizes encodings within the unescape, decodeURI, and decodeURIComponent functions: %XX, %uXXXX, XX, and uXXXXi.

The `http_inspect` inspector detects consecutive white spaces and normalizes them into a single space. When `normalize_javascript` is enabled, you can set `max_javascript_whitespaces` to limit the number of consecutive white spaces in the obfuscated JavaScript.

Type: boolean

Valid values: `true, false`

Default value: `false`

js_norm_bytes_depth

Specifies the number of input JavaScript bytes to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

The `http_inspect` inspector detects consecutive white spaces and normalizes them into a single space. The inspector keeps track of scripts in different PDUs where the start <script> is in one PDU and the end </script> is in another PDU to normalize the traffic effectively. A new buffer `js_data` was added to the Snort 3 IPS buffer that uses the Just in Time (JIT) approach to detect and normalize JavaScript code where the normalizer is called only when this option is used in the rule.

The `http_inspect` inspector normalizes the function name, variable name, and the label name associated with the JavaScript code. In addition, the inspector normalizes JavaScript code transferred in the form of external script using the application/javascript or similar MIME type. The normalizer performs automatic semicolon insertion where the JavaScript functionality is not altered from its original input from the client side.

The `http_inspect` inspector normalizes obfuscated JavaScript data including the `unescape`, `decodeURI,` and `decodeURIComponent` functions, and the `String.fromCharCode` and `String.fromCodePoint` methods. The HTTP inspector normalizes encodings within the unescape, decodeURI, and decodeURIComponent functions: %XX, %uXXXX, \uXX, \u{XXXX}\xXX, decimal code point, and hexadecimal code point.

The `http_inspect` inspector also normalizes the Javascript plus (+) operator and concatenates strings using the operator.

Specify `-1` to place no limit on the number of JavaScript bytes.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

js_norm_identifier_depth

Specifies the maximum number of unique JavaScript identifiers to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `0` to `65536`

Default value: `65536`

js_norm_max_bracket_depth

Specifies the maximum depth of JavaScript bracket nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `1` to `65535`

Default value: `256`

js_norm_max_scope_depth

Specifies the maximum depth of JavaScript scope nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `1` to `65535`

Default value: `256`

js_norm_max_tmpl_nest

Specifies the maximum depth of JavaScript template literal nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `0` to `255`

Default value: `32`

max_javascript_whitespaces

Specifies the maximum consecutive whitespaces allowed within the JavaScript obfuscated data.

Type: integer

Valid range: `1` to `65535`

Default value: `200`

percent_u

Specifies whether to normalize the `%uNNNN` and `%UNNNN` encodings. The four `N` characters represent a hex-encoded value that correlates to a Microsoft internet information services (IIS) Unicode code point. As legitimate clients rarely use `%u` encodings, we recommend that you normalize the HTTP traffic encoded with `%u` encodings.

Type: boolean

Valid values: `true, false`

Default value: `false`

utf8

Specifies whether to normalize the standard UTF-8 Unicode sequences in the URI. The `http_inspect` inspector can normalize two or three byte UTF-8 characters into a single byte.

Type: boolean

Valid values: `true, false`

Default value: `true`

utf8_bare_byte

Specifies whether to normalize UTF-8 characters which include bytes that are not URL or percent encoded. We recommend that you enable the `utf8_bare_byte` parameter.

Type: boolean

Valid values: `true, false`

Default value: `false`

iis_unicode

Specifies whether to normalize the characters in the HTTP message with the Unicode code point.

Note We recommend that you enable the `iis_unicode` parameter. Unicode is commonly seen in attacks and evasion attempts.

Type: boolean

Valid values: `true, false`

Default value: `false`

iis_unicode_code_page

Specifies whether to use the code page from the IIS Unicode map file.

Type: integer

Valid range: `1` to `65535`

Default value: `1252`

iis_double_decode

Specifies whether to normalize characters by performing double decoding of URL encoded characters. Decodes IIS double encoded traffic by making two passes through the request URI. We recommend that you enable the `iis_double_decode` parameter. Double encoding is typically found only in attack scenarios.

Type: boolean

Valid values: `true, false`

Default value: `true`

oversize_dir_length

Specifies the maximum number of bytes allowed for the URL directory.

Type: integer

Valid range: `1` to `65535`

Default value: `300`

backslash_to_slash

Specifies whether to replace the backslash (`\`) with forward slash (`/`) in the URIs.

Type: boolean

Valid values: `true, false`

Default value: `true`

plus_to_space

Specifies whether to replace the plus sign (+) with <sp> in the URIs.

Type: boolean

Valid values: `true, false`

Default value: `true`

simplify_path

Specifies whether to reduce the URI directory path to the simplest form. A URI directory path that includes extra traversals may include: `.`, `..`, and `/`.

Type: boolean

Valid values: `true, false`

Default value: `true`

xff_headers

Specifies the types of `X-Forwarded-For` HTTP header to examine. In the `xff_headers` parameter, list the `X-Forwarded-For` headers from highest to lowest preference.

You can define custom `X-Forwarded-For` type headers. The HTTP header, which carries the original client IP address, can have a vendor-specific header name. In this scenario, the `xff_headers` parameter provides a way to introduce custom headers to the HTTP inspector.

The `xff_headers` default value is `x-forwarded-for true-client-ip`, two commonly known headers. If both default headers are present in the stream, `x-forwarded-for` is preferred over `true-client-ip`. When specifying multiple `X-Forwarded-For` HTTP headers, delimit the header names with a space.

Type: string

Valid values: `x-forwarded-for`, `true-client-ip`

Default value: `x-forwarded-for true-client-ip`

HTTP Inspect Inspector Rules

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

Table 14: HTTP Inspect Inspector Rules

GID:SID	Rule Message
119:1	URI has percent-encoding of an unreserved character
119:2	URI is percent encoded and the result is percent encoded again
119:3	URI has non-standard %u-style Unicode encoding
119:4	URI has Unicode encodings containing bytes that were not percent-encoded
119:6	URI has two-byte or three-byte UTF-8 encoding

GID:SID	Rule Message
119:7	URI has unicode map code point encoding
119:8	URI path contains consecutive slash characters
119:9	backslash character appears in the path portion of a URI
119:10	URI path contains /./ pattern repeating the current directory
119:11	URI path contains /../ pattern moving up a directory
119:12	Tab character in HTTP start line
119:13	HTTP start line or header line terminated by LF without a CR
119:14	Normalized URI includes character from bad_characters list
119:15	URI path contains a segment that is longer than the oversize_dir_length parametery
119:16	chunk length exceeds configured maximum_chunk_length
119:18	URI path includes /../ that goes above the root directory
119:19	HTTP header line exceeds 4096 bytes
119:20	HTTP message has more than 200 header fields
119:21	HTTP message has more than one Content-Length header value
119:24	Host header field appears more than once or has multiple values
119:25	length of HTTP Host header field value exceeds maximum_host_length option
119:28	HTTP POST or PUT request without content-length or chunks
119:31	HTTP request method is not known to Snort
119:32	HTTP request uses primitive HTTP format known as HTTP/0.9
119:33	HTTP request URI has space character that is not percent-encoded
119:34	HTTP connection has more than 100 simultaneous pipelined requests that have not been answered
119:102	invalid status code in HTTP response
119:104	HTTP response has UTF character set that failed to normalize
119:105	HTTP response has UTF-7 character set
119:109	more than one level of JavaScript obfuscation
119:110	consecutive JavaScript whitespaces exceed maximum allowed
119:111	multiple encodings within JavaScript obfuscated data

GID:SID	Rule Message
119:112	SWF file zlib decompression failure
119:113	SWF file LZMA decompression failure
119:114	PDF file deflate decompression failure
119:115	PDF file unsupported compression type
119:116	PDF file with more than one compression applied
119:117	PDF file parse failure
119:201	not HTTP traffic or unrecoverable HTTP protocol error
119:202	chunk length has excessive leading zeros
119:203	white space before or between HTTP messages
119:204	request message without URI
119:205	control character in HTTP response reason phrase
119:206	illegal extra whitespace in start line
119:207	corrupted HTTP version
119:209	format error in HTTP header
119:210	chunk header options present
119:211	URI badly formatted
119:212	unrecognized type of percent encoding in URI
119:213	HTTP chunk misformatted
119:214	white space adjacent to chunk length
119:215	white space within header name
119:216	excessive gzip compression
119:217	gzip decompression failed
119:218	HTTP 0.9 requested followed by another request
119:219	HTTP 0.9 request following a normal request
119:220	message has both Content-Length and Transfer-Encoding
119:221	status code implying no body combined with Transfer-Encoding or nonzero Content-Length
119:222	Transfer-Encoding not ending with chunked

GID:SID	Rule Message
119:223	Transfer-Encoding with encodings before chunked
119:224	misformatted HTTP traffic
119:225	unsupported Content-Encoding used
119:226	unknown Content-Encoding used
119:227	multiple Content-Encodings applied
119:228	server response before client request
119:229	PDF/SWF/ZIP decompression of server response too big
119:230	nonprinting character in HTTP message header name
119:231	bad Content-Length value in HTTP header
119:232	HTTP header line wrapped
119:233	HTTP header line terminated by CR without a LF
119:234	chunk terminated by nonstandard separator
119:235	chunk length terminated by LF without CR
119:236	more than one response with 100 status code
119:237	100 status code not in response to Expect header
119:238	1XX status code other than 100 or 101
119:239	Expect header sent without a message body
119:240	HTTP 1.0 message with Transfer-Encoding header
119:241	Content-Transfer-Encoding used as HTTP header
119:242	illegal field in chunked message trailers
119:243	header field inappropriately appears twice or has two values
119:244	invalid value chunked in Content-Encoding header
119:245	206 response sent to a request without a Range header
119:246	HTTP in version field not all upper case
119:247	white space embedded in critical header value
119:248	gzip compressed data followed by unexpected non-gzip data
119:249	excessive HTTP parameter key repeats
119:253	HTTP CONNECT request with a message body

GID:SID	Rule Message
119:254	HTTP client-to-server traffic after CONNECT request but before CONNECT response
119:255	HTTP CONNECT 2XX response with Content-Length header
119:256	HTTP CONNECT 2XX response with Transfer-Encoding header
119:257	HTTP CONNECT response with 1XX status code
119:258	HTTP CONNECT response before request message completed
119:259	malformed HTTP Content-Disposition filename parameter
119:260	HTTP Content-Length message body was truncated
119:261	HTTP chunked message body was truncated
119:262	HTTP URI scheme longer than 10 characters
119:263	HTTP/1 client requested HTTP/2 upgrade
119:264	HTTP/1 server granted HTTP/2 upgrade
119:265	bad token in JavaScript
119:266	unexpected script opening tag in JavaScript
119:267	unexpected script closing tag in JavaScript
119:268	JavaScript code under the external script tags
119:269	script opening tag in a short form
119:270	max number of unique JavaScript identifiers
119:271	JavaScript bracket nesting is over capacity
119:272	Consecutive commas in HTTP Accept-Encoding header
119:273	missed PDUs during JavaScript normalization
119:274	JavaScript scope nesting is over capacity
119:275	HTTP/1 version other than 1.0 or 1.1e
119:276	HTTP version in start line is 0
119:277	HTTP version in start line is higher than 1

HTTP Inspect Inspector Intrusion Rule Options

http_client_body

Sets the detection cursor to the body of an HTTP request. When an HTTP message does not specify an HTTP header, Snort normalizes `http_client_body` using URI normalization. URI normalization is typically applied to `http_header`.

Syntax: `http_client_body;`

Examples: `http_client_body;`

http_cookie

Sets the detection cursor to the extracted HTTP `Cookie` header field. The `http_cookie` rule option includes the parameters: `http_cookie.request`, `http_cookie.with_header`, `http_cookie.with_body`, and `http_cookie.with_trailer`.

Syntax: http_cookie: <parameter>, <parameter>

Examples: `http_cookie: request;`

http_cookie.request

Matches the HTTP cookie found in the HTTP request message. Use the HTTP request cookie when examining the HTTP response. The `http_cookie.request` parameter is optional.

Syntax: `http_cookie: request;`

Examples: `http_cookie: request;`

http_cookie.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_cookie.with_header` parameter is optional.

Syntax: `http_cookie: with_header;`

Examples: `http_cookie: with_header;`

http_cookie.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_cookie` rule option. The `http_cookie.with_body` parameter is optional.

Syntax: `http_cookie: with_body;`

Examples: `http_cookie: with_body;`

http_cookie.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_cookie` rule option. The `http_cookie.with_trailer` parameter is optional.

Syntax: `http_cookie: with_trailer;`

Examples: `http_cookie: with_trailer;`

http_header

Sets the detection cursor to the normalized HTTP headers. You can specify individual header names using the `field` option.

The `http_header` rule option includes the parameters: `http_header.field`, `http_header.request`, `http_header.with_header`, `http_header.with_body`, and `http_header.with_trailer`.

Syntax: `http_header: field <field_name>,<parameter>, <parameter>`

Examples: `http_header: field Content-Type, with_trailer;`

http_header.field

Matches the specified header name to the normalized HTTP headers. The header name is case insensitive. If you do not specify a header name, the HTTP inspector examines all headers except the HTTP cookie headers (`Cookie` and `Set-Cookie`).

Type: string

Syntax: `http_header: field <field_name>;`

Valid values: An HTTP header name.

Examples: `http_header: field Content-Type;`

http_header.request

Matches the headers found in the HTTP request. Use the HTTP request headers when examining the HTTP response. The `http_header.request` parameter is optional.

Syntax: `http_header: request;`

Examples: `http_header: request;`

http_header.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_header.with_header` parameter is optional.

Syntax: `http_header: with_header;`

Examples: `http_header: with_header;`

http_header.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_header` rule option. The `http_header.with_body` parameter is optional.

Syntax: `http_header: with_body;`

Examples: `http_header: with_body;`

http_header.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_header` rule option. The `http_header.with_trailer` parameter is optional.

Syntax: `http_header: with_trailer;`

Examples: `http_header: with_trailer;`

http_method

Sets the detection cursor to the method of the HTTP request. The common HTTP request method values are `GET`, `POST`, `OPTIONS`, `HEAD`, `DELETE`, `PUT`, `TRACE`, and `CONNECT`.

The `http_method` rule option includes the parameters: `http_method.with_header`, `http_method.with_body`, and `http_method.with_trailer`.

Syntax: `http_method: <parameter>, <parameter>;`

Examples: `http_method; content:"GET";`

http_method.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_method.with_header` parameter is optional.

Syntax: `http_method: with_header;`

Examples: `http_method: with_header;`

http_method.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_header` rule option. The `http_method.with_body` parameter is optional.

Syntax: `http_method: with_body;`

Examples: `http_method: with_body;`

http_method.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_header` rule option. The `http_method.with_trailer` parameter is optional.

Syntax: `http_method: with_trailer;`

Examples: `http_method: with_trailer;`

http_param

Sets the detection cursor to the specified HTTP parameter key. The HTTP parameter key may appear in the query or request body.

The `http_param` rule option includes the parameters: `http_param.param` and `http_method.nocase`.

Syntax: `http_param: <parameter_key>, nocase;`

Examples: `http_param: offset, nocase;`

http_param.param

Matches the specified parameter.

Type: string

Syntax: `http_param: <http_parameter>;`

Valid values: A request query parameter or request body field.

Examples: `http_param: offset;`

http_param.nocase

Match the specified parameter, but do not consider case. The `http_param.nocase` parameter is optional.

Syntax: `http_param: nocase;`

Examples: `http_param: nocase;`

http_raw_body

Sets the detection cursor to the unnormalized request or response message body.

Syntax: `http_raw_body;`

Examples: `http_raw_body;`

http_raw_cookie

Sets the detection cursor to the unnormalized HTTP `Cookie` header. The `http_raw_cookie` rule option includes the parameters: `http_raw_cookie.request`, `http_raw_cookie.with_header`, `http_raw_cookie.with_body`, and `http_raw_cookie.with_trailer`.

Syntax: `http_raw_cookie: <parameter>, <parameter>;`

Examples: `http_raw_cookie: request;`

http_raw_cookie.request

Matches the cookie found in the HTTP request. Use the HTTP request cookie when examining the response message. The `http_raw_cookie.request` parameter is optional.

Syntax: `http_raw_cookie: request;`

Examples: `http_raw_cookie: request;`

http_raw_cookie.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_cookie.with_header` parameter is optional.

Syntax: `http_raw_cookie: with_header;`

Examples: `http_raw_cookie: with_header;`

http_raw_cookie.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_cookie` rule option. The `http_raw_cookie.with_body` parameter is optional.

Syntax: `http_raw_cookie: with_body;`

Examples: `http_raw_cookie: with_body;`

http_raw_cookie.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_cookie` rule option. The `http_raw_cookie.with_trailer` parameter is optional.

Syntax: `http_raw_cookie: with_trailer;`

Examples: `http_raw_cookie: with_trailer;`

http_raw_header

Sets the detection cursor to the unnormalized headers. `http_raw_header` includes all of the unmodified header names and values in the original message.

The `http_raw_header` rule option includes the parameters: `http_raw_header.field`, `http_raw_header.request`, `http_raw_header.with_header`, `http_raw_header.with_body`, and `http_raw_header.with_trailer`.

Syntax: `http_raw_header: field <field_name>, <parameter>, <parameter>;`

Examples: `http_raw_header: field Content-Type, with_trailer;`

http_raw_header.field

Matches the specified header name to the unnormalized HTTP headers. The header name is case insensitive. If you do not specify a header name, the HTTP inspector examines all headers except the HTTP cookie headers (Cookie and Set-Cookie).

Type: string

Syntax: `http_raw_header: field <field_name>`

Valid values: An HTTP header name.

Examples: `http_raw_header: field Content-Type;`

http_raw_header.request

Matches the headers found in the HTTP request message. Use the HTTP request headers when examining the response message. The `http_raw_header.request` parameter is optional.

Syntax: `http_raw_header: request;`

Examples: `http_raw_header: request;`

http_raw_header.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_header.with_header` parameter is optional.

Syntax: `http_raw_header: with_header;`

Examples: `http_raw_header: with_header;`

http_raw_header.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_header` rule option. The `http_raw_header.with_body` parameter is optional.

Syntax: `http_raw_header: with_body;`

Examples: `http_raw_header: with_body;`

http_raw_header.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_header` rule option. The `http_raw_header.with_trailer` parameter is optional.

Syntax: `http_raw_header: with_trailer;`

Examples: `http_raw_header: with_trailer;`

http_raw_request

Sets the detection cursor to the unnormalized request line. To examine a specific part of the first header line, use one of the following rule options: `http_method`, `http_raw_uri`, or `http_version`.

The `http_raw_request` rule option includes the parameters: `http_raw_request.with_header`, `http_raw_request.with_body`, and `http_raw_request.with_trailer`.

Syntax: `http_raw_request: <parameter>, <parameter>;`

Examples: `http_raw_request: with_header;`

http_raw_request.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_request.with_header` parameter is optional.

Syntax: `http_raw_request: with_header;`

Examples: `http_raw_request: with_header;`

http_raw_request.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_request` rule option. The `http_raw_request.with_body` parameter is optional.

Syntax: `http_raw_request: with_body;`

Examples: `http_raw_request: with_body;`

http_raw_request.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_request` rule option. The `http_raw_request.with_trailer` parameter is optional.

Syntax: `http_raw_request: with_trailer;`

Examples: `http_raw_request: with_trailer;`

http_raw_status

Sets the detection cursor to the unnormalized status line. To examine a specific part of the status line, use one of the following rule options: `http_version`, `http_stat_code`, or `http_stat_msg`.

The `http_raw_status` rule option includes the parameters: `http_raw_status.with_body` and `http_raw_status.with_trailer`.

Syntax: `http_raw_status: <parameter>, <parameter>;`

Examples: `http_raw_status: with_body;`

http_raw_status.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_status` rule option. The `http_raw_status.with_body` parameter is optional.

Syntax: `http_raw_status: with_body;`

Examples: `http_raw_status: with_body;`

http_raw_status.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_status` rule option. The `http_raw_status.with_trailer` parameter is optional.

Syntax: `http_raw_status: with_trailer;`

Examples: `http_raw_status: with_trailer;`

http_raw_trailer

Sets the detection cursor to the unnormalized HTTP trailers. Trailers contain information about the message content. The trailers are not available when the client request creates HTTP headers.

`http_raw_trailer` is identical to `http_raw_header`, except that it applies to the end headers. You must create separate rules to inspect the HTTP headers and trailers.

The `http_raw_trailer` rule option includes the parameters: `http_raw_trailer.field`, `http_raw_trailer.request`, `http_raw_trailer.with_header`, `http_raw_trailer.with_body`.

Syntax: `http_raw_trailer: field <field_name>, <parameter>, <parameter>;`

Examples: `http_raw_trailer: field <field_name>, request;`

http_raw_trailer.field

Matches the specified trailer name to the unnormalized HTTP trailers. The trailer name is case insensitive.

Type: string

Syntax: `http_raw_trailer: field <field_name>;`

Valid values: An HTTP trailer name.

Examples: `http_raw_trailer: field trailer-timestamp;`

http_raw_trailer.request

Matches the trailers found in the HTTP request message. Use the HTTP request trailers when examining the response message. The `http_raw_trailer.request` parameter is optional.

Syntax: `http_raw_trailer: request;`

Examples: `http_raw_trailer: request;`

http_raw_trailer.with_header

Specifies that the rule can only examine the HTTP response headers. The `http_raw_trailer.with_header` parameter is optional.

Syntax: `http_raw_trailer: with_header;`

Examples: `http_raw_trailer: with_header;`

http_raw_trailer.with_body

Specifies that another part of the rule examines the HTTP response message body, not the `http_raw_trailer` rule option. The `http_raw_trailer.with_body` parameter is optional.

Syntax: `http_raw_trailer: with_body;`

Examples: `http_raw_trailer: with_body;`

http_raw_uri

Sets the detection cursor to the unnormalized URI.

The `http_raw_uri` rule option includes:

- `http_raw_uri.with_header`
- `http_raw_uri.with_body`
- `http_raw_uri.with_trailer`
- `http_raw_uri.scheme`
- `http_raw_uri.host`
- `http_raw_uri.port`
- `http_raw_uri.path`
- `http_raw_uri.query`
- `http_raw_uri.fragment`

Syntax: `http_raw_uri: <parameter>, <parameter>;`

Examples: `http_raw_uri: with_header, path, query;`

http_raw_uri.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_uri.with_header` parameter is optional.

Syntax: `http_raw_uri: with_header;`

Examples: `http_raw_uri: with_header;`

http_raw_uri.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_uri` rule option. The `http_raw_uri.with_body` parameter is optional.

Syntax: `http_raw_uri: with_body;`

Examples: `http_raw_uri: with_body;`

http_raw_uri.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_uri` rule option. The `http_raw_uri.with_trailer` parameter is optional.

Syntax: `http_raw_uri: with_trailer;`

Examples: `http_raw_uri: with_trailer;`

http_raw_uri.scheme

Matches only against the scheme of the URI. The `http_raw_uri.scheme` parameter is optional.

Syntax: `http_raw_uri: scheme;`

Examples: `http_raw_uri: scheme;`

http_raw_uri.host

Matches only against the host (domain name) of the URI. The `http_raw_uri.host` parameter is optional.

Syntax: `http_raw_uri: host;`

Examples: `http_raw_uri: host;`

http_raw_uri.port

Matches only against the port (TCP port) of the URI. The `http_raw_uri.port` parameter is optional.

Syntax: `http_raw_uri: port;`

Examples: `http_raw_uri: port;`

http_raw_uri.path

Matches only against the path section (directory and file) of the URI. The `http_raw_uri.path` parameter is optional.

Syntax: `http_raw_uri: path;`

Examples: `http_raw_uri: path;`

http_raw_uri.query

Matches only against the query parameters in the URI. The `http_raw_uri.query` parameter is optional.

Syntax: `http_raw_uri: query;`

Examples: `http_raw_uri: query;`

http_raw_uri.fragment

Matches only against the fragment section of the URI. A fragment is part of the file requested, normally found only inside a browser and not transmitted over the network. The `http_raw_uri.fragment` parameter is optional.

Syntax: `http_raw_uri: fragment;`

Examples: `http_raw_uri: fragment;`

http_stat_code

Sets the detection cursor to the HTTP status code. The HTTP status code is a three-digit number ranging between 100 – 599.

The `http_stat_code` rule option includes the parameters: `http_stat_code.with_body` and `http_stat_code.with_trailer`.

Syntax: `http_stat_code: <parameter>, <parameter>;`

Examples: `http_stat_code: with_trailer;`

http_stat_code.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_stat_code` rule option. The `http_stat_code.with_body` parameter is optional.

Syntax: `http_stat_code: with_body;`

Examples: `http_stat_code: with_body;`

http_stat_code.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_stat_code` rule option. The `http_stat_code.with_trailer` parameter is optional.

Syntax: `http_stat_code: with_trailer;`

Examples: `http_stat_code: with_trailer;`

http_stat_msg

Sets the detection cursor to the HTTP status message. The HTTP status message describes the HTTP status code in plain text, for example: `OK`.

The `http_stat_msg` rule option includes the parameters: `http_stat_msg.with_body` and `http_stat_msg.with_trailer`.

Syntax: `http_stat_msg: <parameter>, <parameter>;`

Examples: `http_stat_msg: with_body;`

http_stat_msg.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_stat_msg` rule option. The `http_stat_msg.with_body` parameter is optional.

Syntax: `http_stat_msg: with_body;`

Examples: `http_stat_msg: with_body;`

http_stat_msg.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_stat_msg` rule option. The `http_stat_msg.with_trailer` parameter is optional.

Syntax: `http_stat_msg: with_trailer;`

Examples: `http_stat_msg: with_trailer;`

http_trailer

Sets the detection cursor to the normalized trailers. Trailers contain information about the message content. The trailers are not available when the client request creates HTTP headers.

`http_trailer` is identical to `http_header`, except that it applies to the end headers. You must create separate rules to inspect the HTTP headers and trailers.

The `http_trailer` rule option includes the parameters: `http_trailer.field`, `http_trailer.request`, `http_trailer.with_header`, `http_trailer.with_body`.

Syntax: `http_trailer: field <field_name>, <parameter>, <parameter>;`

Examples: `http_trailer: field trailer-timestamp, with_body;`

http_trailer.field

Matches the specified trailer name to the normalized HTTP trailers. The trailer name is case insensitive.

Type: string

Syntax: `http_trailer: field <field_name>;`

Valid values: An HTTP trailer name.

Examples: `http_trailer: field trailer-timestamp;`

http_trailer.request

Matches the trailers found in the HTTP request message. Use the HTTP request trailers when examining the response message. The `http_trailer.request` parameter is optional.

Syntax: `http_trailer: request;`

Examples: `http_trailer: request;`

http_trailer.with_header

Specifies that another part of the rule examines the HTTP message headers, not the `http_trailer` rule option. The `http_trailer.with_header` parameter is optional.

Syntax: `http_trailer: with_header;`

Examples: `http_trailer: with_header;`

http_trailer.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_trailer` rule option. The `http_trailer.with_body` parameter is optional.

Syntax: `http_trailer: with_body;`

Examples: `http_trailer: with_body;`

http_true_ip

Sets the detection cursor to the final client IP address. When a client sends a request, the proxy server stores the final client IP address. A client IP address is the last IP address listed in the `X-Forwarded-For`, `True-Client-IP`, or any other custom `X-Forwarded-For` type header. If multiple headers are present, Snort considers the headers defined in `xff_headers`.

The `http_true_ip` rule option includes the parameters: `http_true_ip.with_header`, `http_true_ip.with_body`, and `http_true_ip.with_trailer`.

Syntax: `http_true_ip: <parameter>, <parameter>;`

Examples: `http_true_ip: with_header;`

http_true_ip.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_true_ip.with_header` parameter is optional.

Syntax: `http_true_ip: with_header;`

Examples: `http_true_ip: with_header;`

http_true_ip.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_true_ip` rule option. The `http_true_ip.with_body` parameter is optional.

Syntax: `http_true_ip: with_body;`

Examples: `http_true_ip: with_body;`

http_true_ip.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_true_ip` rule option. The `http_true_ip.with_trailer` parameter is optional.

Syntax: `http_true_ip: with_trailer;`

Examples: `http_true_ip: with_trailer;`

http_uri

Sets the detection cursor to the normalized URI buffer.

- `http_uri.with_header`
- `http_uri.with_body`
- `http_uri.with_trailer`
- `http_uri.scheme`
- `http_uri.host`
- `http_uri.port`
- `http_uri.path`
- `http_uri.query`
- `http_uri.fragment`

Syntax: `http_uri: <parameter>, <parameter>;`

Examples: `http_uri: with_trailer, path, query;`

http_uri.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_uri.with_header` parameter is optional.

Syntax: `http_uri: with_header;`

Examples: `http_uri: with_header;`

http_uri.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_uri` rule option. The `http_uri.with_body` parameter is optional.

Syntax: `http_uri: with_body;`

Examples: `http_uri: with_body;`

http_uri.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_uri` rule option. The `http_uri.with_trailer` parameter is optional.

Syntax: `http_uri: with_trailer;`

Examples: `http_uri: with_trailer;`

http_uri.scheme

Matches only against the scheme of the URI. The `http_uri.scheme` parameter is optional.

Syntax: `http_uri: scheme;`

Examples: `http_uri: scheme;`

http_uri.host

Matches only against the host (domain name) of the URI. The `http_uri.host` parameter is optional.

Syntax: `http_uri: host;`

Examples: `http_uri: host;`

http_uri.port

Matches only against the port (TCP port) of the URI. The `http_uri.port` parameter is optional.

Syntax: `http_uri: port;`

Examples: `http_uri: port;`

http_uri.path

Matches only against the path (directory and file) of the URI. The `http_uri.path` parameter is optional.

Syntax: `http_uri: path;`

Examples: `http_uri: path;`

http_uri.query

Matches only against the query parameters in the URI. The `http_uri.query` parameter is optional.

Syntax: `http_uri: uri;`

Examples: `http_uri: query;`

http_uri.fragment

Matches only against the fragment section of the URI. A fragment is part of the file requested, normally found only inside a browser and not transmitted over the network. The `http_uri.fragment` parameter is optional.

Syntax: `http_uri: fragment;`

Examples: `http_uri: fragment;`

http_version

Sets the detection cursor to the beginning of the HTTP version buffer. `http_version` accepts various HTTP versions. The most commonly found versions are: `HTTP/1.0` and `HTTP/1.1`. The `http_version` rule option includes the parameters: `http_version.request`, `http_version.with_header`, `http_version.with_body`, and `http_version.with_trailer`.

Syntax: `http_version: <parameter>, <parameter>;`

Examples: `http_version; content:"HTTP/1.1";`

http_version.request

Matches the version found in the HTTP request. Use the request version when examining the response message. The `http_version.request` parameter is optional.

Syntax: `http_version: request;`

Examples: `http_version: request;`

http_version.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_version.with_header` parameter is optional.

Syntax: `http_version: with_header;`

Examples: `http_version: with_header;`

http_version.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_version` rule option. The `http_version.with_body` parameter is optional.

Syntax: `http_version: with_body;`

Examples: `http_version: with_body;`

http_version.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_version` rule option. The `http_version.with_trailer` parameter is optional.

Syntax: `http_version: with_trailer;`

Examples: `http_version: with_trailer;`

http_version_match

Specifies a list of HTTP versions to match against the standard HTTP versions. Separate multiple versions with a space character. An HTTP request or status line may contain a version. If the version is present, Snort compares this version with the list specified in `http_version_match`.

If the version doesn't have a format of `[0-9].[0-9]` it is considered malformed. A version in the format of `[0-9].[0-9]` that is not `1.0` or `1.1` is considered `other`.

Type: string

Syntax: `http_version_match: <version_list>`

Valid values: `1.0, 1.1, 2.0, 0.9, other, malformed`

Examples: `http_version_match: "1.0 1.1";`

js_data

Sets the detection cursor to the normalized JavaScript data. This option is specific to the enhanced JavaScript normalizer.

Syntax: `js_data;`

Examples: `js_data;`

vba_data

Sets the detection cursor to the Microsoft Office Visual Basic for Applications macros buffer.

Syntax: `vba_data;`

Examples: `vba_data;`

CHAPTER 12

IEC104 Inspector

IEC104 Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

The IEC 60870-5-104 (IEC104) protocol describes a communication standard to exchange telecontrol messages between electric power systems. The IEC104 protocol uses TCP port 2404.

The `iec104` inspector detects IEC104 messages in network traffic. The `iec104` inspector analyzes and normalizes IEC104 messages by either combining a message spread across multiple frames, or splitting apart multiple messages within one frame.

When enabled, the intrusion rule options provide access to the IEC104 application protocol control information (APCI) type and the application service data unit (ASDU) function code.

IEC104 Inspector Parameters

IEC104 TCP port configuration

The `binder` inspector defines the IEC104 TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "role": "server",
            "proto": "tcp",
            "ports": "2404"
         },
        "use": {
            "type": "iec104"
        }
    }
]
```

Note The `iec104` inspector does not provide any parameters.

IEC104 Inspector Rules

Enable the `iec104` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 15: IEC104 Inspector Rules

GID:SID	Rule Message
151:1	Length in IEC104 APCI header does not match the length needed for the given IEC104 ASDU type id
151:2	IEC104 Start byte does not match 0x68
151:3	Reserved IEC104 ASDU type id in use
151:4	IEC104 APCI U Reserved field contains a non-default value
151:5	IEC104 APCI U message type was set to an invalid value
151:6	IEC104 APCI S Reserved field contains a non-default value
151:7	IEC104 APCI I number of elements set to zero
151:8	IEC104 APCI I SQ bit set on an ASDU that does not support the feature
151:9	IEC104 APCI I number of elements set to greater than one on an ASDU that does not support the feature
151:10	IEC104 APCI I Cause of Initialization set to a reserved value
151:11	IEC104 APCI I Qualifier of Interrogation Command set to a reserved value
151:12	IEC104 APCI I Qualifier of Counter Interrogation Command request parameter set to a reserved value
151:13	IEC104 APCI I Qualifier of Parameter of Measured Values kind of parameter set to a reserved value

GID:SID	Rule Message
151:14	IEC104 APCI I Qualifier of Parameter of Measured Values local parameter change set to a technically valid but unused value
151:15	IEC104 APCI I Qualifier of Parameter of Measured Values parameter option set to a technically valid but unused value
151:16	IEC104 APCI I Qualifier of Parameter Activation set to a reserved value
151:17	IEC104 APCI I Qualifier of Command set to a reserved value
151:18	IEC104 APCI I Qualifier of Reset Process set to a reserved value
151:19	IEC104 APCI I File Ready Qualifier set to a reserved value
151:20	IEC104 APCI I Section Ready Qualifier set to a reserved value
151:21	IEC104 APCI I Select and Call Qualifier set to a reserved value
151:22	IEC104 APCI I Last Section or Segment Qualifier set to a reserved value
151:23	IEC104 APCI I Acknowledge File or Section Qualifier set to a reserved value
151:24	IEC104 APCI I Structure Qualifier set on a message where it should have no effect
151:25	IEC104 APCI I Single Point Information Reserved field contains a non-default value
151:26	IEC104 APCI I Double Point Information Reserved field contains a non-default value
151:27	IEC104 APCI I Cause of Transmission set to a reserved value
151:28	IEC104 APCI I Cause of Transmission set to a value not allowed for the ASDU
151:29	IEC104 APCI I invalid two octet common address value detected
151:30	IEC104 APCI I Quality Descriptor Structure Reserved field contains a non-default value
151:31	IEC104 APCI I Quality Descriptor for Events of Protection Equipment Structure Reserved field contains a non-default value
151:32	IEC104 APCI I IEEE STD 754 value results in NaN
151:33	IEC104 APCI I IEEE STD 754 value results in infinity
151:34	IEC104 APCI I Single Event of Protection Equipment Structure Reserved field contains a non-default value
151:35	IEC104 APCI I Start Event of Protection Equipment Structure Reserved field contains a non-default value
151:36	IEC104 APCI I Output Circuit Information Structure Reserved field contains a non-default value
151:37	IEC104 APCI I Abnormal Fixed Test Bit Pattern detected

GID:SID	Rule Message
151:38	IEC104 APCI I Single Command Structure Reserved field contains a non-default value
151:39	IEC104 APCI I Double Command Structure contains an invalid value
151:40	IEC104 APCI I Regulating Step Command Structure Reserved field contains a non-default value
151:41	IEC104 APCI I Time2a Millisecond set outside of the allowable range
151:42	IEC104 APCI I Time2a Minute set outside of the allowable range
151:43	IEC104 APCI I Time2a Minute Reserved field contains a non-default value
151:44	IEC104 APCI I Time2a Hours set outside of the allowable range
151:45	IEC104 APCI I Time2a Hours Reserved field contains a non-default value
151:46	IEC104 APCI I Time2a Day of Month set outside of the allowable range
151:47	IEC104 APCI I Time2a Month set outside of the allowable range
151:48	IEC104 APCI I Time2a Month Reserved field contains a non-default value
151:49	IEC104 APCI I Time2a Year set outside of the allowable range
151:50	IEC104 APCI I Time2a Year Reserved field contains a non-default value
151:51	IEC104 APCI I a null Length of Segment value has been detected
151:52	IEC104 APCI I an invalid Length of Segment value has been detected
151:53	IEC104 APCI I Status of File set to a reserved value
151:54	IEC104 APCI I Qualifier of Set Point Command ql field set to a reserved value

IEC104 Inspector Intrusion Rule Options

iec104_apci_type

Verifies that the IEC104 message matches the IEC104 application protocol information control (APIC) type set in the option.

The `iec104_apci_type` intrusion rule option accepts a string specified using the full APIC type name, or uppercase or lowercase APIC type abbreviation.

Type: string

Syntax: `iec104_apci_type: <apic_type>;`

Examples:

```
iec104_apci_type: unnumbered_control_function;
```

```
iec104_apci_type: S;
```

```
iec104_apci_type: I;
```

```
iec104_apci_type: i;
```

iec104_asdu_func

Verifies that the IEC104 message matches the IEC104 application service data unit (ASDU) function code set in the option.

The `iec104_asdu_func` intrusion rule option accepts a string specified using the uppercase or lowercase ASDU function code.

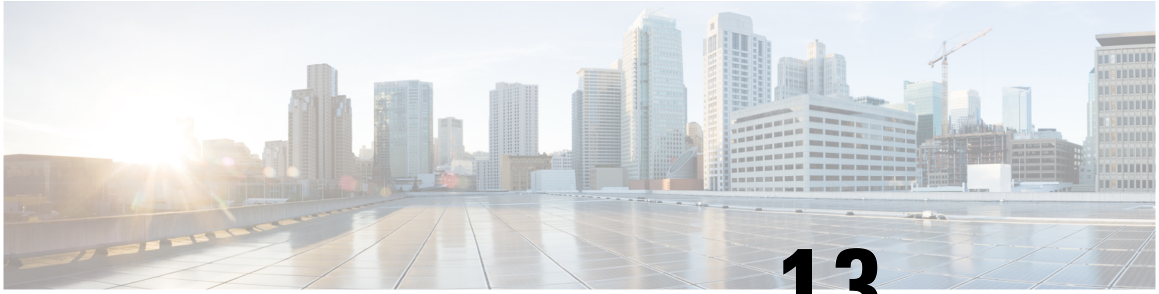
Type: string

Syntax: `iec104_asdu_func: <asdu_func>;`

Examples:

```
iec104_asdu_func: M_SP_NA_1;
```

```
iec104_asdu_func: m_sp_na_1;
```



CHAPTER 13

IMAP Inspector

IMAP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`true`

Internet Message Application Protocol (IMAP) enables email clients to retrieve messages from a remote IMAP3 server. An IMAP3 server uses TCP port 143 for insecure sessions or TCP port 993 for IMAP over SSL/TLS.

The `imap` inspector detects IMAP traffic and analyzes IMAP commands and responses.

The `imap` inspector can identify the command, header, and body sections of IMAP messages, and extract and decode multi-purpose internet mail extensions (MIME) attachments. MIME attachments may include multiple attachments and large attachments that span multiple packets.

The `imap` inspector identifies and adds IMAP traffic to the Snort allow list. When enabled, intrusion rules generate events on anomalous IMAP traffic.

IMAP Inspector Parameters

IMAP service configuration

The `binder` inspector defines the IMAP `service` configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "service": "imap",
            "role": any
        },
        "use": {
            "type": "imap"
        }
    }
]
```

b_64_decode_depth

Specifies the maximum number of bytes to extract and decode from each Base64 encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 141:4 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data).

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

bitenc_decode_depth

Specifies the maximum number of bytes to extract from each non-encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable the extraction of the non-encoded MIME attachment. Specify -1 to place no limit on the number of bytes to extract. These attachment types include 7-bit, 8-bit, binary, and various multipart content types such as plain text, JPEG and PNG images, and MP4 files.

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

decompress_pdf

Specifies whether to decompress `application/pdf` (PDF) files in MIME attachments.

You can enable rule 141:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_swf

Specifies whether to decompress `application/vnd.adobe.flash-movie` (SWF) files in MIME attachments.

You can enable rule 141:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: integer

Valid values: `true`, `false`

Default value: `false`

decompress_vba

Specifies whether to decompress Microsoft Office Visual Basic for Applications macro files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_zip

Specifies whether to decompress `application/zip` (ZIP) files in MIME attachments.

You can enable rule 141:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

qp_decode_depth

Specifies the maximum number of bytes to extract and decode from each quoted-printable (QP) encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 141:5 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data).

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

uu_decode_depth

Specifies the maximum number of bytes to extract and decode from each Unix-to-Unix encoded (uuencoded) MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 141:7 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data).

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

IMAP Inspector Rules

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

Table 16: IMAP Inspector Rules

GID:SID	Rule Message
141:1	unknown IMAP3 command
141:2	unknown IMAP3 response
141:4	base64 decoding failed
141:5	quoted-printable decoding failed
141:7	Unix-to-Unix decoding failed
141:8	file decompression failed

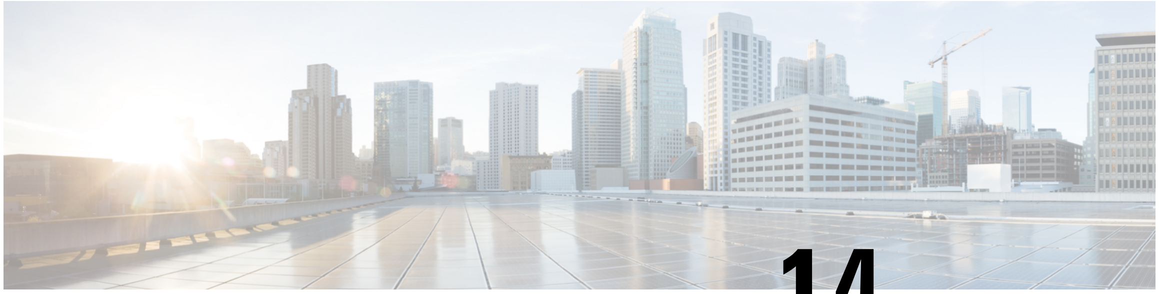
IMAP Inspector Intrusion Rule Options

vba_data

Sets the detection cursor to the Microsoft Office Visual Basic for Applications macros buffer.

Syntax: `vba_data;`

Examples: `vba_data;`



CHAPTER 14

MMS Inspector

MMS Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

IEC 61850 is an international standard that defines communication protocols for electric power systems. The Manufacturing Message Specification (MMS) protocol is one of the IEC 61850 protocols. MMS enables the real-time transfer of Supervisory Control and Data Acquisition (SCADA) data between various manufacturing and process control devices. The MMS protocol uses TCP port 102 to exchange messages between client and server devices.

The `mms` inspector detects and analyzes MMS traffic. MMS messages may include multiple Protocol Data Units (PDUs) within one TCP packet, one PDU split across multiple TCP packets, or a combination of the two message configurations. The `mms` inspector normalizes the MMS traffic to present complete MMS messages to a device.

You write Snort 3 rules for MMS messages without decoding the MMS protocol. The `mms` inspector analyzes the OSI layers that encapsulate the MMS protocol, and provides access to certain MMS protocol fields and data content through rule options. For information about the MMS rule options, see MMS Inspector Intrusion Rule Options, on page 120

MMS Inspector Parameters

MMS service configuration

The binder inspector defines the MMS service configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "service": "mms"
        },
        "use": {
            "type": "mms"
        }
    }
]
```

MMS Inspector Rules

The mms inspector does not have any associated rules.

MMS Inspector Intrusion Rule Options

mms_data

Sets the detection cursor position to the start of the MMS Protocol Data Unit (PDU), bypassing all of the OSI encapsulation layers. When an intrusion rule includes mms_data, the next rule options in the rule begin processing from the MMS PDU.

Syntax: mms_data;

Examples:

The following sample intrusion rule sets the mms_data rule option. The mms_data rule option positions the detection cursor to the start of the MMS PDU, and checks the byte at that position for the value of an Initiate-Request message.

```
alert tcp ( \
msg: "PROTOCOL-SCADA MMS Initiate-Request"; \
flow: to_server, established; \
mms_data; \
content:"|A8|", depth 1; \
sid:1000000; \
)
```

mms_func

Compares the provided function name or number with the Confirmed Service field in the MMS request or response. Alert when the MMS function name or number matches the Confirmed Service.

Type: `string`

Syntax: `mms_func <function>;`

Examples:

The following sample intrusion rule sets the `mms_func` rule option and alerts when the `Confirmed Service Request` service matches the provided function name. In addition, `mms_func` enables the fast pattern matching feature to match on the `Confirmed Service Request (0xA0)` message.

```
alert tcp ( \
msg: "PROTOCOL-SCADA MMS svc get_name_list"; \
flow: to_server, established; \
content:"|A0|"; \
mms_func: get_name_list; \
sid:1000000; \
)
```

The following sample intrusion rule sets the `mms_func` rule option and alerts when the `GetNameList` message matches the function number.

```
alert tcp ( \
msg: "PROTOCOL-SCADA MMS svc get_name_list"; \
flow: to_server, established; \
content:"|A0|"; \
mms_func:1; \
sid:1000001; \
)
```

CHAPTER 15

Modbus Inspector

Modbus Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

The Modbus protocol defines a communication standard to exchange messages between a Supervisory Control and Data Acquisition (SCADA) system and a Programmable Automation Controller (PLC). The Modbus protocol uses TCP port 502.

The `modbus` inspector detects and analyzes Modbus messages in network traffic. When enabled, the Modbus intrusion rule options provide access to certain Modbus protocol fields.

Best Practices for Configuring the Modbus Inspector

If your network does not contain an enabled Modbus device, you should not enable the `modbus` inspector in a network analysis policy that you apply to traffic.

Modbus Inspector Parameters

Modbus TCP port configuration

The `binder` inspector defines the Modbus TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "role": "server",
            "proto": "tcp",
            "ports": "502"
        },
        "use": {
            "type": "modbus"
        }
    },
    {
        "when": {
            "role": "any",
            "service:" "modbus"
        },
        "use": {
            "type":"modbus"
        }
    }
]
```

Note The `modbus` inspector does not provide any parameters.

Modbus Inspector Rules

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

Table 17: Modbus Inspector Rules

GID:SID	Rule Message
144:1	length in Modbus MBAP header does not match the length needed for the given function
144:2	Modbus protocol ID is non-zero
144:3	reserved Modbus function code in use

Modbus Inspector Intrusion Rule Options

You can use a `modbus` option alone or in combination with the `content` and `byte_jump` intrusion rule options.

modbus_data

Sets the data cursor to the beginning of the Modbus `Data` field.

Syntax: `modbus_data;`

Examples: `modbus_data;`

modbus_func

Verifies that the Modbus `Function` field matches the specified Modbus function code. You can set a positive integer or string literal to represent a Modbus function code.

Type: string

Syntax: `modbus_func: <function>;`

Valid values:

Table 18: Modbus Function Code Values

Code	String
1	read_coils
2	read_discrete_inputs
3	read_holding_registers
4	read_input_registers
5	write_single_coil
6	write_single_register
7	read_exception_status
8	diagnostics
11	get_comm_event_counter
12	get_comm_event_log
15	write_multiple_coils
16	write_multiple_registers
17	report_slave_id
20	read_file_record
21	write_file_record

Code	String
22	mask_write_register
23	read_write_multiple_registers
24	read_fifo_queue
43	encapsulated_interface_transport

Examples:

```
modbus_func: read_coils;
```

```
modbus_func: 8;
```

modbus_unit

Verifies that the Modbus Unit ID in the message matches the specified unit ID. You can set a number to represent the Modbus Unit ID.

Type: integer

Syntax: `modbus_unit: <unit_id>;`

Valid range: `0` to `255`

Examples:

```
modbus_unit: 1;
```

CHAPTER 16

Normalizer Inspector

Normalizer Inspector Overview

Type	Inspector (packet)
Usage	Context
Instance Type	Network
Other Inspectors Required	None
Enabled	`true`

The `normalizer` inspector detects and removes protocol anomalies in packets. The `normalizer` inspector can minimize the chances of attackers creating packets to evade detection in inline deployments.

Note Before your send traffic from your network, you must deploy relevant configurations to managed devices using routed, switched, or transparent interfaces, or inline interface pairs.

You can specify the normalization of any combination of IPv4, IPv6, ICMPv4, ICMPv6, and TCP protocols in packets. The `normalizer` inspector conducts per-packet normalizations and handles most normalizations. The `stream_tcp` inspector handles TCP state-related packet and stream normalizations, including TCP payload normalization.

Inline normalization takes place immediately after decoding and before processing by other inspectors. Normalization proceeds from the inner to outer packet layers.

The `normalizer` inspector does not generate events. The `normalizer` inspector prepares packets for use by other inspectors and in inline deployments. The inspector helps ensure that the packets the system processes are the same as the packets received by the hosts on your network.

Normalizer Inspector Parameters

Locate the `normalizer` scope in your configuration to set the `normalizer` inspector parameters.

ip6

Clears the `Reserved` flag in IPv6 traffic.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

icmp4

Clears the `Reserved` flag in ICMPv4 traffic.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

icmp6

Clears the `Reserved` flag in ICMPv6 traffic.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ip4.base

Clears the single-bit `Reserved` subfield of the IPv4 Flags header field as well as parameter padding. Fixes urgent pointer/flag issues. We recommend that you enable `ip4.base`.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ip4.df

Clears the single-bit `Don’t Fragment` subfield of the IPv4 Flags header field. Enable `ip4.df` to allow a downstream router to fragment packets instead of dropping them. The `ip4.df` parameter can prevent evasions which create packets to be dropped.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ip4.rf

Clears the `Reserved` bits on incoming packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ip4.tos

Clears the one byte `Differentiated Services` field, formerly known as `Type of Service`.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ip4.trim

Truncates packets with excess payload to the datagram length specified in the IP header plus the Layer 2 (for example, Ethernet) header, but does not truncate below the minimum frame length.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.base

Clears the single-bit `Reserved` subfield of the TCP header as well as option padding bytes. Fixes urgent pointer or flag issues.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.block

Specfies whether to drop packets during TCP normalization.

When enabled, Snort blocks anomalous TCP packets that, if normalized, would be invalid and likely would be blocked by the receiving host. For example, Snort blocks any SYN packet transmitted subsequent to an established session.

Snort drops any packet that matches any of the following TCP stream inspector rules, regardless of whether the rules are enabled:

- 129:1
- 129:3
- 129:4
- 129:6
- 129:8

- 129:11
- 129:14 through 129:19

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.ecn

Enables per-packet or per-stream normalization of `Explicit Congestion Notification (ECN)` flags.

- Specify `packet` to clear ECN flags on a per-packet basis regardless of negotiation.
- Specify `stream` to clear ECN flags on a per-stream basis if ECN use was not negotiated. If you specify `stream`, you must enable `tcp.require_3whs` in the TCP stream inspector for normalization to take place.
- Specify `off` to disable the `tcp.ecn` parameter.

Type: enum

Valid values: `off`, `packet`, `stream`

Default value: `off`

tcp.ips

Enables normalization of the TCP Data field to ensure consistency in retransmitted data. Any segment that cannot be properly reassembled is dropped.

Type: boolean

Valid values: `true`, `false`

Default value: `true`

tcp.opts

Specifies whether to normalize specific TCP options which you allow in traffic. Snort does not normalize options that you explicitly allow. Snort normalizes options that you do not explicitly allow.

Snort always allows the following TCP options because they are commonly used for optimal TCP performance:

- Maximum Segment Size (MSS)
- Window Scale
- Time Stamp TCP

Snort does not automatically allow other less commonly used options.

When `tcp.opts` is enabled, TCP traffic normalizations include the following:

- Sets all option bytes to No Operation (TCP Option 1), except for MSS, Window Scale, Time Stamp, and any explicitly allowed options.
- Sets the Time Stamp octets to No Operation if Time Stamp is present but invalid, or valid but not negotiated.

- Blocks the packet if Time Stamp is negotiated but not present
- Clears the Time Stamp Echo Reply (TSecr) option field if the Acknowledgment (ACK) control bit is not set.
- Sets the MSS and Window Scale options to No Operation (TCP Option 1) if the SYN control bit is not set.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.pad

Clears any option padding bytes.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.req_pay

Clears the TCP header `Urgent Pointer` field and the urgent (URG) control bit if there is no payload.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.req_urg

Clears the 16-bit TCP header `Urgent Pointer` field if the TCP header urgent (URG) control bit is not set.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.req_urp

Clears the TCP header `urgent (URG)` control bit if the TCP header `Urgent Pointer` field is not set.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.resv

Clears the `Reserved` bits in the TCP header.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.trim_mss

Trims the TCP `Data` field to the Maximum Segment Size (MSS) if the payload is longer than MSS.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.trim_rst

Clears data from the RST packet.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.trim_syn

Removes data in TCP synchronization (SYN) packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.trim_win

Trims the TCP `Data` field to the size specified in the `Window` field.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp.urp

Sets the two-byte TCP header `Urgent Pointer` field to the payload length if the pointer is greater than the payload length.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

Normalizer Inspector Rules

The `normalizer` inspector does not have any associated rules.

Normalizer Inspector Intrusion Rule Options

The `normalizer` inspector does not have any intrusion rule options.

CHAPTER 17

POP Inspector

POP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`true`

Post Office Protocol version 3 (POP3) enables email clients to retrieve messages from a remote POP3 server. A POP3 server uses TCP port 110 for insecure sessions or TCP port 995 for POP over SSL/TLS.

The `pop` inspector detects POP traffic and analyzes POP commands and responses.

The `pop` inspector can identify the command, header, and body sections of POP messages, and extract and decode multi-purpose internet mail extensions (MIME) attachments. The `pop` inspector processes MIME attachments, including multiple attachments and large attachments that span multiple packets.

The `pop` inspector identifies and adds POP messages to the Snort allow list. When enabled, intrusion rules generate events on anomalous POP traffic.

POP Inspector Parameters

Note Decoding, or extraction when the MIME email attachment does not require decoding, can include multiple attachments and large attachments that span multiple packets.

The highest value is used when the values for the `b_64_decode_depth`, `bitenc_decode_depth`, `qp_decode_depth`, or `uu_decode_depth` parameters are different in:

- the default network analysis policy
- any other custom network analysis policy invoked by network analysis rules in the same access control policy

POP service configuration

The `binder` inspector defines the POP `service` configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
           "service": "pop",
           "role": any
        },
        "use": {
           "type": "pop"
        }
    }
]
```

b_64_decode_depth

Specifies the maximum number of bytes to extract and decode from each Base64 encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 142:4 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails.

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

bitenc_decode_depth

Specifies the maximum number of bytes to extract from each non-encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable the extraction of the non-encoded MIME attachment. Specify -1 to place no limit on the number of bytes to extract. These attachment types include 7-bit, 8-bit, binary, and various multipart content types such as plain text, JPEG and PNG images, and MP4 files.

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

decompress_pdf

Specifies whether to decompress `application/pdf` (PDF) files in MIME attachments.

You can enable rule 142:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_swf

Specifies whether to decompress `application/vnd.adobe.flash-movie` (SWF) files in MIME attachments.

You can enable rule 142:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_vba

Specifies whether to decompress Microsoft Office Visual Basic for Applications macro files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_zip

Specifies whether to decompress `application/zip` (ZIP) files in MIME attachments.

You can enable rule 142:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

qp_decode_depth

Specifies the maximum number of bytes to extract and decode from each quoted-printable (QP) encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 142:5 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data).

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

uu_decode_depth

Specifies the maximum number of bytes to extract and decode from each Unix-to-Unix encoded (uuencoded) MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 142:7 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data).

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

POP Inspector Rules

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

Table 19: POP Inspector Rules

GID:SID	Rule Message
142:1	unknown POP3 command
142:2	unknown POP3 response
142:4	base64 decoding failed
142:5	quoted-printable decoding failed
142:7	Unix-to-Unix decoding failed
142:8	file decompression failed

POP Inspector Intrusion Rule Options

vba_data

Sets the detection cursor to the Microsoft Office Visual Basic for Applications macros buffer.

Syntax: `vba_data;`

Examples: `vba_data;`

CHAPTER 18

Port Scan Inspector

Port Scan Inspector Overview

Type	Inspector (probe)
Usage	Global
Instance Type	Global
Other Inspectors Required	None
Enabled	false

A port scan is a form of network reconnaissance that is often used by attackers as a prelude to an attack. In a port scan, an attacker sends packets designed to probe for network protocols and services on a targeted host. By examining the packets sent in response by a host, the attacker can determine which ports are open on the host and, either directly or by inference, which application protocols are running on these ports.

By itself, a port scan is not evidence of an attack. Legitimate users on your network may employ similar port scanning techniques used by attackers.

The `port_scan` inspector detects four types of portscan and monitors connection attempts on TCP, UDP, ICMP, and IP protocols. By detecting patterns of activity, the `port_scan` inspector helps you determine which port scans might be malicious.

Table 20: Portscan Protocol Types

Protocol	Description
TCP	Detects TCP probes such as SYN scans, ACK scans, TCP connect() scans, and scans with unusual flag combinations such (Xmas tree, FIN, and NULL).
UDP	Detects UDP probes such as zero-byte UDP packets.

Protocol	Description
ICMP	Detects ICMP echo requests (pings).
IP	Detects IP protocol scans. Instead of looking for open ports, Snort searches for IP protocols which are supported on a target host.

Port scans are generally divided into four types based on the number of targeted hosts, the number of scanning hosts, and the number of ports that are scanned.

Table 21: Portscan Types

Type	Description
Portscan	A one-to-one port scan in which an attacker uses one or a few hosts to scan multiple ports on a single target host. One-to-one port scans are characterized by: • a low number of scanning hosts • a single host that is scanned • a high number of ports scanned A portscan detects TCP, UDP, and IP port scans.
Portsweep	A one-to-many port sweep in which an attacker uses one or a few hosts to scan a single port on multiple target hosts. Port sweeps are characterized by: • a low number of scanning hosts • a high number of scanned hosts • a low number of unique ports scanned A portsweep detects TCP, UDP, ICMP, and IP port sweeps.
Decoy Portscan	A one-to-one port scan in which the attacker mixes spoofed source IP addresses with the actual scanning IP address. Decoy port scans are characterized by: • a high number of scanning hosts • a low number of ports that are scanned only once • a single (or a low number of) scanned hosts The decoy port scan detects TCP, UDP, and IP protocol port scans.

Type	Description
Distributed Portscan	A many-to-one port scan in which multiple hosts query a single host for open ports. Distributed port scans are characterized by: • a high number of scanning hosts • a high number of ports that are scanned only once • a single (or a low number of) scanned hosts The distributed portscan detects TCP, UDP, and IP protocol port scans.

Port Scan Sensitivity Levels

The port_scan inspector provides three default scan sensitivity levels.

- default_low_port_scan
- default_med_port_scan
- default_high_port_scan

You can configure additional scan sensitivity levels with various filters:

- scans
- rejects
- nets
- ports

The port_scan inspector learns about a probe by gathering negative responses from the probed hosts. For example, when a web client uses TCP to connect to a web server, the client can assume that the web server listens on port 80. However, when an attacker probes a server, the attacker does not know in advance if the server offers web services. When the port_scan inspector detects a negative response (ICMP unreachable or TCP RST packet), it records the response as a potential portscan. The process is more difficult when the targeted host is on the other side of a device such as a firewall or router that filters negative responses. In this case, the port_scan inspector can generate filtered portscan events based on the sensitivity level that you select.

Best Practices for Configuring the Port Scan Inspector

To optimize the detection of port scans, we recommend that you tune the port_scan inspector to match your networks.

- Ensure that you carefully configure the watch_ip parameter. The watch_ip parameter helps the port_scan inspector filter legitimate hosts that are very active on your network. Some of the most common examples are NAT IPs, DNS cache servers, syslog servers, and nfs servers.
- Most of the false positives that the port_scan inspector may generate are of the filtered scan alert type. The alert type may indicate that a host was overly active during a specific time period. If the host

continually generates the filtered scan `alert` type, add the host to the `ignore_scanners` list or use a lower scan sensitivity level.

- Make use of the Priority Count, Connection Count, IP Count, Port Count, IP range, and Port range to determine false positives. The easiest way to determine false positives is through simple ratio estimations. The following is a list of ratios to estimate and the associated values that indicate a legitimate scan as opposed to a false positive.
  - Connection Count / IP Count - This ratio indicates an estimated average of connections per IP. For port scans, this ratio should be high. For port sweeps, this ratio should be low.
  - Port Count / IP Count - This ratio indicates an estimated average of ports connected to per IP. For port scans, this ratio should be high and indicates that the scanned host's ports were connected to by fewer IPs. For port sweeps, this ratio should be low, indicating that the scanning host connected to few ports but on many hosts.
  - Connection Count / Port Count - This ratio indicates an estimated average of connections per port. For port scans, this ratio should be low. This indicates that each connection was to a different port. For port sweeps, this ratio should be high. This indicates that there were many connections to the same port.

  The higher the priority count, the more likely it is a real port scan or port sweep (unless the host is managed by a firewall).

- If you are unable to detect port scans, you can lower the scan sensitivity level. You get the best protection with a higher scan sensitivity level. The low scan sensitivity level only generates alerts based on error responses and does not catch filtered scans. The low scan sensitivity level error responses can indicate a port scan, and the alerts generated by the low sensitivity level are highly accurate and require the least tuning. Filtered or high sensitivity level scans are prone to false positives.

Port Scan Inspector Parameters

memcap

Specifies the maximum tracker memory in bytes.

Type: integer

Valid range: `1024` to `9,007,199,254,740,992 (maxSZ)`

Default value: `10,485,760`

protos

Specifies the protocols to monitor. Provide a string of protocol abbreviations. To specify multiple protocols, separate each protocol abbreviation with a space.

Type: string

Valid values: `tcp`, `udp`, `icmp`, `ip`, `all`

Default value: `all`

scan_types

Specifies the types of port scan to examine. Provide a string of protocol abbreviations. To specify multiple protocols, separate each protocol string with a space.

Type: string

Valid values: `portscan`, `portsweep`, `decoy_portscan`, `distributed_portscan`, `all`

Default value: `all`

watch_ip

Specifies a list of CIDR blocks and IPs with optional ports to watch.

If `watch_ip` is not defined, the `port_scan` inspector examines all network traffic.

Type: string

Valid values: CIDR or IP address, list of CIDR or IP addresses

Default value: None

alert_all

Specifies whether to alert on all events over the threshold within the established window. If `alert_all` is set to `false`, the `port_scan` inspector only alerts on the first event over the threshold within the window.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

include_midstream

Specifies whether to list CIDRs with optional ports.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

tcp_decoy.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

tcp_decoy.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_decoy.scan

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

tcp_decoy.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_dist.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

tcp_dist.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_dist.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

tcp_dist.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_ports.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

tcp_ports.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_ports.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

tcp_ports.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_sweep.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

tcp_sweep.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_sweep.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

tcp_sweep.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_decoy.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

udp_decoy.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_decoy.scans

Specifies the of number scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

udp_decoy.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_dist.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

udp_dist.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_dist.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

udp_dist.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_ports.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

udp_ports.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_ports.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

udp_ports.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_sweep.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

udp_sweep.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

udp_sweep.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

udp_sweep.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_decoy.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

ip_decoy.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_decoy.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

ip_decoy.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_dist.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

ip_dist.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_dist.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

ip_dist.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_sweep.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

ip_sweep.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_sweep.scans

Specifies the of number scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

ip_sweep.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_proto.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

ip_proto.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

ip_proto.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

ip_proto.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

icmp_sweep.rejects

Specifies the number of scan attempts with negative responses.

Type: integer

Valid range: `0` to `65535`

Default value: `15`

icmp_sweep.ports

Specifies the number of times the port (or protocol) changed from a prior attempt.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

icmp_sweep.scans

Specifies the number of scan attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `100`

icmp_sweep.nets

Specifies the number of times the address changed from prior attempts.

Type: integer

Valid range: `0` to `65535`

Default value: `25`

tcp_window

Specifies the detection interval for transmission control protocol (TCP) scans.

Type: integer

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

Default value: `0`

udp_window

Specifies the detection interval for user datagram protocol (UDP) scans.

Type: integer

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

Default value: `0`

ip_window

Specifies the detection interval for internet protocol (IP) scans.

Type: integer

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

Default value: `0`

icmp_window

Specifies the detection interval for internet control message protocol (ICMP) scans.

Type: integer

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

Default value: `0`

Port Scan Inspector Rules

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

Table 22: Port Scan Inspector Rules

GID:SID	Rule Message
122:1	TCP portscan
122:2	TCP decoy portscan
122:3	TCP portsweep
122:4	TCP distributed portscan
122:5	TCP filtered portscan
122:6	TCP filtered decoy portscan
122:7	TCP filtered portsweep
122:8	TCP filtered distributed portscan
122:9	IP protocol scan
122:10	IP decoy protocol scan
122:11	IP protocol sweep
122:12	IP distributed protocol scan
122:13	IP filtered protocol scan
122:14	IP filtered decoy protocol scan
122:15	IP filtered protocol sweep
122:16	IP filtered distributed protocol scan
122:17	UDP portscan
122:18	UDP decoy portscan
122:19	UDP portsweep
122:20	UDP distributed portscan
122:21	UDP filtered portscan
122:22	UDP filtered decoy portscan
122:23	UDP filtered portsweep
122:24	UDP filtered distributed portscan

GID:SID	Rule Message
122:25	ICMP sweep
122:26	ICMP filtered sweep
122:27	open port

Port Scan Inspector Intrusion Rule Options

The `port_scan` inspector does not have any intrusion rule options.

CHAPTER 19

Rate Filter

Rate Filter Overview

Type	Module (basic)
Usage	Context
Instance Type	Singleton
Enabled	`false`

Rate-based attacks attempt to overwhelm a network or host by sending excessive traffic to a network or host, causing it to slow down or deny legitimate requests. You can use rate-based prevention to change the action of an intrusion rule in response to excessive matches on that rule.

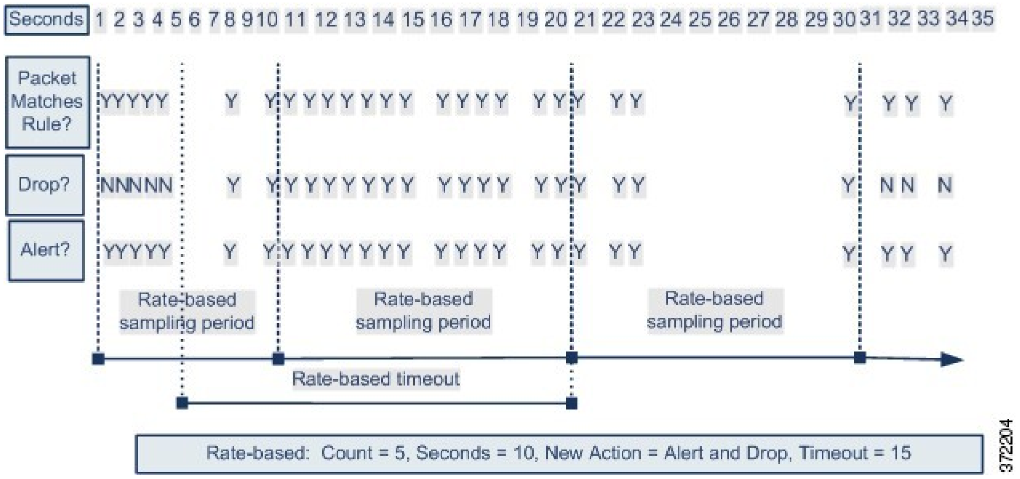
The `rate_filter` detects when too many matches for a rule occur within a given interval. You can use this feature on managed devices deployed inline to block rate-based attacks for a specified time, then revert to a rule state where rule matches only generate events and do not drop traffic.

You can configure the `rate_filter` to respond to any intrusion rule, but the rule you specify must be enabled for `rate_filter` to detect an attack and respond. For example, to establish a defense against a DDOS/SYN flood attack, enable rule 135:1 (TCP SYN received), and configure the `rate_filter` to alert on excessive triggers of rule 135:1.

Rate-based attack prevention identifies abnormal traffic patterns and attempts to minimize the impact of that traffic on legitimate requests. You can identify excessive rule matches in traffic going to a particular destination IP address or addresses or coming from a particular source IP address or addresses. You can also respond to excessive matches for a particular rule across all detected traffic.

The following diagram shows an example where an attacker is attempting to access a host. Repeated attempts to find a password trigger a rule which has rate-based attack prevention configured. The rate-based settings change the rule attribute to Drop and Generate Events after rule matches occur five times in a 10-second span. The new rule attribute times out after 15 seconds.

After the timeout, note that packets are still dropped in the rate-based sampling period that follows. If the sampled rate is above the threshold in the current or previous sampling period, the new action continues. The new action reverts to Generate Events only after a sampling period completes where the sampled rate was below the threshold rate.


Seconds 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
Packet Matches Rule?
YYYYY Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Drop?
NNNNN Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y N N N
Alert?
YYYYY Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Rate-based sampling period
Rate-based sampling period
Rate-based sampling period
Rate-based timeout
Rate-based: Count = 5, Seconds = 10, New Action = Alert and Drop, Timeout = 15
372204

You can define multiple rate-based filters on the same rule as well as on different rules. In an intrusion policy with multiple rate-based filters defined, the first filter listed in the policy has the highest priority. When two rate-based filter actions conflict, the action of the first rate-based filter is carried out.

The configuration parameters you set for the `rate_filter` apply to all traffic throughout your deployment. However, the system maintains a separate counter for the number of matches within the sampling period for each unique connection your system monitors. The system also applies changes to an action on a per-connection basis.

Note Rate-based actions cannot enable disabled rules or drop traffic that matches disabled rules.

Rate Filter Parameters

rate_filter[]

Specifies an array of `rate_filter` information. Each `rate_filter` includes a set of fields that can alter a rule action if the traffic contains a rate-based attack.

Type: array (object)

Example:

```
{
    "rate_filter": {
        "data": [
            {
                "apply_to": "[10.1.2.100, 10.1.2.101]",
                "count": 5,
                "gid": 135,
                "new_action": "alert",
                "seconds": 1,
                "sid": 1,
                "timeout": 5,
                "track": "by_src"
            }
        ],
        "enabled": true,
        "type": "singleton"
    }
}
```

rate_filter[].gid

Specifies a generator ID (GID) which identifies the rule to match.

Type: integer

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

Default value: `1`

rate_filter[].sid

Specifies a signature ID (SID) which identifies the rule to match.

Type: integer

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

Default value: `1`

rate_filter[].track

Specifies a filter to match source or destination addresses.

Type: enum

Valid values:

- `by_src`: Filter only traffic that matches the rule specified by `rate_filter[].gid` and `rate_filter[].sid`, and where source address matches `rate_filter[].apply_to`.
- `by_dst`: Filter only traffic that matches the rule specified by `gid` and `sid`, and where destination address matches `rate_filter[].apply_to`.
- `by_rule`: Filter all traffic that matches the rule specified by `rate_filter[].gid` and `rate_filter[].sid`.

Default value: `by_src`

rate_filter[].count

Specifies the number of rule matches to allow in the sampling period (`rate_filter[].seconds`) before applying the alternative action (`rate_filter[].new_action`).

Type: integer

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

Default value: `1`

rate_filter[].seconds

Specifies the number of seconds in the sampling period to match traffic. `rate_filter[].seconds` represents the amount of time to elapse before resetting the internal counter of matches to zero.

Type: integer

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

Default value: `1`

rate_filter[].new_action

Specifies the action to take in response to matches in traffic that exceed the limitation specified by `rate_filter[].seconds` and `rate_filter[].count`.

Type: string

Valid values: One of the following strings: `alert`, `block`, `drop`, `log`, `pass`, `react`, `reject`, `rewrite`.

Default value: `alert`

rate_filter[].timeout

Specifies the number of seconds to perform the action specified by `rate_filter[].new_action` in response to matching traffic.

Type: integer

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

Default value: `0`

rate_filter[].apply_to

Specifies the list of network addresses to match against traffic source or destination address depending on the value of `rate_filter[].track`.

Type: string

Valid values: A valid IPv4 address, or an IPv4 address block in CIDR format.

Default value: None

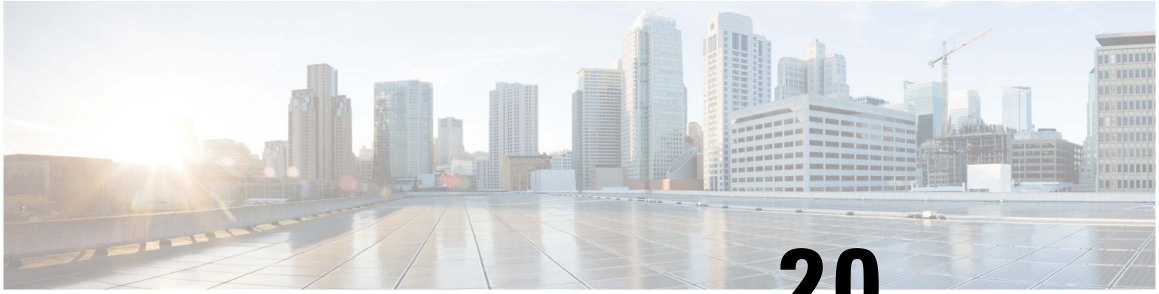
Rate Filter Rules

The `rate_filter` does not have any associated rules.

You can confgure the `rate_filter` to respond to any intrusion rules. Enable the `rate_filter` for a rule to detect an attack and respond.

Rate Filter Intrusion Rule Options

The `rate_filter` does not have any intrusion rule options.



CHAPTER 20

S7CommPlus Inspector

S7CommPlus Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

S7CommPlus is a proprietary protocol developed by Siemens. S7CommPlus enables communication between programmable logic controllers of the Siemens S7 family of products.

The `s7commplus` inspector detects and analyzes S7CommPlus traffic. You can set intrusion rule options to alert on the specified S7CommPlus function and operation code header fields, and detect attacks in S7CommPlus traffic.

Best Practices for Configuring the S7CommPlus Inspector

If your network does not contain an enabled S7CommPlus device, you should not enable the `s7commplus` inspector in a network analysis policy that you apply to traffic.

S7CommPlus Inspector Parameters

S7CommPlus TCP port configuration

The binder inspector defines the S7CommPlus TCP port configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "role": "server",
            "proto": "tcp",
            "ports": "102"
        },
        "use": {
            "type": "s7commplus"
        }
    },
    {
        "when": {
            "role": "any",
            "service": "s7commplus"
        },
        "use": {
            "type": "s7commplus"
        }
    }
]
```

Note The s7commplus inspector does not provide any parameters.

S7CommPlus Inspector Rules

Enable the s7commplus inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 23: S7CommPlus Inspector Rules

GID:SID	Rule Message
149:1	length in S7commplus MBAP header does not match the length needed for the given S7commplus function
149:2	S7commplus protocol ID is non-zero
149:3	reserved S7commplus function code in use

S7CommPlus Inspector Intrusion Rule Options

You can use the `s7commplus` keywords alone or in combination to create custom intrusion rules that identify attacks against traffic detected by the `s7commplus` inspector. For configurable keywords, specify a single known value or a single integer within the allowed range.

Note the following:

- Multiple `s7commplus` keywords in the same rule are AND-ed.
- Using multiple `s7commplus_func` or `s7commplus_opcode` keywords in the same rule negates the rule. The negated rule cannot match traffic. To search for multiple values with these keywords, create multiple rules.

s7commplus_content

Use the `s7commplus_content` keyword to position the detection cursor to the beginning of the S7CommPlus packet payload. We recommend that you set this keyword before you use a `content` or `protected_content` keyword in an S7CommPlus intrusion rule.

Syntax: `s7commplus_content;`

Examples: `s7commplus_content;`

s7commplus_func

Use the `s7commplus_func` keyword to match against one of the specified S7CommPlus header parameters. You can specify the S7CommPlus parameter name or the corresponding hexadecimal code.

Type: string

Syntax: `s7commplus_func: <header_parameter>;`

Valid values:

Name	Code
explore	0x04BB
createobject	0x04CA
deleteobject	0x04D4
setvariable	0x04F2
getlink	0x0524
setmultivar	0x0542
getmultivar	0x054C
beginsequence	0x0556
endsequence	0x0560
invoke	0x056B

Name	Code
getvarsubstr	0x0586
0x0 through 0xFF	Note that numeric expressions allow for additional values.

Examples: `s7commplus_func: createobject;`

s7commplus_opcode

Use the `s7commplus_opcode` keyword to match against one of the specified S7CommPlus header parameters. You can specify the S7CommPlus parameter name or the corresponding hexadecimal code.

Type: string

Syntax: `s7commplus_opcode: <header_parameter>`

Valid values:

Name	Code
request	0x31
response	0x32
notification	0x33
response2	0x02
0x0 through 0xFF	Note that numeric expressions allow for additional values.

Examples: `s7commplus_opcode: 0x31;`

CHAPTER 21

SIP Inspector

SIP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_udp`
Enabled	`true`

The Session Initiation Protocol (SIP) manages the creation, modification, and teardown of real-time call sessions that include one or more participants. The applications that SIP can control include: internet telephony, multimedia conferencing, instant messaging, online gaming, and file transfer. The SIP protocol is a text-based, request and response protocol.

A SIP request includes a `method` field that identifies the purpose of the request, and a `Request-URI` which specifies where to send the request. A status code in each SIP response indicates the outcome of the requested action. The SIP protocol uses TCP (port 5060) or UDP (port 5061).

After SIP creates a call session, SIP can transmit audio and video streams over the real-time transport protocol (RTP). The SIP message body embeds the data-channel parameter negotiation, session announcement, and session invitation in the Session Description Protocol (SDP) format.

The `sip` inspector detects and analyzes SIP messages in network traffic. The `sip` inspector extracts the SIP header and message body and passes any data in the SIP message body to the detection engine.

The `sip` inspector detects anomalies and known vulnerabilities in SIP traffic, including disordered and invalid call sequences.

Note

- The sip inspector does not decode RTP messages. The sip inspector identifies the RTP channel based on the port defined in the SDP data.
- UDP typically carries media sessions supported by SIP. The sip inspector obtains session tracking information from the decoded UDP stream.
- SIP rule options allow you to position the detection cursor to the SIP packet header or message body and to limit detection to packets for specific SIP methods or status codes.

SIP Inspector Parameters

SIP service configuration

The binder inspector defines the SIP service configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "role": "any",
            "service": "sip"
        },
        "use": {
            "type": "sip"
        }
    }
]
```

ignore_call_channel

Specifies whether to inspect audio/video data channel traffic. When enabled, the sip inspector decodes all non-data SIP channel traffic and ignores audio/video SIP data channel traffic.

Type: boolean

Valid values: true, false

Default value: false

max_call_id_len

Specifies the maximum number of bytes to allow in the Call-ID header field. The Call-ID field uniquely identifies the SIP session in requests and responses. The sip inspector does not generate an alert when the max_call_id_len is 0.

You can enable rule 140:5 to generate events, and in an inline deployment, drop offending packets. The sip inspector generates an event when the Call-ID header length is greater than the value of max_call_id_len.

Type: integer

Valid range: 0 to 65535

Default value: `256`

max_contact_len

Specifies the maximum number of bytes to allow in the `Contact` header field. The `Contact` field provides a URI that specifies the location to contact with subsequent messages. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:15 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the `Contact` header field length is greater than the value of `max_contact_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `256`

max_content_len

Specifies the maximum number of bytes to allow in the content of the message body. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:16 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the content length is greater than the value of `max_content_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `1024`

max_dialogs

Specifies the maximum number of dialogs allowed within a stream session. If the number of dialogs is more than the set limit, the `sip` inspector drops the oldest dialogs until the number of dialogs does not exceed the maximum number specified.

You can enable rule 140:27 to generate events, and in an inline deployment, drop offending packets.

Type: integer

Valid range: `1` to `4,294,967,295 (max32)`

Default value: `4`

max_from_len

Specifies the maximum number of bytes to allow in the `From` header field. The `From` field identifies the sender of the message. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:9 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the `From` field length is greater than the value of `max_from_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `256`

max_request_name_len

Specifies the maximum number of bytes to allow in the request name. The SIP request name refers to the name of the method specified in the SIP `CSeq` transaction identifier. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:7 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the request name length is greater than the value of `max_request_name_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `20`

max_requestName_len

The `max_requestName_len` parameter is deprecated. Use the `max_request_name_len` parameter instead.

max_to_len

Specifies the maximum number of bytes to allow in the `To` header field. The `To` field identifies the recipient of the message. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:11 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the `To` field length is greater than the value of `max_to_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `256`

max_uri_len

Specifies the maximum number of bytes to allow in the SIP `Request-URI`. The `Request-URI` indicates the destination path to the requested resource. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:3 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the `Request-URI` field length is greater than the value of `max_uri_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `256`

max_via_len

Specifies the maximum number of bytes to allow in the `Via` header field. The `Via` field identifies the transport to use in the request and the location of the recipient. The `sip` inspector does not generate an alert when the value is 0.

You can enable rule 140:13 to generate events, and in an inline deployment, drop offending packets. The `sip` inspector generates an event when the `Via` field length is greater than the value of `max_via_len`.

Type: integer

Valid range: `0` to `65535`

Default value: `1024`

methods

Specifies a list of SIP methods to detect. Method names are case-insensitive. Use a comma or space to separate method names in the list. A method name can only include alphabetic characters, numbers, and the underscore character.

Type: string

Valid values: `ack`, `benotify`, `bye`, `cancel`, `do`, `info`, `invite`, `join`, `message`, `notify`, `options`, `prack`, `publish`, `quath`, `refer`, `register`, `service`, `sprack`, `subscribe`, `unsubscribe`, `update`

Default value: `invite cancel ack bye register options`

SIP Inspector Rules

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

Table 24: SIP Inspector Rules

GID:SID	Rule Message
140:2	empty request URI
140:3	URI is too long
140:4	empty call-Id
140:5	Call-Id is too long
140:6	CSeq number is too large or negative
140:7	request name in CSeq is too long
140:8	empty From header
140:9	From header is too long
140:10	empty To header
140:11	To header is too long
140:12	empty Via header
140:13	Via header is too long
140:14	empty Contact
140:15	contact is too long
140:16	content length is too large or negative
140:17	multiple SIP messages in a packet
140:18	content length mismatch
140:19	request name is invalid

GID:SID	Rule Message
140:20	Invite replay attack
140:21	illegal session information modification
140:22	response status code is not a 3 digit number
140:23	empty Content-type header
140:24	SIP version is invalid
140:25	mismatch in METHOD of request and the CSEQ header
140:26	method is unknown
140:27	maximum dialogs within a session reached

SIP Inspector Intrusion Rule Options

sip_method

A SIP request method identifies the purpose of the request. Use the `sip_method` keyword to match the method in a SIP request. Method names are case-insensitive. Separate multiple method names with a comma.

Type: string

Syntax: `sip_method: <methods>;`

Valid values: `ack`, `benotify`, `bye`, `cancel`, `do`, `info`, `invite`, `join`, `message`, `notify`, `options`, `prack`, `publish`, `quath`, `refer`, `register`, `service`, `sprack`, `subscribe`, `unsubscribe`, `update`

Examples: `sip_method: "ack,service,info,bye";`

sip_stat_code

A SIP response includes a three-digit status code. The SIP status code indicates the outcome of the requested action. Use the `sip_stat_code` keyword to match a SIP response with the specified status codes.

You can specify a one-digit number that represents the first digit of a three-digit status code, a three-digit number, or a comma-separated list of numbers using either number combination. A list matches if any single number in the list matches the code in the SIP response.

Type: integer

Syntax: `sip_stat_code: <codes>;`

Valid ranges:

- `1` to `9`
- `100` to `999`

Examples: `sip_stat_code: "1";`

Table 25: SIP Parameter Values and Status Codes

Parameter Value	Detected Status Codes	Description
189	189	Set a specific status code.
1	100 - 199	Set a single digit.
222, 3	222; 300 - 399	Set a comma-separated list of three-digit or single digit numbers.

sip_header

Use the `sip_header` keyword to position the detection cursor to the beginning of the extracted SIP header buffer. Restricts inspection to the header fields.

Syntax: `sip_header;`

Examples: `sip_header;`

sip_body

Use the `sip_body` keyword to position the detection cursor to the beginning of the extracted SIP message body. Restricts inspection to the message body.

Syntax: `sip_body;`

Examples: `sip_body;`

Note The `sip` inspector extracts the entire message body and makes it available to the rules engine. The rules engine is not limited to searching for session description protocol (SDP) content.

CHAPTER 22

SMTP Inspector

SMTP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`true`

Simple Mail Transfer Protcocol (SMTP) enables a mail client to send messages to a mail server. SMTP issues commands to deliver a message to a recipient. An SMTP server uses TCP port 25 for insecure sessions or TCP port 587 for SMTP over SSL/TLS.

The `smtp` inspector detects SMTP traffic and analyzes SMTP commands and responses.

The `smtp` inspector identifies the command, header, and body sections of SMTP messages, and extracts and decodes multi-purpose internet mail extensions (MIME) attachments. MIME attachments may include multiple attachments and large attachments that span multiple packets.

The `smtp` inspector identifies and adds SMTP messages to the Snort allow list. When enabled, intrusion rules generate events on anomalous SMTP traffic.

You can configure the `smtp` inspector to:

- Log sender email ID, recipient email ID, email headers, and attachment filenames along with all generated events for the session.
- Normalize SMTP command lines by removing extraneous space characters. The `smtp` inspector normalizes the space (ASCII 0x20) or tab (ASCII 0x09) characters.

- Ignore TLS-encrypted traffic to improve performance.
- Ignore plain-text mail data to improve performance.

Best Practices for Configuring the SMTP Inspector

We recommend that you follow the guidelines from RFC 2821 to configure the `smtp` inspector's core configuration parameters:

- `max_command_line_len`: 512 characters
- `max_header_line_len`: 1024 characters
- `max_response_line_len`: 512 characters

SMTP Inspector Parameters

SMTP service configuration

The `binder` inspector defines the SMTP `service` configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "service": "smtp",
            "role": any
        },
        "use": {
            "type": "smtp"
        }
    }
]
```

alt_max_command_line_len[]

Specifies an array of SMTP command and an alternate maximum line length for the command. The alternate maximum line length overrides the value of the `max_command_line_len` for the SMTP command. You can enable rule 124:4 to generate events for this parameter.

Type: array

Example:

```
{
    "alt_max_command_line_len": [
        {
            "command": "AUTH",
            "length": 240
        }
    ]
}
```

alt_max_command_line_len[].command

Specifies a command string.

Type: string

Valid values: SMTP command

Default value: See Table 26: SMTP Commands and Default Alternate Command Lengths.

alt_max_command_line_len[].length

Specifies an alternate maximum command line length. Specify `0` to disable the detection of the command line length for a command.

Type: integer

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

Default value: See Table 26: SMTP Commands and Default Alternate Command Lengths.

Table 26: SMTP Commands and Default Alternate Command Lengths

Command	Length
ATRN	255
AUTH	246
BDAT	255
DATA	246
DEBUG	255
EHLO	500
EMAL	255
ESAM	255
ESND	255
ESOM	255
ETRN	500
EVFY	255
EXPN	255
HELO	500
HELP	500
IDENT	255
MAIL	260

Command	Length
NOOP	255
ONEX	246
QUEU	246
QUIT	246
RCPT	300
RSET	255
SAML	246
SEND	246
SIZE	255
SOML	246
STARTTLS	246
TICK	246
TIME	246
TURN	246
TURNME	246
VERB	246
VRFY	255
XADR	246
XAUTH	246
XCIR	246
XEXCH50	246
X-EXPS	246
XGEN	246
XLICENSE	246
X-LINK2STATE	246
XQUE	246
XSTA	246
XTRN	246

Command	Length
XUSR	246

auth_cmds

Specifies a list of SMTP commands that initiate the authentication exchange. Separate multiple SMTP commands with a space.

Type: string

Valid values: SMTP authentication exchange initiation commands

Default value: `AUTH XAUTH X-EXPS`

b64_decode_depth

Specifies the maximum number of bytes to extract and decode from each Base64 encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 124:10 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails.

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

binary_data_cmds

Specifies a list of SMTP commands that initiate sending data and use a length value (in octets) after the command to indicate the amount of data to be sent. Separate multiple SMTP commands with a space.

Type: string

Valid values: Valid SMTP data send initiation commands that use a data length argument

Default value: `BDATA XEXCH50`

bitenc_decode_depth

Specifies the maximum number of bytes to extract from each non-encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable the extraction of the non-encoded MIME attachment. Specify -1 to place no limit on the number of bytes to extract. These attachment types include 7-bit, 8-bit, binary, and various multipart content types such as plain text, JPEG and PNG images, and MP4 files.

Type: integer

Valid range: `-1` to `65535`

Default value: `-1`

data_cmds

Specifies a list of SMTP commands that initiate sending data and use an end of data delimiter (<CRLF>.<CRLF>).

Type: string

Valid values: SMTP data send initiation command that uses an end of data delimiter.

Default value: `DATA`

decompress_pdf

Specifies whether to decompress `application/pdf` (PDF) files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_swf

Specifies whether to decompress `application/vnd.adobe.flash-movie` (SWF) files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_vba

Specifies whether to decompress Microsoft Office Visual Basic for Applications macro files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

decompress_zip

Specifies whether to decompress `application/zip` (ZIP) files in MIME attachments.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

email_hdrs_log_depth

Specifies the number of bytes of the email header to extract from the SMTP data. Specify 0 to disable extraction of the email header.

Type: integer

Valid range: `0` to `20480`

Default value: `1464`

ignore_data

Specifies whether to decode the email data section (except for MIME mail headers).

Type: boolean

Valid values: `true`, `false`

Default value: `false`

ignore_tls_data

Specifies whether to decode TLS-encrypted data.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

log_email_hdrs

Specifies whether to decode and log the SMTP email header and all generated events for the session.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

log_filename

Specifies whether to decode and log the MIME attachment filenames extracted from the `Content-Disposition` header within the MIME body, and all generated events for the session. If the message contains multiple MIME attachments, the SMTP inspector logs the filenames separated by a comma. The SMTP inspector logs no more than 1024 bytes.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

log_mailfrom

Specifies whether to decode and log the sender's email address extracted from the SMTP `MAIL FROM` command, and all generated events for the session. If the message contains multiple senders, the SMTP inspector logs the senders separated by a comma. The SMTP inspector logs no more than 1024 bytes.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

log_rcptto

Specifies whether to decode and log the recipient email addresses from the SMTP `RCPT TO` command, and all generated events for the session. If the message contains multiple recipients, the SMTP inspector logs the recipients separated by a comma. The SMTP inspector logs no more than 1024 bytes.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

max_auth_command_line_len

Specifies the maximum number of bytes accepted for the SMTP authentication command line.

You can enable rule 124:15 to generate events, and in an inline deployment, drop offending packets. Specify 0 to disable alerts on SMTP `AUTH` commands, or omit `max_auth_command_line_len` parameter from your Snort configuration.

Type: integer

Valid range: `0` to `65535`

Default value: `1000`

max_command_line_len

Specifies the maximum number of bytes accepted for the SMTP command line.

RFC 2821, the Network Working Group specification on SMTP, recommends a maximum command line length of 512 bytes. Specify 0 to disable alerts on SMTP command line length, or omit the `max_command_line_len` parameter from your Snort configuration.

You can enable rule 124:1 to generate events, and in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `65535`

Default value: `512`

max_header_line_len

Specifies the maximum number of bytes accepted for the SMTP data header line.

RFC 2821, the Network Working Group specification on SMTP, recommends a maximum data header line length of 1024 bytes. Specify 0 to disable alerts on SMTP data header line length, or omit the `max_header_line_len` parameter from your Snort configuration.

You can enable rules 124:2 and 124:7 to generate events, and in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `65535`

Default value: `1000`

max_response_line_len

Specifies the maximum number of bytes accepted for the SMTP response line.

RFC 2821, the Network Working Group specification on SMTP, recommends a maximum response line length of 512 bytes. Specify 0 to disable alerts on SMTP response line length, or omit the `max_response_line_len` parameter from your Snort configuration.

You can enable rules 124:3 to generate events, and in an inline deployment, drop offending packets.

Type: integer

Valid range: `0` to `65535`

Default value: 512

normalize

Specifies whether to normalize all commands, no commands, or a list of commands. You can specify the list of commands in the normalize_cmds parameter. The inspector checks for more than one space (ASCII 0x20) or tab (ASCII 0x09) character after a command.

Type: enum

Valid values:

- none
- cmds
- all

Default value: none

normalize_cmds

Specifies a list of SMTP commands to normalize. Separate multiple SMTP commands with a space.

Type: string

Valid values: SMTP commands

Default value: None

qp_decode_depth

Specifies the maximum number of bytes to extract and decode from each quoted-printable (QP) encoded MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 124:11 to generate events, and in an inline deployment, drop offending packets.

Type: integer

Valid range: -1 to 65535

Default value: -1

uu_decode_depth

Specifies the maximum number of bytes to extract and decode from each Unix-to-Unix encoded (uuencoded) MIME email attachment. You can specify an integer less than 65535, or specify 0 to disable decoding. Specify -1 to place no limit on the number of bytes to decode.

You can enable rule 124:13 to generate events for this parameter, and in an inline deployment, drop offending packets when decoding fails (due to incorrect encoding or corrupted data, for instance).

Type: integer

Valid range: -1 to 65535

Default value: -1

valid_cmds

Specifies an additional list of SMTP commands which the SMTP inspector considers valid.

The SMTP inspector defines a list of default, valid SMTP commands: `ATRN AUTH BDAT DATA DEBUG EHLO EMAL ESAM ESND ESOM ETRN EVFY EXPN HELO HELP IDENT MAIL NOOP ONEX QUEU QUIT RCPT RSET SAML SEND SIZE STARTTLS SOML TICK TIME TURN TURNME VERB VRFY X-EXPS X-LINK2STATE XADR XAUTH XCIR XEXCH50 XGEN XLICENSE XQUE XSTA XTRN XUSR`.

You can enable rule 124:5 to generate events, and in an inline deployment, drop offending packets.

Type: string

Valid values: SMTP commands

Default value: None

xlink2state

Specifies how the SMTP inspector handles packets that are part of X-Link2State Microsoft Exchange buffer data overflow attacks (See CVE-2005-0560 for a description of the vulnerability). You can disable detection (`disable`), enable detection and generate alerts (`alert`), or enable detection and drop the offending packets (`drop`).

You can enable rule 124:8 to generate events for this parameter, and in an inline deployment, drop offending packets.

Type: enum

Valid values:

- `disable`
- `alert`
- `drop`

Default value: `alert`

SMTP Inspector Rules

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

Table 27: SMTP Inspector Rules

GID:SID	Rule Message
124:1	attempted command buffer overflow
124:2	attempted data header buffer overflow
124:3	attempted response buffer overflow
124:4	attempted specific command buffer overflow
124:5	unknown command

GID:SID	Rule Message
124:6	illegal command
124:7	attempted header name buffer overflow
124:8	attempted X-Link2State command buffer overflow
124:10	base64 decoding failed
124:11	quoted-printable decoding failed
124:13	Unix-to-Unix decoding failed
124:14	Cyrus SASL authentication attack
124:15	attempted authentication command buffer overflow
124:16	file decompression failed

SMTP Inspector Intrusion Rule Options

vba_data

Sets the detection cursor to the Microsoft Office Visual Basic for Applications macros buffer.

Syntax: `vba_data;`

Examples: `vba_data;`

CHAPTER 23

SnortML

Type	Inspector (passive)
Usage	Inspect
Instance Type	Singleton
Other Inspectors Required	`snort_ml_engine, http_inspect`
Enabled	Max Detect

Every day new vulnerabilities are discovered in software critical to the function of the modern world. Security analysts take apart these new vulnerabilities, isolate what is necessary to trigger them, and write signatures to detect exploits targeting them. Most signatures can only really be written for specific vulnerabilities.

SnortML is a neural network-based exploit detection for the Snort intrusion prevention system. It is designed to not only learn to detect known attacks from training data, but also learn to detect attacks it has never seen before.

The `snort_ml` inspector searches primarily for SQL injection attacks over HTTP. As this inspector may affect performance, it is only enabled by default when in `Max Detect` mode.

- SnortML Rules, on page 183
- SnortML Parameters, on page 184

SnortML Rules

Enable the `snort_ml` inspector rule to generate events and, in an inline deployment, drop offending packets. The `snort_ml` inspector rule is only enabled by default under the Maximum Detection NAP policy.

Table 28: Snort ML Inspector Rules

GID:SID	Rule Message
411:1	(snort_ml) potential threat found in HTTP parameters via Neural Network Based Exploit Detection.

SnortML Parameters

uri_depth

Specifies the number of bytes to scan from the HTTP URI. The value `-1` means unlimited.

Type: integer

Valid range: `-1` to `2147483648`

Default value: `-1`

client_body_depth

Specifies the number of bytes to scan from the HTTP client body. The value `-1` means unlimited.

Type: integer

Valid range: `-1` to `2147483648`

Default value: `0`

CHAPTER 24

SSH Inspector

SSH Inspector Overview

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

Secure Shell Protocol (SSH) is network protocol that enables secure communication between a client and server over an unsecured network. SSH supports tunneling and authenticates a remote host using public-key cryptography.

You can use SSH to securely transfer files, or login into a remote host and interact with the command line. The SSH protocol uses port 22 over TCP, UDP, or SCTP.

The `ssh` inspector decodes stream packets and detects the following SSH exploits:

- Challenge-Response Buffer Overflow exploit
- CRC-32 exploit
- SecureCRT SSH Client Buffer Overflow exploit
- Incorrect SSH message direction

Challenge-Response Buffer Overflow and CRC-32 attacks occur after authentication when the network connection between hosts is encrypted. Both types of attack send a large payload of more than 20 KB to the server immediately after the authentication challenge.

The ssh inspector detects the Challenge-Response Buffer Overflow and CRC-32 attacks by counting the number of bytes transmitted to the server. If the bytes exceed the defined limit within a predefined number of packets, the ssh inspector generates an alert. CRC-32 attacks apply only to SSH Version 1 and Challenge-Response Buffer Overflow exploits apply only to SSH Version 2. The ssh inspector reads the SSH version string at the beginning of the session to identify the type of attack.

The SecureCRT SSH Client Buffer Overflow and protocol mismatch attacks occur before the key exchange when hosts are attempting to secure a connection. The SecureCRT SSH Client Buffer Overflow attack sends an overly long protocol identifier string to the client, causing a buffer overflow. A protocol mismatch attack occurs when either a non-SSH client application attempts to connect to a secure SSH server, or the server and client version numbers do not match.

Note The ssh inspector does not handle brute force attacks.

Best Practices for Configuring the SSH Inspector

We recommend that you use the default ssh inspector configuration settings. If you exceed the maximum number of encrypted packets for a session, defined in the max_encrypted_packets parameter, the ssh inspector stops processing traffic for that session to improve performance. The ssh inspector only detects SSH vulnerabilities that appear at the beginning of the SSH session.

Note If the ssh inspector generates a false positive on Challenge-Response Overflow or CRC 32, you can increase the number of required client bytes with the max_client_bytes parameter.

SSH Inspector Parameters

SSH service configuration

The binder inspector defines the SSH service configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "service": "ssh",
            "role": any
        },
        "use": {
            "type": "ssh"
        }
    }
]
```

max_encrypted_packets

Specifies the maximum number of encrypted packets to examine before the `ssh` inspector ignores an SSH session. If you exceed the maximum number of encrypted packets for a session, the `ssh` inspector stops processing traffic for that session to improve performance.

Type: integer

Valid range: `-1` to `65535`

Default value: `25`

max_client_bytes

Specifies the maximum number of unanswered bytes to transmit to the server before the `ssh` inspector alerts on Challenge-Response Overflow or CRC 32. If you exceed the `max_client_bytes` limit before `max_encrypted_packets` are sent, the inspector assumes an attack has occurred and ignores the traffic.

You can enable rule 128:1 to generate an alert when the inspector detects a Challenge-Response Overflow or rule 128:2 to generate an alert when the inspector detects a CRC 32 exploit.

For each valid response the client receives from the server, the `ssh` inspector resets the packet count for `max_client bytes`.

Note We do not recommend that you set `max_client_bytes` to `0` or `1`. If you set the `max_client_bytes` to `0` or `1`, the `ssh` inspector always alerts.

Type: integer

Valid range: `0` to `65535`

Default value: `19600`

max_server_version_len

Specifies the maximum length of the SSH server version string. If the length of the SSH server version string exceeds the `max_server_version_len`, the `ssh` inspector generates an alert. You can enable rule 128:3 to alert on the Secure CRT server version string overflow.

Type: integer

Valid range: `0` to `255`

Default value: `80`

Note The `ssh` inspector default configuration does not enable any alerts.

SSH Inspector Rules

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

Table 29: SSH Inspector Rules

GID:SID	Rule Message
128:1	challenge-response overflow exploit
128:2	SSH1 CRC32 exploit
128:3	server version string overflow
128:5	bad message direction
128:6	payload size incorrect for the given payload
128:7	failed to detect SSH version string

SSH Inspector Intrusion Rule Options

The `ssh` inspector does not have any intrusion rule options.

CHAPTER 25

Stream ICMP Inspector

Stream ICMP Inspector Overview

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

Internet Control Message Protocol (ICMP) is a network-layer protocol used by network utility applications and network devices. ICMP sends diagnostic and error information to identify communication success or failure between IP hosts. An ICMP message includes header and data sections.

ICMP conveys information about other flows. It does not carry data that needs reassembly, nor does it require target-based binding.

The `stream_icmp` inspector defines ICMP flow tracking. For pings, the inspector provides basic flow tracking through the source and destination IP address fields and the port fields in the ICMP header. For unreachable destinations, the inspector analyzes the original IP addresses and transport ports, then it updates the session's state. The `port_scan` inspector can use the unreachable host and port, if available.

Best Practices for Configuring the Stream ICMP Inspector

Consider the following best practices when you configure the `stream_icmp` inspector:

- Create a stream_icmp inspector for each session timeout that you want to apply to a host or network. The stream_icmp inspector associates the session_timeout with the ICMP hosts or networks defined in the binder inspector.

  You can have multiple versions of the stream_icmp inspector in the same network analysis policy (NAP).

Stream ICMP Inspector Parameters

session_timeout

Specifies the number of seconds that the stream_icmp inspector keeps an inactive ICMP stream in the state table. The next time Snort detects an ICMP datagram with the same flow key, it checks if the session timeout on the earlier flow has expired. If the timeout has expired, Snort closes the flow and starts a new flow. Snort checks for stale flows associated with the base stream configuration.

Type: integer

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

Default value: 60

Stream ICMP Inspector Rules

The stream_icmp inspector does not have any associated rules.

Stream ICMP Inspector Intrusion Rule Options

The stream_icmp inspector does not have any intrusion rule options.

CHAPTER 26

Stream IP Inspector

Stream IP Inspector Overview

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

Internet Protocol (IP) is a connectionless, network-layer protocol that forms the basis of the internet. IP uses host addresses to route messages from a source host to a destination host across IP networks. IP can route both TCP and UDP data packets in addition to other transport protocols.

An IP message contains header and data sections. The IP header includes IP addresses used to route a messsage to its destination. The IP data section encapsulates the message payload. IP handles reassembly and fragmentation of messages.

The `stream_ip` inspector detects an IP network flow and examines the packets in the flow. The `stream_ip` inspector defines IP session and flow tracking, operating system policy, and datagram overlaps configuration parameters. Depending on the mode, either the `stream_ip` inspector or the Snort data plane handles defragmentation.

Best Practices for Configuring the Stream IP Inspector

Consider the following best practices when you configure the `stream_ip` inspector:

- Create a stream_ip inspector for each IP configuration that you want to apply to a host, endpoint, or network. The stream IP inspector associates the IP configuration with the IP hosts, endpoints, or networks defined in the binder inspector.

  You can have multiple versions of the stream_ip inspector in the same network analysis policy.

Stream IP Inspector Parameters

max_overlaps

Specifies the maximum allowed overlaps for each datagram. Specify 0 to permit an unlimited number of overlaps.

You can enable rule 123:12 to trigger an alert for excessive fragment overlaps.

Type: integer

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

Default value: 0

min_frag_length

Specifies the minimun number of bytes expected in the IP fragment. Specify 0 to permit an unlimited number of bytes in the IP fragment.

You can enable rule 123:13 to trigger an alert for fragments shorter than min_frag_length.

Type: integer

Valid range: 0 to 65535

Default value: 0

min_ttl

Specifies a minimum number of hops or time to live (TTL). Discard fragments below the specified minimum TTL.

You can enable rule 123:11 to trigger an alert for fragments with a TTL below this value.

Type: integer

Valid range: 1 to 255

Default value: 1

policy

Specifies the operating system of the target host or hosts. The operating system determines the appropriate IP fragment reassembly policy and operating system characteristics. You can define only one policy parameter for each stream IP 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 IP stream inspector to specify the correct operating system.

Type: enum

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

Table 30: Valid Values for Policy

Policy	Operating Systems
first	Unknown OS
linux	Linux
bsd	AIX FreeBSD OpenBSD
bsd_right	HP JetDirect (printer)
last	Cisco IOS
windows	Windows 98 Windows NT Windows 2000 Windows XP
solaris	Solaris OS SunOS

Default value: linux

session_timeout

Specifies the number of seconds that the stream_ip inspector keeps an inactive IP stream in the state table. The next time Snort detects an IP datagram with the same flow key, it checks if the session timeout on the earlier flow has expired. If the timeout has expired, Snort closes the flow and starts a new flow. Snort checks for stale flows associated with the base stream configuration.

Type: integer

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

Default value: 60

Stream IP Inspector Rules

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

Table 31: Stream IP Inspector Rules

GID:SID	Rule Message
123:1	inconsistent IP options on fragmented packets
123:2	teardrop attack
123:3	short fragment, possible DOS attempt
123:4	fragment packet ends after defragmented packet
123:5	zero-byte fragment packet
123:6	bad fragment size, packet size is negative
123:7	bad fragment size, packet size is greater than 65536
123:8	fragmentation overlap
123:11	TTL value less than configured minimum, not using for reassembly
123:12	excessive fragment overlap
123:13	tiny fragment

Stream IP Inspector Intrusion Rule Options

The `stream_ip` inspector does not have any intrusion rule options.

CHAPTER 27

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, on page 13.

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 32: 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 33: 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 34: Range Formats.

Examples: `stream_size: >6;`

Table 34: 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

CHAPTER 28

Stream UDP Inspector

Stream UDP Inspector Overview

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

User Datagram Protocol (UDP) is a connectionless, low-latency transport layer protocol. UDP enables stateless communication between two network endpoints before an agreement is provided by the receiving party. To evaluate the integrity of the message header and data, UDP uses checksums.

The `stream_udp` inspector checks the source and destination IP address fields in the IP datagram header, and the port fields in the UDP header to determine the direction of flow and identify a session. A session ends when a configurable timer is exceeded, or when either endpoint receives an ICMP message that the other endpoint is unreachable.

The UDP stream inspector does not generate events. You can enable packet decoder rules (GID 116) to detect UDP header anomalies.

Best Practices for Configuring the Stream UDP Inspector

Consider the following best practices when you configure the `stream_udp` inspector:

- Create a `stream_udp` inspector for each session timeout that you want to apply to a host or endpoint. The stream UDP inspector associates the `session_timeout` with the UDP hosts defined in the `binder` inspector.

You can have multiple versions of the `stream_udp` inspector in the same network analysis policy.

- Enable packet decoder rules (GID 116) to detect UDP header anomalies.

Stream UDP Inspector Parameters

session_timeout

Specifies the number of seconds that the UDP inspector keeps an inactive UDP stream in the state table. The next time Snort detects a UDP datagram with the same flow key, it checks if the session timeout on the earlier flow has expired. If the timeout has expired, Snort closes the flow and starts a new flow. Snort checks for stale flows associated with the base stream configuration.

Type: integer

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

Default value: `30`

Stream UDP Inspector Rules

The `stream_udp` inspector does not have any associated rules.

Stream UDP Inspector Intrusion Rule Options

The `stream_udp` inspector does not have any intrusion rule options.

CHAPTER 29

Telnet Inspector

Telnet Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`false`

Telnet is an application layer protocol that creates an 8-bit byte communication channel over TCP. Telnet uses a network virtual terminal to communicate between a client and a remote host. A Telnet server uses TCP port 23.

The `telnet` inspector normalizes the Telnet data buffer by detecting the Telnet command sequences and option negotiation. The `telnet` inspector eliminates the Telnet command sequences (RFC 854) from the packet. The `telnet` inspector can detect encrypted Telnet connections by examining the Telnet encryption option (RFC 2946).

Telnet Inspector Parameters

Telnt service configuration

The `binder` inspector defines the telnet `service` configuration. For more information, see the Binder Inspector Overview, on page 13.

Example:

```
[
    {
        "when": {
            "service": "telnet",
            "role": any
        },
        "use": {
            "type": "telnet"
        }
    }
]
```

ayt_attack_thresh

Specifies the maximum number of consecutive Are You There (AYT) telnet commands. The `telnet` inspector detects and alerts on the number of consecutive AYT commands that exceed the `ayt_attack_thresh` value. The `ayt_attack_thresh` parameter addresses specific vulnerabilities related to BSD implementations of telnet. Specify `-1` to disable the `ayt_attack_thresh` parameter. You can enable rule 126:1 to generate events and, in an inline deployment, drop offending packets for this parameter.

Type: integer

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

Default value: `-1`

encrypted_traffic

Specifies whether to detect encrypted telnet traffic. You can enable rule 126:2 to generate events and, in an inline deployment, drop offending packets for this parameter.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

normalize

Specifies whether to normalize telnet traffic. The `telnet` inspector normalizes telnet traffic by eliminating telnet escape sequences. If an enabled intrusion rule specifies a `raw` content parameter, the rule ignores the normalized telnet buffer created by the `telnet` inspector.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

Telnet Inspector Rules

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

Table 35: Telnet Inspector Rules

GID:SID	Rule Message
126:1	consecutive Telnet AYT commands beyond threshold

GID:SID	Rule Message
126:2	Telnet traffic encrypted
126:3	Telnet subnegotiation begin command without subnegotiation end

Telnet Inspector Intrusion Rule Options

The `telnet` inspector does not have any intrusion rule options.