CHAPTER 7

Defining Signatures

This chapter describes how to define and create signatures. It contains the following sections:

Understanding Signatures

Attacks or other misuses of network resources can be defined as network intrusions. Sensors that use a signature-based technology can detect network intrusions. A *signature* is a set of rules that your sensor uses to detect typical intrusive activity, such as DoS attacks. As sensors scan network packets, they use signatures to detect known attacks and respond with actions that you define.

The sensor compares the list of signatures with network activity. When a match is found, the sensor takes an action, such as logging the event or sending an alert. Sensors let you modify existing signatures and define new ones.

Signature-based intrusion detection can produce false positives because certain normal network activity can be misinterpreted as malicious activity. For example, some network applications or operating systems may send out numerous ICMP messages, which a signature-based detection system might interpret as an attempt by an attacker to map out a network segment. You can minimize false positives by tuning your signatures.

To configure a sensor to monitor network traffic for a particular signature, you must enable the signature. By default, the most critical signatures are enabled when you install the signature update. When an attack is detected that matches an enabled signature, the sensor generates an alert, which is stored in the sensor's event store. The alerts, as well as other events, may be retrieved from the event store by web-based clients. By default the sensor logs all Informational alerts or higher.

Some signatures have subsignatures, that is, the signature is divided into subcategories. When you configure a subsignature, changes made to the parameters of one subsignature apply only to that subsignature. For example, if you edit signature 3050 subsignature 1 and change the severity, the severity change applies to only subsignature 1 and not to 3050 2, 3050 3, and 3050 4.

IPS 5.1 contains over 1000 built-in default signatures. You cannot rename or delete signatures from the list of built-in signatures, but you can retire signatures to remove them from the sensing engine. You can later activate retired signatures; however, this process requires the sensing engines to rebuild their

configuration, which takes time and could delay the processing of traffic. You can tune built-in signatures by adjusting several signature parameters. Built-in signatures that have been modified are called *tuned* signatures.

You can create signatures, which are called *custom* signatures. Custom signature IDs begin at 60000. You can configure them for several things, such as matching of strings on UDP connections, tracking of network floods, and scans. Each signature is created using a signature engine specifically designed for the type of traffic being monitored.

Signature Variables

This section describes signature variables, and contains the following topics:

Understanding Signature Variables

When you want to use the same value within multiple signatures, use a variable. When you change the value of a variable, the variables in all signatures are updated. This saves you from having to change the variable repeatedly as you configure signatures.

Note You must preface the variable with a dollar ($) sign to indicate that you are using a variable rather than a string.

Some variables cannot be deleted because they are necessary to the signature system. If a variable is protected, you cannot select it to edit it. You receive an error message if you try to delete protected variables. You can edit only one variable at a time.

Configuring Signature Variables

Use the **variables** command in the signature definition submode to create variables.

The following options apply:

- *variable-name*—Identifies the name assigned to this variable.

  A valid name can only contain numbers or letters. You can also use a hyphen (-) or underscore (_).
- **ip-addr-range**—System-defined variable for grouping IP addresses.

  The valid values are: A.B.C.D-A.B.C.D[,A.B.C.D-A.B.C.D]
- **web-ports**—System-defined variable for ports to look for HTTP traffic.

  To designate multiple port numbers for a single variable, place a comma between the entries. For example, 80, 3128, 8000, 8010, 8080, 8888, 24326.

To configure signature variables, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Create a signature variable for a group of IP addresses:

```
sensor(config-sig)# variables IPADD ip-addr-range 10.1.1.1-10.1.1.24
```

Step 4 Edit the signature variable for web ports:

```
sensor(config-sig)# variables WEBPORTS web-ports 80,3128,8000
```

WEBPORTS has a predefined set of ports where web servers are running, but you can edit the value. This variable affects all signatures that have web ports. The default is 80, 3128, 8000, 8010, 8080, 8888, 24326.

Step 5 Verify the changes:

```
sensor(config-sig)# show settings
   variables (min: 0, max: 256, current: 2)
   -----------------------------------------------
      variable-name: IPADD
      -----------------------------------------------
         ip-addr-range: 10.1.1.1-10.1.1.24
      -----------------------------------------------
      <protected entry>
      variable-name: WEBPORTS
      -----------------------------------------------
         web-ports: 80,3128,8000 default: 80-80,3128-3128,8000-8000,8010-8010,80
80-8080,8888-8888,24326-24326
      -----------------------------------------------
```

Step 6 Exit signature definition submode:

```
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring Signatures

This section describes how to configure signature parameters, and contains the following topics:

Configuring General Signature Parameters

The following options apply to configuring the general parameters of a specific signature:

- **alert-frequency**—Sets the summary options for grouping alerts.

  For the procedure, see Configuring Alert Frequency, page 7-5.

- **alert-severity**—Sets the severity of the alert.

  For the procedure, see Configuring Alert Severity, page 7-6.

- **engine**—Specifies the signature engine. You can assign actions when you are in the engine submode.

  For more information about signature engines, see Appendix B, "Signature Engines." For the procedure for assigning actions, see Assigning Actions to Signatures, page 7-11.

- **event-counter**—Sets the event count.

  For the procedure, see Configuring Event Counter, page 7-8.

- **promisc-delta**—The delta value used to determine the seriousness of the alert.

Caution We do not recommend that you change the promisc-delta setting for a signature.

  Promiscuous delta lowers the RR of certain alerts in promiscuous mode. Because the sensor does not know the attributes of the target system and in promiscuous mode cannot deny packets, it is useful to lower the prioritization of promiscuous alerts (based on the lower RR) so the administrator can focus on investigating higher RR alerts.

  In inline mode, the sensor can deny the offending packets and they never reach the target host, so it does not matter if the target was vulnerable. The attack was not allowed on the network and so we do not subtract from the RR value.

  Signatures that are not service, OS, or application specific have 0 for the promiscuously delta. If the signature is specific to an OS, service, or application, it has a promiscuous delta of 5, 10, or 15 calculated from 5 points for each category.

- **sig-description**—Your description of the signature.
- **sig-fidelity-rating**—Rating of the fidelity of signature.

  For the procedure, see Configuring Signature Fidelity Rating, page 7-9.

- **status**—Sets the status of the signature to enabled or retired.

  For the procedure, see Configuring the Status of Signatures, page 7-10.

Configuring Alert Frequency

Use the **alert-frequency** command in the signature definition submode to configure the alert frequency for a signature.

The following options apply:

- *sig_id*—Identifies the unique numerical value assigned to this signature.

  This value lets the sensor identify a particular signature. The value is 1000 to 65000.

- *subsig_id*—Identifies the unique numerical value assigned to this subsignature.

  A subsignature ID is used to identify a more granular version of a broad signature. The value is 0 to 255.

- **alert-frequency**—How often the sensor alerts you when this signature is firing.

  Specify the following parameters for this signature:

  - **summary-mode**—The way you want the sensor to group the alerts:

    fire-all—Fires an alert on all events.

    fire-once—Fires an alert only once.

    global-summarize—Summarizes an alert so that it only fires once regardless of how many attackers or victims.

    summarize—Summarize all the alerts.

  - **summary-interval**—Time in seconds used in each summary alert.

    The value is 1 to 65535.

  - **summary-key**—Storage type on which to summarize this signature.

    Axxx—Attacker address.

    Axxb—Attacker address and victim port.

    AxBx—Attacker and victim addresses.

    AaBb—Attacker and victim addresses and ports.

    xxBx—Victim address.

  - **specify-global-summary-threshold {yes | no}**—Specifies whether you want to configure a global summary threshold (optional).
  - **global-summary-threshold**—Threshold number of events to take alert into global summary.

To configure the alert frequency parameters of a signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify the signature you want to configure:

```
sensor(config-sig)# signatures 9000 0
```

Step 4 Enter alert frequency submode:

```
sensor(config-sig-sig)# alert-frequency
```

Step 5 Configure the alert frequency of this signature:

a. Configure the summary mode to, for example, fire once:

```
sensor(config-sig-sig-ale)# summary-mode fire-once
sensor(config-sig-sig-ale-fir)# specify-global-summary-threshold yes
sensor(config-sig-sig-ale-fir-yes)# global-summary-threshold 3000
sensor(config-sig-sig-ale-fir-yes)# summary-interval 5000
```

b. Configure the summary key:

```
sensor(config-sig-sig-ale-fir-yes)# exit
sensor(config-sig-sig-ale-fir)# summary-key AxBx
```

c. Verify the settings:

```
sensor(config-sig-sig-ale-fir)# show settings
   fire-once
   -----------------------------------------------
      summary-key: AxBx default: Axxx
      specify-global-summary-threshold
      -----------------------------------------------
         yes
         -----------------------------------------------
            global-summary-threshold: 3000 default: 120
            summary-interval: 5000 default: 15
         -----------------------------------------------
      -----------------------------------------------
   -----------------------------------------------
sensor(config-sig-sig-ale-fir)#
```

Step 6 Exit alert-frequency submode:

```
sensor(config-sig-sig-ale-fir)# exit
sensor(config-sig-sig-ale)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring Alert Severity

Use the **alert-severity** command in the signature definition submode to configure the severity of a signature.

The following options apply:

- *sig_id*—Identifies the unique numerical value assigned to this signature.

  This value lets the sensor identify a particular signature. The value is 1000 to 65000.

- *subsig_id*—Identifies the unique numerical value assigned to this subsignature.

  A subsignature ID is used to identify a more granular version of a broad signature. The value is 0 to 255.

- **alert-severity**—Severity of the alert:
  - **high** —Dangerous alert.
  - **medium**—Medium level alert.

- **low**—Low level alert.
- **informational**—Informational alert.

  This is the default.

To configure the alert severity, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Choose the signature you want to configure:

```
sensor(config-sig)# signatures 9000 0
```

Step 4 Assign the alert severity:

```
sensor(config-sig-sig)# alert-severity medium
```

Step 5 Verify the settings:

```
sensor(config-sig-sig)# show settings
   <protected entry>
   sig-id: 9000
   subsig-id: 0
   -----------------------------------------------
      alert-severity: medium default: informational
      sig-fidelity-rating: 75 <defaulted>
      promisc-delta: 0 <defaulted>
      sig-description
      ----------------------------------------------
         sig-name: Back Door Probe (TCP 12345) <defaulted>
         sig-string-info: SYN to TCP 12345 <defaulted>
         sig-comment: <defaulted>
         alert-traits: 0 <defaulted>
         release: 40 <defaulted>
      ----------------------------------------------
      engine
      ----------------------------------------------
         atomic-ip
         ---------------------------------------------
            event-action: produce-alert <defaulted>
            fragment-status: any <defaulted>
            specify-l4-protocol
            ---------------------------------------------
--MORE--
```

Step 6 Exit signatures submode:

```
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring Event Counter

Use the **event-counter** command in the signature definition submode to configure how the sensor counts events. For example, you can specify that you want the sensor to send an alert only if the same signature fires 5 times for the same address set.

The following options apply:

- **event-count**—The number of times an event must occur before an alert is generated. The valid range is 1 to 65535. The default is 1.
- **event-count-key**—The storage type on which to count events for this signatures.
  - **Axxx**—Attacker address
  - **AxBx**—Attacker and victim addresses
  - **Axxb**—Attacker address and victim port
  - **xxBx**—Victim address
  - **AaBb**—Attacker and victim addresses and ports
- **specify-alert-interval {yes | no}**—Enables alert interval.
  - **alert-interval**—The time in seconds before the event count is reset. The default is 60.

To configure event counter, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Choose the signature for which you want to configure event counter:

```
sensor(config-sig)# signatures 9000 0
```

Step 4 Enter event counter submode:

```
sensor(config-sig-sig)# event-counter
```

Step 5 Configure how many times an event must occur before an alert is generated:

```
sensor(config-sig-sig-eve)# event-count 2
```

Step 6 Configure the storage type on which you want to count events for this signature:

```
sensor(config-sig-sig-eve)# event-count-key AxBx
```

Step 7 (Optional) Enable alert interval:

```
sensor(config-sig-sig-eve)# specify-alert-interval yes
```

Step 8 (Optional) Specify the amount of time in seconds before the event count should be reset:

```
sensor(config-sig-sig-eve-yes)# alert-interval 30
```

Step 9 Verify the settings:

```
sensor(config-sig-sig-eve-yes)# exit
sensor(config-sig-sig-eve)# show settings
   event-counter
   -----------------------------------------------
      event-count: 2 default: 1
```

```
      event-count-key: AxBx default: Axxx
      specify-alert-interval
      ----------------------------------------------
         yes
         ----------------------------------------------
            alert-interval: 30 default: 60
         ----------------------------------------------
      ----------------------------------------------
   ----------------------------------------------
sensor(config-sig-sig-eve)#
```

Step 10 Exit signatures submode:

```
sensor(config-sig-sig-eve)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 11 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring Signature Fidelity Rating

Use the **sig-fidelity-rating** command in the signature definition submode to configure the signature fidelity rating for a signature.

The following option applies:

- **sig-fidelity-rating**—Identifies the weight associated with how well this signature might perform in the absence of specific knowledge of the target.

  The valid value is 0 to 100.

To configure the signature fidelity rating for a signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Choose the signature you want to configure:

```
sensor(config-sig)# signatures 12000 0
```

Step 4 Configure the fidelity rating for this signature:

```
sensor(config-sig-sig)# sig-fidelity-rating 50
```

Step 5 Verify the settings:

```
sensor(config-sig-sig)# show settings
   <protected entry>
   sig-id: 12000
   subsig-id: 0
   ----------------------------------------------
      alert-severity: low <defaulted>
      sig-fidelity-rating: 50 default: 85
      promisc-delta: 15 <defaulted>
      sig-description
      ----------------------------------------------
```

```
        sig-name: Gator Spyware Beacon <defaulted>
        sig-string-info: /download/ User-Agent: Gator <defaulted>
        sig-comment: <defaulted>
        alert-traits: 0 <defaulted>
        release: 71 <defaulted>
      ------------------------------------------------
```

Step 6 Exit signatures submode:

```
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring the Status of Signatures

Use the **status** command in the signature definition submode to specify the status of a specific signature.

The following options apply:

- **status**—Identifies whether the signature is enabled, disabled, or retired.
  - **enabled {true | false}**—Enables the signature.
  - **retired {true | false}**—Retires the signature.

Caution Activating and retiring signatures can take 30 minutes or longer.

To change the status of a signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Choose the signature you want to configure:

```
sensor(config-sig)# signatures 12000 0
```

Step 4 Change the status for this signature:

```
sensor(config-sig-sig)# status
sensor(config-sig-sig-sta)# enabled true
```

Step 5 Verify the settings:

```
sensor(config-sig-sig-sta)# show settings
   status
   ------------------------------------------------
      enabled: true default: false
      retired: false <defaulted>
   ------------------------------------------------
sensor(config-sig-sig-sta)#
```

Step 6 Exit signatures submode:

```
sensor(config-sig-sig-sta)# exit
```

```
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Assigning Actions to Signatures

Use the **event-action** command in the signature definition submode to configure the actions the sensor takes when the signature fires.

The following options apply:

- **deny-attacker-inline** —(Inline mode only) Does not transmit this packet and future packets from the attacker address for a specified period of time.
- **deny-attacker-service-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
- **deny-attacker-victim-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.
- **deny-connection-inline**—(Inline mode only) Does not transmit this packet and future packets on the TCP Flow.
- **deny-packet-inline**—(Inline mode only) Does not transmit this packet.
- **log-attacker-packets**—Starts IP logging of packets containing the attacker address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
- **log-pair-packets**—Starts IP logging of packets containing the attacker-victim address pair. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
- **log-victim-packets**—Starts IP logging of packets containing the victim address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
- **produce-alert** —Writes the event to the Event Store as an alert.
- **produce-verbose-alert**—Includes an encoded dump (possibly truncated) of the offending packet in the alert. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
- **request-block-connection**—Sends a request to ARC to block this connection. You must have blocking devices configured to implement this action.
- **request-block-host**—Sends a request to ARC to block this attacker host. You must have blocking devices configured to implement this action.
- **request-rate-limit**—Sends a rate limit request to ARC to perform rate limiting. You must have rate limiting devices configured to implement this action.
- **request-snmp-trap**—Sends a request to the Notification Application component of the sensor to perform SNMP notification. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected. You must have SNMP configured on the sensor to implement this action.

- **reset-tcp-connection**—Sends TCP resets to hijack and terminate the TCP flow. **Reset TCP Connection** only works on TCP signatures that analyze a single connection. It does not work for sweeps or floods.
- **modify-packet-inline**— Modifies packet data to remove ambiguity about what the end point might do with the packet.

Understaning Deny Packet Inline

For signatures that have deny-packet-inline configured as an action or for an event action override that adds deny-packet-inline as an action, the following actions may be taken:

- droppedPacket
- deniedFlow
- tcpOneWayResetSent

The deny packet inline action is represented as a dropped packet action in the alert. When a deny packet inline occurs for a TCP connection, it is automatically upgraded to a deny connection inline action and seen as a denied flow in the alert. If the IPS denies just one packet, the TCP continues to try to send that same packet again and again, so the IPS denies the entire connection to ensure it never succeeds with the resends.

When a deny connection inline occurs, the IPS also automatically sends a TCP one-way reset, which shows up as a TCP one-way reset sent in the alert. When the IPS denies the connection, it leaves an open connection on both the client (generally the attacker) and the server (generally the victim). Too many open connections can result in resource problems on the victim. So the IPS sends a TCP reset to the victim to close the connection on the victim side (usually the server), which conserves the resources of the victim. It also prevents a failover that would otherwise allow the connection to fail over to a different network path and reach the victim. The IPS leaves the attacker side open and denies all traffic from it.

To configure event actions for a signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator privileges.

Step 2 Enter signature definition mode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
sensor(config-sig)#
```

Step 3 Choose the signature you want to configure:

```
sensor(config-sig)# signatures 1200 0
```

Step 4 Enter the normalizer engine:

```
sensor(config-sig-sig)# engine normalizer
```

Step 5 Configure the event action:

```
sensor(config-sig-sig-nor)# event-action produce-alert|request-snmp-trap
```

Note Each time you configure the event actions for a signature, you overwrite the previous configuration. For example, if you always want to produce an alert when the signature is fired, you must configure it along with the other event actions you want. Use the | symbol to add more than one event action, for example, **product-alert|deny-packet-inline|request-snmp-trap**.

Step 6 Verify the settings:

```
sensor(config-sig-sig-nor)# show settings
   normalizer
   -----------------------------------------------
      event-action: produce-alert|request-snmp-trap default:
produce-alert|deny-packet-inline
```

Step 7 Exit event action submode:

```
sensor(config-sig-sig-nor)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 8 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring AIC Signatures

This section describes the AIC signatures and how to configure them. It contains the following topics:

Overview

AIC provides detailed analysis of web traffic. It provides granular control over HTTP sessions to prevent abuse of the HTTP protocol. It also allows administrative control over applications that attempt to tunnel over specified ports, such as instant messaging, and tunneling applications such as, gotomypc. Inspection and policy checks for P2P and instant messaging is possible if these applications are running over HTTP.

AIC also provides a way to inspect FTP traffic and control the commands being issued. You can enable or disable the predefined signatures or you can create policies through custom signatures.

The AIC engine runs when HTTP traffic is received on AIC web ports. If traffic is web traffic, but not received on the AIC web ports, the Service HTTP engine is executed. AIC inspection can be on any port if it is configured as an AIC web port and the traffic to be inspected is HTTP traffic.

Caution The AIC web ports are regular HTTP web ports. You can turn on AIC web ports to distinguish which ports should watch for regular HTTP traffic and which ports should watch for AIC enforcement. You might use AIC web ports, for example, if you have a proxy on port 82 and you need to monitor it. We recommend that you do not configure separate ports for AIC enforcement.

AIC has the following categories of signatures:

- HTTP request method
  - Define request method
  - Recognized request methods

  For a list of signature IDs and descriptions, see AIC Request Method Signatures, page 7-16.
- MIME type
  - Define content type
  - Recognized content type

  For a list of signature IDs and descriptions, see AIC MIME Define Content Type Signatures, page 7-17. For the procedure for creating a custom MIME signature, see Example AIC MIME-Type Signature, page 7-42.
- Define web traffic policy

  There is one predefined signature, 12674, that specifies the action to take when noncompliant HTTP traffic is seen. The parameter Alarm on Non HTTP Traffic enables the signature. By default this signature is enabled.
- Transfer encodings
  - Associate an action with each method
  - List methods recognized by the sensor
  - Specify which actions need to be taken when a chunked encoding error is seen

  For a list of signature IDs and descriptions, see AIC Transfer Encoding Signatures, page 7-20.
- FTP commands

  Associates an action with an FTP command. For a list of signature IDs and descriptions, see AIC FTP Commands Signatures, page 7-20.

Configuring the Application Policy

Use the **application-policy** command in the signature definition submode to enable the web AIC feature. You can configure the sensor to provide Layer 4 to Layer 7 packet inspection to prevent malicious attacks related to web and FTP services.

The following options apply:

- **ftp-enable {true | false}**—Enables protection for FTP services. Set to true to require the sensor to inspect FTP traffic.

  The default is false.
- **http-policy**—Enables inspection of HTTP traffic.
  - **aic-web-ports**—Variable for ports to look for AIC traffic.

    The valid range is 0 to 65535. A comma-separated list of integer ranges a-b[,c-d] within 0-65535. The second number in the range must be greater than or equal to the first number.

    The default is 80-80,3128-3128,8000-8000,8010-8010,8080-8080,8888-8888,24326-24326.

    Note We recommend that you not configure AIC web ports, but rather use the default web ports.

- **http-enable {true | false}**—Enables protection for web services. Set to true to require the sensor to inspect HTTP traffic for compliance with the RFC.

  The default is false.

- **max-outstanding-http-requests-per-connection**—Maximum allowed HTTP requests per connection.

  The valid value is 1 to 16. The default is 10.

To configure the application policy, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter application policy submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
sensor(config-sig)# application-policy
```

Step 3 Enable inspection of FTP traffic:

```
sensor(config-sig-app)# ftp-enable true
```

Step 4 Configure the HTTP application policy:

a. Enter HTTP application policy submode:

```
sensor(config-sig-app)# http-policy
```

b. Enable HTTP application policy enforcement:

```
sensor(config-sig-app-htt)# http-enable true
```

c. Specify the number of outstanding HTTP requests per connection that can be outstanding without having received a response from the server:

```
sensor(config-sig-app-htt)# max-outstanding-http-requests-per-connection 5
```

d. (Optional) Edit the AIC ports:

```
sensor(config-sig-app-htt)# aic-web-ports 80-80,3128-3128
```

Note We recommend that you not configure AIC web ports, but rather use the default web ports.

Step 5 Verify your settings:

```
sensor(config-sig-app)# show settings
   application-policy
   -----------------------------------------------
      http-policy
      -----------------------------------------------
         http-enable: true default: false
         max-outstanding-http-requests-per-connection: 5 default: 10
         aic-web-ports: 80-80,3128-3128 default: 80-80,3128-3128,8000-8000,8010-
8010,8080-8080,8888-8888,24326-24326
      -----------------------------------------------
      ftp-enable: true default: false
   -----------------------------------------------
sensor(config-sig-app)#
```

Step 6 Exit signature definition submode:

```
sensor(config-sig-app)# exit
```

```
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

AIC Request Method Signatures

The HTTP request method has two categories of signatures:

- Define request method—Allows actions to be associated with request methods. You can expand and modify the signatures (Define Request Method).
- Recognized request methods—Lists methods that are recognized by the sensor (Recognized Request Methods).

Table 7-1 lists the predefined define request method signatures. Enable the signatures that have the predefined method you need. For the procedure for enabling signatures, see Configuring the Status of Signatures, page 7-10.

Table 7-1 Request Method Signatures

Signature ID	Define Request Method
12676	Request Method Not Recognized
12677	Define Request Method PUT
12678	Define Request Method CONNECT
12679	Define Request Method DELETE
12680	Define Request Method GET
12681	Define Request Method HEAD
12682	Define Request Method OPTIONS
12683	Define Request Method POST
12685	Define Request Method TRACE
12695	Define Request Method INDEX
12696	Define Request Method MOVE
12697	Define Request Method MKDIR
12698	Define Request Method COPY
12699	Define Request Method EDIT
12700	Define Request Method UNEDIT
12701	Define Request Method SAVE
12702	Define Request Method LOCK
12703	Define Request Method UNLOCK
12704	Define Request Method REVLABEL
12705	Define Request Method REVLOG
12706	Define Request Method REVADD
12707	Define Request Method REVNUM

Table 7-1 Request Method Signatures (continued)

Signature ID	Define Request Method
12708	Define Request Method SETATTRIBUTE
12709	Define Request Method GETATTRIBUTENAME
12710	Define Request Method GETPROPERTIES
12711	Define Request Method STARTENV
12712	Define Request Method STOPREV

AIC MIME Define Content Type Signatures

There are two policies associated with MIME types:

- Define content type—Associates specific actions for the following cases (Define Content Type):
  - Deny a specific MIME type, such as an image/jpeg
  - Message size violation
  - MIME-type mentioned in header and body do not match
- Recognized content type (Recognized Content Type)

Table 7-2 lists the predefined define content type signatures. Enable the signatures that have the predefined content type you need. For the procedure for enabling signatures, see Configuring the Status of Signatures, page 7-10. You can also create custom define content type signatures. For the procedure, see Example AIC MIME-Type Signature, page 7-42.

Table 7-2 Define Content Type Signatures

Signature ID	Signature Description
12621	Content Type image/gif Invalid Message Length
12622 2	Content Type image/png Verification Failed
12623 0 12623 1 12623 2	Content Type image/tiff Header Check Content Type image/tiff Invalid Message Length Content Type image/tiff Verification Failed
12624 0 12624 1 12624 2	Content Type image/x-3ds Header Check Content Type image/x-3ds Invalid Message Length Content Type image/x-3ds Verification Failed
12626 0 12626 1 12626 2	Content Type image/x-portable-bitmap Header Check Content Type image/x-portable-bitmap Invalid Message Length Content Type image/x-portable-bitmap Verification Failed
12627 0 12627 1 12627 2	Content Type image/x-portable-graymap Header Check Content Type image/x-portable-graymap Invalid Message Length Content Type image/x-portable-graymap Verification Failed
12628 0 12628 1 12628 2	Content Type image/jpeg Header Check Content Type image/jpeg Invalid Message Length Content Type image/jpeg Verification Failed
12629 0 12629 1	Content Type image/cgf Header Check Content Type image/cgf Invalid Message Length

Table 7-2 Define Content Type Signatures (continued)

Signature ID	Signature Description
12631 0	Content Type image/x-xpm Header Check
12631 1	Content Type image/x-xpm Invalid Message Length
12633 0	Content Type audio/midi Header Check
12633 1	Content Type audio/midi Invalid Message Length
12633 2	Content Type audio/midi Verification Failed
12634 0	Content Type audio/basic Header Check
12634 1	Content Type audio/basic Invalid Message Length
12634 2	Content Type audio/basic Verification Failed
12635 0	Content Type audio/mpeg Header Check
12635 1	Content Type audio/mpeg Invalid Message Length
12635 2	Content Type audio/mpeg Verification Failed
12636 0	Content Type audio/x-adpcm Header Check
12636 1	Content Type audio/x-adpcm Invalid Message Length
12636 2	Content Type audio/x-adpcm Verification Failed
12637 0	Content Type audio/x-aiff Header Check
12637 1	Content Type audio/x-aiff Invalid Message Length
12637 2	Content Type audio/x-aiff Verification Failed
12638 0	Content Type audio/x-ogg Header Check
12638 1	Content Type audio/x-ogg Invalid Message Length
12638 2	Content Type audio/x-ogg Verification Failed
12639 0	Content Type audio/x-wav Header Check
12639 1	Content Type audio/x-wav Invalid Message Length
12639 2	Content Type audio/x-wav Verification Failed
12641 0	Content Type text/html Header Check
12641 1	Content Type text/html Invalid Message Length
12641 2	Content Type text/html Verification Failed
12642 0	Content Type text/css Header Check
12642 1	Content Type text/css Invalid Message Length
12643 0	Content Type text/plain Header Check
12643 1	Content Type text/plain Invalid Message Length
12644 0	Content Type text/richtext Header Check
12644 1	Content Type text/richtext Invalid Message Length
12645 0	Content Type text/sgml Header Check
12645 1	Content Type text/sgml Invalid Message Length
12645 2	Content Type text/sgml Verification Failed
12646 0	Content Type text/xml Header Check
12646 1	Content Type text/xml Invalid Message Length
12646 2	Content Type text/xml Verification Failed
12648 0	Content Type video/flc Header Check
12648 1	Content Type video/flc Invalid Message Length
12648 2	Content Type video/flc Verification Failed
12649 0	Content Type video/mpeg Header Check
12649 1	Content Type video/mpeg Invalid Message Length
12649 2	Content Type video/mpeg Verification Failed

Table 7-2 Define Content Type Signatures (continued)

Signature ID	Signature Description
12650 0	Content Type text/xmcd Header Check
12650 1	Content Type text/xmcd Invalid Message Length
12651 0	Content Type video/quicktime Header Check
12651 1	Content Type video/quicktime Invalid Message Length
12651 2	Content Type video/quicktime Verification Failed
12652 0	Content Type video/sgi Header Check
12652 1	Content Type video/sgi Verification Failed
12653 0	Content Type video/x-avi Header Check
12653 1	Content Type video/x-avi Invalid Message Length
12654 0	Content Type video/x-fli Header Check
12654 1	Content Type video/x-fli Invalid Message Length
12654 2	Content Type video/x-fli Verification Failed
12655 0	Content Type video/x-mng Header Check
12655 1	Content Type video/x-mng Invalid Message Length
12655 2	Content Type video/x-mng Verification Failed
12656 0	Content Type application/x-msvideo Header Check
12656 1	Content Type application/x-msvideo Invalid Message Length
12656 2	Content Type application/x-msvideo Verification Failed
12658 0	Content Type application/ms-word Header Check
12658 1	Content Type application/ms-word Invalid Message Length
12659 0	Content Type application/octet-stream Header Check
12659 1	Content Type application/octet-stream Invalid Message Length
12660 0	Content Type application/postscript Header Check
12660 1	Content Type application/postscript Invalid Message Length
12660 2	Content Type application/postscript Verification Failed
12661 0	Content Type application/vnd.ms-excel Header Check
12661 1	Content Type application/vnd.ms-excel Invalid Message Length
12662 0	Content Type application/vnd.ms-powerpoint Header Check
12662 1	Content Type application/vnd.ms-powerpoint Invalid Message Length
12663 0	Content Type application/zip Header Check
12663 1	Content Type application/zip Invalid Message Length
12663 2	Content Type application/zip Verification Failed
12664 0	Content Type application/x-gzip Header Check
12664 1	Content Type application/x-gzip Invalid Message Length
12664 2	Content Type application/x-gzip Verification Failed
12665 0	Content Type application/x-java-archive Header Check
12665 1	Content Type application/x-java-archive Invalid Message Length
12666 0	Content Type application/x-java-vm Header Check
12666 1	Content Type application/x-java-vm Invalid Message Length
12667 0	Content Type application/pdf Header Check
12667 1	Content Type application/pdf Invalid Message Length
12667 2	Content Type application/pdf Verification Failed

Table 7-2 Define Content Type Signatures (continued)

Signature ID	Signature Description
12668 0 12668 1	Content Type unknown Header Check Content Type unknown Invalid Message Length
12669 0 12669 1	Content Type image/x-bitmap Header Check Content Type image/x-bitmap Invalid Message Length
12673 0	Recognized content type

AIC Transfer Encoding Signatures

There are three policies associated with transfer encoding:

- Associate an action with each method (Define Transfer Encoding)
- List methods recognized by the sensor (Recognized Transfer Encodings)
- Specify which actions need to be taken when a chunked encoding error is seen (Chunked Transfer Encoding Error)

Table 7-3 lists the predefined transfer encoding signatures. Enable the signatures that have the predefined transfer encoding method you need. For the procedure for enabling signatures, see Configuring the Status of Signatures, page 7-10.

Table 7-3 Transfer Encoding Signatures

Signature ID	Transfer Encoding Method
12686	Recognized Transfer Encoding
12687	Define Transfer Encoding Deflate
12688	Define Transfer Encoding Identity
12689	Define Transfer Encoding Compress
12690	Define Transfer Encoding GZIP
12693	Define Transfer Encoding Chunked
12694	Chunked Transfer Encoding Error

AIC FTP Commands Signatures

Table 7-4 lists the predefined FTP commands signatures. Enable the signatures that have the predefined FTP command you need. For the procedure for enabling signatures, see Configuring the Status of Signatures, page 7-10.

Table 7-4 FTP Commands Signatures

Signature ID	FTP Command
12900	Unrecognized FTP command
12901	Define FTP command abor
12902	Define FTP command acct
12903	Define FTP command allo
12904	Define FTP command appe

Table 7-4 FTP Commands Signatures (continued)

Signature ID	FTP Command
12905	Define FTP command cdup
12906	Define FTP command cwd
12907	Define FTP command dele
12908	Define FTP command help
12909	Define FTP command list
12910	Define FTP command mkd
12911	Define FTP command mode
12912	Define FTP command nlst
12913	Define FTP command noop
12914	Define FTP command pass
12915	Define FTP command pasv
12916	Define FTP command port
12917	Define FTP command pwd
12918	Define FTP command quit
12919	Define FTP command rein
12920	Define FTP command rest
12921	Define FTP command retr
12922	Define FTP command rmd
12923	Define FTP command rnfr
12924	Define FTP command rnto
12925	Define FTP command site
12926	Define FTP command smnt
12927	Define FTP command stat
12928	Define FTP command stor
12929	Define FTP command stou
12930	Define FTP command stru
12931	Define FTP command syst
12932	Define FTP command type
12933	Define FTP command user

Configuring IP Fragment Reassembly

This section describes IP fragment reassembly, lists the IP fragment reassembly signatures with the configurable parameters, describes how to configure these parameters, and how to configure the method for IP fragment reassembly. It contains the following topics:

Overview

You can configure the sensor to reassemble a datagram that has been fragmented over multiple packets. You can specify boundaries that the sensor uses to determine how many datagram fragments it reassemble and how long to wait for more fragments of a datagram. The goal is to ensure that the sensor does not allocate all its resources to datagrams that cannot be completely reassembled, either because the sensor missed some frame transmissions or because an attack has been launched that is based on generating random fragmented datagrams.

You configure the IP fragment reassembly per signature.

IP Fragment Reassembly Signatures and Configurable Parameters

Table 7-5 lists IP fragment reassembly signatures with the parameters that you can configure for IP fragment reassembly. The IP fragment reassembly signatures are part of the Normalizer engine.

Table 7-5 IP Fragment Reassembly Signatures

IP Fragment Reassembly Signature	Parameter With Default Value
1200 IP Fragmentation Buffer Full	Specify Max Fragments 10000
1201 IP Fragment Overlap	None
1202 IP Fragment Overrun - Datagram Too Long	Specify Max Datagram Size 65536
1203 IP Fragment Overwrite - Data is Overwritten	None
1204 IP Fragment Missing Initial Fragment	None
1205 IP Fragment Too Many Datagrams	Specify Max Partial Datagrams 1000
1206 IP Fragment Too Small	Specify Max Small Frags 2 Specify Min Fragment Size 400
1207 IP Fragment Too Many Datagrams	Specify Max Fragments per Datagram 170
1208 IP Fragment Incomplete Datagram	Specify Fragment Reassembly Timeout 60
1220 Jolt2 Fragment Reassembly DoS attack	Specify Max Last Fragments 4
1225 Fragment Flags Invalid	None

Configuring IP Fragment Reassembly Parameters

To configure IP fragment reassembly parameters for a specific signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify the IP fragment reassembly signature ID and subsignature ID:

```
sensor(config-sig)# signatures 1200 0
```

Step 4 Specify the engine:

```
sensor(config-sig-sig)# engine normalizer
```

Step 5 Enter edit default signatures submode:

```
sensor(config-sig-sig-nor)# edit-default-sigs-only default-signatures-only
```

Step 6 Enable and change the default setting (if desired) of any of the IP fragment reassembly parameter for signature 1200 for example, specifying the maximum fragments:

```
sensor(config-sig-sig-nor-def)# specify-max-fragments yes
sensor(config-sig-sig-nor-def-yes)# max-fragments 20000
```

Step 7 Verify the settings:

```
sensor(config-sig-sig-nor-def-yes)# show settings
   yes
   -----------------------------------------------
      max-fragments: 20000 default: 10000
   -----------------------------------------------
sensor(config-sig-sig-nor-def-yes)#
```

Step 8 Exit signature definition submode:

```
sensor(config-sig-sig-nor-def-yes)# exit
sensor(config-sig-sig-nor-def)# exit
sensor(config-sig-sig-nor)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 9 Press **Enter** for apply the changes or enter **no** to discard them.

Configuring the Method for IP Fragment Reassembly

Use the **fragment-reassembly** command in the signature definition submode to configure the method the sensor will use to reassemble fragments. You can configure this option if your sensor is operating in promiscuous mode. If your sensor is operating in line mode, the method is NT only.

The following options apply:

- **ip-reassemble-mode**—Identifies the method the sensor uses to reassemble the fragments based on the operating system.
  - **nt**—Windows systems.
  - **solaris**—Solaris systems.
  - **linux**—GNU/Linux systems.
  - **bsd**—BSD UNIX systems.

    The default is nt.

To configure IP fragment reassembly, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter fragment reassembly submode:

```
sensor# configure terminal
```

```
sensor(config)# service signature-definition sig0
sensor(config-sig)# fragment-reassembly
```

Step 3 Configure the operating system you want the sensor to use to reassemble IP fragments:

```
sensor(config-sig-fra)# ip-reassemble-mode linux
```

Step 4 Verify the setting:

```
sensor(config-sig-fra)# show settings
   fragment-reassembly
   -----------------------------------------------
      ip-reassemble-mode: linux default: nt
   -----------------------------------------------
sensor(config-sig-fra)#
```

Step 5 Exit signature-definition submode:

```
sensor(config-sig-fra)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 6 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring TCP Stream Reassembly

This section describes TCP stream reassembly, lists the TCP stream reassembly signatures with the configurable parameters, describes how to configure TCP stream signatures, and how to configure the mode for TCP stream reassembly. It contains the following topics:

Overview

You can configure the sensor to monitor only TCP sessions that have been established by a complete three-way handshake. You can also configure how long to wait for the handshake to complete, and how long to keep monitoring a connection where no more packets have been seen. The goal is to prevent the sensor from creating alerts where a valid TCP session has not been established. There are known attacks against sensors that try to get the sensor to generate alerts by simply replaying pieces of an attack. The TCP session reassembly feature helps to mitigate these types of attacks against the sensor.

You configure TCP stream reassembly parameters per signature. You can configure the mode for TCP stream reassembly.

TCP Stream Signatures and Configurable Parameters

Table 7-6 lists TCP stream reassembly signatures with the parameters that you can configure for TCP stream reassembly. TCP stream reassembly signatures are part of the Normalizer engine.

Table 7-6 TCP Stream Reassembly Signatures

Signature ID and Name	Description	Parameter With Default Value and Range	Default Actions
1300 TCP Segment Overwrite[1]	Fires when the data in an overlapping TCP segment (such as a retransmit) sends data that is different from the data already seen on this session	—	Deny Connection Inline Product Alert[2]
1301 TCP Inactive Timeout[3]	Fires when a TCP session has been idle for a TCP Idle Timeout.	TCP Idle Timeout 3600 (15-3600)	None[4]
1302 TCP Embryonic Timeout[5]	Fires when a TCP session has not completes the three-way handshake in TCP embryonic timeout seconds.	TCP Embryonic Timeout 15 (3-300)	None[6]
1303 TCP Closing Timeout[7]	Fires when a TCP session has not closed completely in TCP Closed Timeout seconds after the first FIN.	TCP Closed Timeout 5 (1-60)	None[8]
1304 TCP Max Segments Queued Per Session	Fires when the number of queued out of order segments for a session exceed TCP Max Queue. The segment containing the sequence furthermost from the expected sequence is dropped.	TCP Max Queue 32 (0-128)	Deny Packet Inline Product Alert[9]
1305 TCP Urgent Flag[10]	Fires when the TCP urgent flag is seen	None	Modify Packet Inline is disabled[11]
1306 0 TCP Option Other	Fires when a TCP option in the range of TCP Option Number is seen.	TCP Option Number 6-7,9-255 (Integer Range Allow Multiple 0-255 constraints)	Modify Packet Inline Produce Alert[12]
1306 1 TCP SACK Allowed Option	Fires when a TCP selective ACK allowed option is seen.	—	Modify Packet Inline disabled[13]

Table 7-6 TCP Stream Reassembly Signatures (continued)

Signature ID and Name	Description	Parameter With Default Value and Range	Default Actions
1306 2 TCP SACK Data Option	Fires when a TCP selective ACK data option is seen.	—	Modify Packet Inline disabled[14]
1306 3 TCP Timestamp Option	Fires when a TCP timestamp option is seen.	—	Modify Packet Inline disabled[15]
1306 4 TCP Window Scale Option	Fires when a TCP window scale option is seen.	—	Modify Packet Inline disabled[16]
1307 TCP Window Size Variation	Fires when the right edge of the recv window for TCP moves to the right (decreases).	—	Deny Connection Inline Produce Alert disabled[17]
1308 TTL Varies[18]	Fire when the TTL seen on one direction of a session is higher than the minimum that has been observed	—	Modify Packet Inline[19]
1309 TCP Reserved Bits Set	Fires when the reserved bits (including bits used for ECN) are set on the TCP header.	—	Modify Packet Inline Produce Alert disabled[20]
1310 TCP Retransmit Protection[21]	Fires when the sensor detects that a retransmitted segment has different data than the original segment.	—	Deny Connection Inline Produce Alert[22]
1311 TCP Packet Exceeds MSS	Fires when a packet exceeds the MSS that was exchanged during the three-way handshake.	—	Deny Connection Inline Produce Alert[23]
1312 TCP Min MSS	Fires when the MSS value in a packet containing a SYN flag is less that TCP Min MSS.	TCP Min MSS 400 (0-16000)	Modify Packet Inline disabled[24]
1313 TCP Max MSS	Fires when the MSS value in a packet containing a SYN flag exceed TCP Max MSS	TCP Max MSS1460 (0-16000)	Modify Packet Inline disabled[25]
1314 TCP Data SYN	Fires when TCP payload is sent in the SYN packet.	—	Deny Packet Inline disabled[26]
1315 ACK Without TCP Stream	Fires when an ACK packet is sent that does not belong to a stream.	—	Produce Alert disabled[27]

Table 7-6 TCP Stream Reassembly Signatures (continued)

Signature ID and Name	Description	Parameter With Default Value and Range	Default Actions
1317 Zero Window Probe	Fires when a zero window probe packet is detected.	Modify Packet Inline removes data from the Zero Window Probe packet.	Modify Packet Inline
1330[28] 0 TCP Drop - Bad Checksum	Fires when TCP packet has bad checksum.	Modify Packet Inline corrects the checksum.	Deny Packet Inline
1330 1 TCP Drop - Bad TCP Flags	Fires when TCP packet has bad flag combination.	—	Deny Packet Inline
1330 2 TCP Drop - Urgent Pointer With No Flag	Fires when TCP packet has a URG pointer and no URG flag.	Modify Packet Inline clears the pointer.	Modify Packet Inline disabled
1330 3 TCP Drop - Bad Option List	Fires when TCP packet has a bad option list.	—	Deny Packet Inline
1330 4 TCP Drop - Bad Option Length	Fires when TCP packet has a bad option length.	—	Deny Packet Inline
1330 5 TCP Drop - MSS Option Without SYN	Fires when TCP MSS option is seen in packet without the SYN flag set.	Modify Packet Inline clears the MSS option.	Modify Packet Inline
1330 6 TCP Drop - WinScale Option Without SYN	Fires when TCP window scale option is seen in packet without the SYN flag set.	Modify Packet Inline clears the window scale option.	Modify Packet Inline
1330 7 TCP Drop - Bad WinScale Option Value	Fires when a TCP packet has a bad window scale value.	Modify Packet Inline sets the value to the closest constraint value.	Modify Packet Inline
1330 8 TCP Drop - SACK Allow Without SYN	Fires when the TCP SACK allowed option is seen in a packet without the SYN flags set.	Modify Packet Inline clears the SACK allowed option.	Modify Packet Inline
1330 9 TCP Drop - Data in SYN|ACK	Fires when TCP packet with SYN and ACK flags set also contains data.	—	Deny Packet Inline
1330 10 TCP Drop - Data Past FIN	Fires when TCP data is sequenced after FIN.	—	Deny Packet Inline
1330 11 TCP Drop - Timestamp not Allowed	Fires when TCP packet has timestamp option when timestamp option is not allowed.	—	Deny Packet Inline
1330 12 TCP Drop - Segment Out of Order	Fires when TCP segment is out of order and cannot be queued.	—	Deny Packet Inline

Table 7-6 TCP Stream Reassembly Signatures (continued)

Signature ID and Name	Description	Parameter With Default Value and Range	Default Actions
1330 13 TCP Drop - Invalid TCP Packet	Fires when TCP packet has invalid header.	—	Deny Packet Inline
1330 14 TCP Drop - RST or SYN in window	Fires when TCP packet with RST or SYN flag was sent in the sequence window but was not the next sequence.	—	Deny Packet Inline
1330 15 TCP Drop - Segment Already ACKed	Fires when TCP packet sequence is already ACKed by peer (excluding keepalives).	—	Deny Packet Inline
1330 16 TCP Drop - PAWS Failed	Fires when TCP packet fails PAWS check.	—	Deny Packet Inline
1330 17 TCP Drop - Segment out of State Order	Fires when TCP packet is not proper for the TCP session state.	—	Deny Packet Inline
1330 18 TCP Drop - Segment out of Window	Fires when TCP packet sequence number is outside of allowed window.	—	Deny Packet Inline
3050 Half Open SYN Attack		syn-flood-max-embryonic 5000	
3250 TCP Hijack		max-old-ack 200	
3251 TCP Hijack Simplex Mode		max-old-ack 100	

1. IPS keeps the last 256 bytes in each direction of the TCP session.
2. Modify Packet Inline has no effect on this signature. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
3. The timer is reset to 0 after each packet on the TCP session. by default, this signature does not produce an alert. You can choose to produce alerts for expiring TCP connections if desired. A statistic of total number of expired flows is updated any time a flow expires.
4. Modify Packet Inline, Deny Connection Inline, and Deny Packet Inline have no effect on this signature.
5. The timer starts with the first SYN packet and is not reset. State for the session is reset and any subsequent packets for this flow appear to be out of order (unless it is a SYN).
6. Modify Packet Inline, Deny Connection Inline, and Deny Packet Inline have no effect on this signature.
7. The timer starts with the first FIN packet and is not reset. State for the session is reset and any subsequent packets for this flow appear to be out of order (unless it is a SYN).
8. Modify Packet Inline, Deny Connection Inline, and Deny Packet Inline have no effect on this signature.
9. Modify Packet Inline and Deny Packet Inline have no effect on this signature. Deny Connection Inline drops the current packet and the TCP session.
10. Phrak 57 describes a way to evade security policy using URG pointers. You can normalize the packet when it is in inline mode with this signature.
11. Modify Packet Inline strips the URG flag and zeros the URG pointer from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
12. Modify Packet Inline strips the selected option(s) from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
13. Modify Packet Inline strips the selected ACK allowed option from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.

14. Modify Packet Inline strips the selected ACK allowed option from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
15. Modify Packet Inline strips the timestamp option from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
16. Modify Packet Inline strips the window scale option from the packet. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
17. Modify Packet Inline has no effect on this signature. Deny Connection Inline drops the current packet and the TCP connection. Deny Packet Inline drops the packet.
18. This signature is used to cause TTLs to monotonically decrease for each direction on a session. For example, if TTL 45 is the lowest TTL seen from A to B, then all future packets from A to B will have a maximum of 45 if Modify Packet Inline is set. Each new low TTL becomes the new maximum for packets on that session.
19. Modify Packet Inline ensures that the IP TTL monotonically decreases. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
20. Modify Packet Inline clears all reserved TCP flags. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
21. This signature is not limited to the last 256 bytes like signature 1300.
22. Modify Packet Inline has no effect on this signature. Deny Connection Inline drops the current packet and the TCP connection. Deny Packet Inline drops the packet.
23. Modify Packet Inline has no effect on this signature. Deny Connection Inline drops the current packet and the TCP connection. Deny Packet Inline drops the packet.
24. 2.4.21-15.EL.cisco.1 Modify Packet Inline raises the MSS value to TCP Min MSS. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet 2.4.21-15.EL.cisco.1.
25. Modify Packet Inline lowers the MSS value to TCP Max MSS. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet 2.4.21-15.EL.cisco.1.
26. Modify Packet Inline has no effect on this signature. Deny Connection Inline drops the current packet and the TCP session. Deny Packet Inline drops the packet.
27. Modify Packet Inline, Deny Connection Inline, and Deny Packet Inline have no effect on this signature. By default, the 1330 signatures drop packets for which this signature sends alerts.
28. These subsignatures represent the reasons why the Normalizer might drop a TCP packet. By default these subsignatures drop packets. These subsignatures let you permit packets that fail the checks in the Normalizer through the IPS. The drop reasons have an entry in the TCP statistics. By default these subsignatures do not produce an alert.

Configuring TCP Stream Reassembly Signatures

To configure TCP stream reassembly for a specific signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify the TCP stream reassembly signature ID and subsignature ID:

```
sensor(config-sig)# signatures 1313 0
```

Step 4 Specify the engine:

```
sensor(config-sig-sig)# engine normalizer
```

Step 5 Enter edit default signatures submode:

```
sensor(config-sig-sig-nor)# edit-default-sigs-only default-signatures-only
```

Step 6 Enable and change the default setting (if desired) of the maximum MSS parameter for signature 1313:

```
sensor(config-sig-sig-nor-def)# specify-tcp-max-mss yes
sensor(config-sig-sig-nor-def-yes)# tcp-max-mss 1380
```

Note Changing this parameter from the default of 1460 to 1380 helps prevent fragmentation of traffic going through a VPN tunnel.

Step 7 Verify the settings:

```
sensor(config-sig-sig-nor-def-yes)# show settings
   yes
   -----------------------------------------------
      tcp-max-mss: 1380 default: 1460
   -----------------------------------------------
sensor(config-sig-sig-nor-def-yes)#
```

Step 8 Exit signature definition submode:

```
sensor(config-sig-sig-nor-def-yes)# exit
sensor(config-sig-sig-nor-def)# exit
sensor(config-sig-sig-nor)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 9 Press **Enter** for apply the changes or enter **no** to discard them.

Configuring the Mode for TCP Stream Reassembly

Use the **stream-reassembly** command in the signature definition submode to configure the mode that the sensor will use to reassemble TCP sessions.

The following options apply:

- **tcp-3-way-handshake-required {true | false}**—Specifies that the sensor should only track sessions for which the 3-way handshake is completed.

  The default is true.

- **tcp-reassembly-mode**—Specifies the mode the sensor should use to reassemble TCP sessions.
  - **strict**—Only allows the next expected in the sequence.
  - **loose**—Allows gaps in the sequence.
  - **asym**—Allows asymmetric traffic to be reassembled.

    The default is strict.

Caution The asymmetric option disables TCP window evasion checking.

To configure the TCP stream reassembly parameters, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter TCP stream reassembly submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
sensor(config-sig)# stream-reassembly
```

Step 3 Specify that the sensor should only track session for which the 3-way handshake is completed:

```
sensor(config-sig-str)# tcp-3-way-handshake-required true
```

Step 4 Specify the mode the sensor should use to reassemble TCP sessions:

```
sensor(config-sig-str)# tcp-reassembly-mode strict
```

Step 5 Verify the settings:

```
sensor(config-sig-str)# show settings
   stream-reassembly
   -----------------------------------------------
      tcp-3-way-handshake-required: true default: true
      tcp-reassembly-mode: strict default: strict
   -----------------------------------------------
sensor(config-sig-str)#
```

Step 6 Exit TCP reassembly submode:

```
sensor(config-sig-str)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 7 Press **Enter** to apply the changes or enter **no** to discard them.

Configuring IP Logging

You can configure a sensor to generate an IP session log when the sensor detects an attack. When IP logging is configured as a response action for a signature and the signature is triggered, all packets to and from the source address of the alert are logged for a specified period of time.

Use the **ip-log** command in the signature definition submode to configure IP logging.

The following options apply:

- **ip-log-bytes**—Identifies the maximum number of bytes you want logged.

  The valid value is 0 to 2147483647. The default is 0.
- **ip-log-packets**—Identifies the number of packets you want logged.

  The valid value is 0 to 65535. The default is 0.
- **ip-log-time**—Identifies the duration you want the sensor to log.

  The valid value is 30 to 300 seconds. The default is 30 seconds.

Note When the sensor meets any one of the IP logging conditions, it stops IP logging.

To configure the IP logging parameters, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter IP log submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
sensor(config-sig)# ip-log
```

Step 3 Specify the IP logging parameters:

a. Specify the maximum number of bytes you want logged:

```
sensor(config-sig-ip)# ip-log-bytes 200000
```

b. Specify the number of packets you want logged:

```
sensor(config-sig-ip)# ip-log-packets 150
```

c. Specify the length of time you want the sensor to log:

```
sensor(config-sig-ip)# ip-log-time 60
```

Step 4 Verify the settings:

```
sensor(config-sig-ip)# show settings
   ip-log
   -----------------------------------------------
      ip-log-packets: 150 default: 0
      ip-log-time: 60 default: 30
      ip-log-bytes: 200000 default: 0
   -----------------------------------------------
sensor(config-sig-ip)#
```

Step 5 Exit IP log submode:

```
sensor(config-sig-ip)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 6 Press **Enter** to apply the changes or enter **no** to discard them.

Creating Custom Signatures

This section describes how to create custom signatures, and contains the following topics:

Sequence for Creating a Custom Signature

Use the following sequence when you create a custom signature:

Step 1 Select a signature engine.

Step 2 Assign the signature identifiers:

- Signature ID
- SubSignature ID
- Signature name
- Alert notes (optional)
- User comments (optional)

Step 3 Assign the engine-specific parameters.

The parameters differ for each signature engine, although there is a group of master parameters that applies to each engine.

Step 4 Assign the alert response:

- Signature fidelity rating
- Severity of the alert

Step 5 Assign the alert behavior.

Step 6 Apply the changes.

Example String TCP Signature

The String engine is a generic-based pattern-matching inspection engine for ICMP, TCP, and UDP protocols. The String engine uses a regular expression engine that can combine multiple patterns into a single pattern-matching table allowing for a single search through the data.

There are three String engines: String ICMP, String TCP, and String UDP.

The following example demonstrates how to create a custom String TCP signature.

Note This procedure also applies to String UDP and ICMP signatures.

The following parameters apply to the String TCP engine:

- **default**—Sets the value back to the system default setting.
- **direction**—Direction of the traffic:
  - **from-service**—Traffic from service port destined to client port.
  - **to-service**—Traffic from client port destined to service port.
- **event-action**—Action(s) to perform when alert is triggered:
  - **deny-attacker-inline** —(Inline mode only) does not transmit this packet and future packets from the attacker address for a specified period of time.

  - **deny-attacker-service-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
  - **deny-attacker-victim-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.
  - **deny-connection-inline**—(Inline mode only) Does not transmit this packet and future packets on the TCP Flow.
  - **deny-packet-inline**—(Inline mode only) Does not transmit this packet.
  - **log-attacker-packets**—Starts IP logging of packets containing the attacker address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-pair-packets**—Starts IP logging of packets containing the attacker-victim address pair. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-victim-packets**—Starts IP logging of packets containing the victim address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **produce-alert** —Writes the event to the Event Store as an alert.
  - **produce-verbose-alert**—Includes an encoded dump (possibly truncated) of the offending packet in the alert. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **request-block-connection**—Sends a request to ARC to block this connection. You must have blocking devices configured to implement this action.
  - **request-block-host**—Sends a request to ARC to block this attacker host. You must have blocking devices configured to implement this action.
  - **request-rate-limit**—Sends a rate limit request to ARC to perform rate limiting. You must have rate limiting devices configured to implement this action.
  - **request-snmp-trap**—Sends a request to the Notification Application component of the sensor to perform SNMP notification. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected. You must have SNMP configured on the sensor to implement this action.
  - **reset-tcp-connection**—Sends TCP resets to hijack and terminate the TCP flow. **Reset TCP Connection** only works on TCP signatures that analyze a single connection. It does not work for sweeps or floods.
  - **modify-packet-inline**— Modifies packet data to remove ambiguity about what the end point might do with the packet.
- **no**—Removes an entry or selection setting.
- **regex-string** —A regular expression to search for in a single TCP packet.
- **service-ports**—Ports or port ranges where the target service may reside.

  The valid range is 0 to 65535. It is a separated list of integer ranges a-b[,c-d] within 0 to 65535. The second number in the range must be greater than or equal to the first number.
- **specify-exact-match-offset {yes | no}**—(Optional) Enables exact-match-offset.
- **specify-min-match-length {yes | no}**—(Optional) Enables min-match-length.
- **strip-telnet-options**—Strips Telnet option characters from data before searching.
- **swap-attacker-victim {true | false}**—Swaps the attacker and victim addresses and ports (source and destination) in the alert message and for any actions taken. The default is false for no swapping.

To create a signature based on the String TCP engine, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify a signature ID and subsignature ID for the signature:

```
sensor(config-sig)# signatures 60025 0
```

Custom signatures are in the range of 60000 to 65000.

Step 4 Enter signature description submode:

```
sensor(config-sig-sig)# sig-description
```

Step 5 Specify a name for the new signature:

```
sensor(config-sig-sig-sig)# sig-name This is my new name
```

You can also specify a additional comments about the sig using the **sig-comment** command or additional information about the signature using the **sig-string-info** command.

Step 6 Exit signature description submode:

```
sensor(config-sig-sig-sig)# exit
```

Step 7 Specify the string TCP engine:

```
sensor(config-sig-sig)# engine string-tcp
```

Step 8 Specify the service ports:

```
sensor(config-sig-sig-str)# service-ports 23
```

Step 9 Specify the direction:

```
sensor(config-sig-sig-str)# direction to-service
```

Step 10 Specify the regex string to search for in the TCP packet:

```
sensor(config-sig-sig-str)# regex-string This-is-my-new-Sig-regex
```

Step 11 You can change the event actions if needed according to your security policy using the **event-action** command. The default event action is **produce-alert**.

Step 12 You can modify the following optional parameters for this custom String TCP signature:

- **specify-exact-match-offset**
- **specify-min-match-length**
- **strip-telnet-options**
- **swap-attacker-victim**.

Step 13 Verify the settings:

```
sensor(config-sig-sig-str)# show settings
   string-tcp
   -----------------------------------------------
      event-action: produce-alert <defaulted>
      strip-telnet-options: false <defaulted>
      specify-min-match-length
      -----------------------------------------------
```

```
            no
            ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
         regex-string: This-is-my-new-Sig-regex
         service-ports: 23
         direction: to-service default: to-service
         specify-exact-match-offset
         ------------------------------------------------
            no
            ------------------------------------------------
               specify-max-match-offset
               ------------------------------------------------
                  no
                  ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
               specify-min-match-offset
               ------------------------------------------------
                  no
                  ------------------------------------------------
                  ------------------------------------------------
               ------------------------------------------------
            ------------------------------------------------
         ------------------------------------------------
         swap-attacker-victim: false <defaulted>
      ------------------------------------------------
sensor(config-sig-sig-str)#
```

Step 14 Exit signature definition submode:

```
sensor(config-sig-sig-str)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 15 Press **Enter** to apply the changes or enter **no** to discard them.

Example Service HTTP Signature

The Service HTTP engine is a service-specific string-based pattern-matching inspection engine. The HTTP protocol is one of the most commonly used in today's networks. In addition, it requires the most amount of preprocessing time and has the most number of signatures requiring inspection making it critical to the system's overall performance.

The Service HTTP engine uses a Regex library that can combine multiple patterns into a single pattern-matching table allowing a single search through the data. This engine searches traffic directed to web services only to web services, or HTTP requests. You cannot inspect return traffic with this engine. You can specify separate web ports of interest in each signature in this engine.

HTTP deobfuscation is the process of decoding an HTTP message by normalizing encoded characters to ASCII equivalent characters. It is also known as ASCII normalization.

Before an HTTP packet can be inspected, the data must be deobfuscated or normalized to the same representation that the target system sees when it processes the data. It is ideal to have a customized decoding technique for each host target type, which involves knowing what operating system and web server version is running on the target. The Service HTTP engine has default deobfuscation behavior for the Microsoft IIS web server.

The following example demonstrates how to create a custom Service HTTP signature.

The following options apply to the Service HTTP engine:

- **de-obfuscate {true | false}**—Applies anti-evasive deobfuscation before searching.
- **default**—Sets the value back to the system default setting.
- **event-action** —Action(s) to perform when alert is triggered:
  - **deny-attacker-inline** —(Inline mode only) does not transmit this packet and future packets from the attacker address for a specified period of time.
  - **deny-attacker-service-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
  - **deny-attacker-victim-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.
  - **deny-connection-inline**—(Inline mode only) Does not transmit this packet and future packets on the TCP Flow.
  - **deny-packet-inline**—(Inline mode only) Does not transmit this packet.
  - **log-attacker-packets**—Starts IP logging of packets containing the attacker address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-pair-packets**—Starts IP logging of packets containing the attacker-victim address pair. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-victim-packets**—Starts IP logging of packets containing the victim address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **produce-alert** —Writes the event to the Event Store as an alert.
  - **produce-verbose-alert**—Includes an encoded dump (possibly truncated) of the offending packet in the alert. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **request-block-connection**—Sends a request to ARC to block this connection. You must have blocking devices configured to implement this action.
  - **request-block-host**—Sends a request to ARC to block this attacker host. You must have blocking devices configured to implement this action.
  - **request-rate-limit**—Sends a rate limit request to ARC to perform rate limiting. You must have rate limiting devices configured to implement this action.
  - **request-snmp-trap**—Sends a request to the Notification Application component of the sensor to perform SNMP notification. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected. You must have SNMP configured on the sensor to implement this action.
  - **reset-tcp-connection**—Sends TCP resets to hijack and terminate the TCP flow. **Reset TCP Connection** only works on TCP signatures that analyze a single connection. It does not work for sweeps or floods.
  - **modify-packet-inline**— Modifies packet data to remove ambiguity about what the end point might do with the packet.
- **max-field-sizes** —Grouping for maximum field sizes:
  - **specify-max-arg-field-length {yes | no}**—Enables max-arg-field-length (optional).
  - **specify-max-header-field-length {yes | no}]**—Enables max-header-field-length (optional).
  - **specify-max-request-length {yes | no}**—Enables max-request-length (optional).

  - **specify-max-uri-field-length {yes | no}**—Enables max-uri-field-length (optional).
- **no**—Removes an entry or selection setting.
- **regex**—Regular expression grouping:
  - **specify-arg-name-regex**—Enables arg-name-regex (optional).
  - **specify-header-regex** —Enables header-regex (optional).
  - **specify-request-regex**—Enables request-regex (optional).
  - **specify-uri-regex**—Enables uri-regex (optional).
- **service-ports** —A comma-separated list of ports or port ranges where the target service may reside.
- **swap-attacker-victim {true | false}**—Swaps the attacker and victim addresses and ports (source and destination) in the alert message and for any actions taken. The default is false for no swapping.

To create a custom signature based on the Service HTTP engine, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify a signature ID and a subsignature ID for the signature:

```
sensor(config-sig)# signatures 63000 0
```

Custom signatures are in the range of 60000 to 65000.

Step 4 Enter signature description mode:

```
sensor(config-sig-sig)# sig-description
```

Step 5 Specify a signature name:

```
sensor(config-sig-sig-sig)# sig-name myWebSig
```

Step 6 Specify the alert traits:

```
sensor(config-sig-sig-sig)# alert-traits 2
```

The valid range is from 0 to 65535.

Step 7 Exit signature description submode:

```
sensor(config-sig-sig-sig)# exit
```

Step 8 Assign the alert frequency:

```
sensor(config-sig-sig)# alert-frequency
sensor(config-sig-sig-ale)# summary-mode fire-all
sensor(config-sig-sig-ale-fir)# summary-key Axxx
sensor(config-sig-sig-ale-fir)# specify-summary-threshold yes
sensor(config-sig-sig-ale-fir-yes)# summary-threshold 200
```

Step 9 Exit alert frequency submode:

```
sensor(config-sig-sig-ale-fir-yes)# exit
sensor(config-sig-sig-ale-fir)# exit
sensor(config-sig-sig-ale)# exit
```

Step 10 Configure the signature to apply anti-evasive deobfuscation before searching:

```
sensor(config-sig-sig)# engine service-http
sensor(config-sig-sig-ser)# de-obfuscate true
```

Step 11 Configure the regex parameters:

```
sensor(config-sig-sig)# engine service-http
sensor(config-sig-sig-ser)# regex
sensor(config-sig-sig-ser-reg)# specify-uri-regex yes
sensor(config-sig-sig-ser-reg-yes)# uri-regex [Mm][Yy][Ff][Oo][Oo]
```

Step 12 Exit regex submode:

```
sensor(config-sig-sig-ser-reg-yes)# exit
sensor(config-sig-sig-ser-reg-)# exit
```

Step 13 Configure the service ports using the signature variable WEBPORTS:

```
sensor(config-sig-sig-ser)# service-ports $WEBPORTS
```

Step 14 Exit signature definition submode:

```
sensor(config-sig-sig-ser)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 15 Press **Enter** to apply the changes or enter **no** to discard them.

Example MEG Signature

The Meta engine defines events that occur in a related manner within a sliding time interval. This engine processes events rather than packets. As signature events are generated, the Meta engine inspects them to determine if they match any or several Meta definitions. The Meta engine generates a signature event after all requirements for the event are met.

All signature events are handed off to the Meta engine by SEAP. SEAP hands off the event after processing the minimum hits option. Summarization and event action are processed after the Meta engine has processed the component events. For more information about SEAP, see Signature Event Action Processor, page 6-2.

Caution A large number of Meta signatures could adversely affect overall sensor performance.

The following example demonstrates how to create a MEG signature based on the Meta engine.

Note The Meta engine is different from other engines in that it takes alerts as input where most engines take packets as input. For more information on the Meta engine, see Meta Engine, page B-13.

The following options apply to the Meta signature engine:

- **component-list**—List of Meta components.
  - **edit**—Edits an existing entry in the list.
  - **insert** *name1*—Inserts a new entry into the list.

  - **move**—Moves an entry in the list.
  - **begin**—Places the entry at the beginning of the active list.
  - **end**—Places the entry at the end of the active list.
  - **inactive**—Places the entry into the inactive list.
  - **before**—Places the entry before the specified entry.
  - **after**—Places the entry after the specified entry.
- **component-count**—Number of times component must fire before this component is satisfied.
- **component-sig-id**—Signature ID of the signature to match this component on.
- **component-subsig-id**—Subsignature ID of the signature to match this component on.
- **component-list-in-order {true | false}**—Whether or not to have the component list fire in order.
- **event-action**—Action(s) to perform when alert is triggered:
  - **deny-attacker-inline** —(Inline mode only) does not transmit this packet and future packets from the attacker address for a specified period of time.
  - **deny-attacker-service-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
  - **deny-attacker-victim-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.
  - **deny-connection-inline**—(Inline mode only) Does not transmit this packet and future packets on the TCP Flow.
  - **deny-packet-inline**—(Inline mode only) Does not transmit this packet.
  - **log-attacker-packets**—Starts IP logging of packets containing the attacker address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-pair-packets**—Starts IP logging of packets containing the attacker-victim address pair. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-victim-packets**—Starts IP logging of packets containing the victim address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **produce-alert** —Writes the event to the Event Store as an alert.
  - **produce-verbose-alert**—Includes an encoded dump (possibly truncated) of the offending packet in the alert. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **request-block-connection**—Sends a request to ARC to block this connection. You must have blocking devices configured to implement this action.
  - **request-block-host**—Sends a request to ARC to block this attacker host. You must have blocking devices configured to implement this action.
  - **request-rate-limit**—Sends a rate limit request to ARC to perform rate limiting. You must have rate limiting devices configured to implement this action.
  - **request-snmp-trap**—Sends a request to the Notification Application component of the sensor to perform SNMP notification. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected. You must have SNMP configured on the sensor to implement this action.
  - **reset-tcp-connection**—Sends TCP resets to hijack and terminate the TCP flow. **Reset TCP Connection** only works on TCP signatures that analyze a single connection. It does not work for sweeps or floods.

  - **modify-packet-inline**— Modifies packet data to remove ambiguity about what the end point might do with the packet.
- **meta-key**—Storage type for the Meta signature.
  - **AaBb**—Attacker and victim addresses and ports.
  - **AxBx**—Attacker and victim addresses.
  - **Axxx**—Attacker address.
  - **xxBx**—Victim address.
- **meta-reset-interval**—Time in seconds to reset the Meta signature.

  The valid range is 0 to 3600 seconds. The default is 60 seconds.

Note Signature 64000 subsignature 0 will fire when it sees the alerts from signature 2000 subsignature 0 and signature 3000 subsignature 0 on the same source address. The source address selection is a result of the meta key default value of Axxx. You can change the behavior by changing the meta key setting to xxBx (destination address) for example.

To create a MEG signature based on the Meta engine, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter signature definition submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

Step 3 Specify a signature ID and a subsignature ID for the signature:

```
sensor(config-sig)# signatures 64000 0
```

Custom signatures are in the range of 60000 to 65000.

Step 4 Specify the signature engine:

```
sensor(config-sig-sig)# engine meta
```

Step 5 Insert a MEG signature (named c1) at the beginning of the list:

```
sensor(config-sig-sig-met)# component-list insert c1 begin
```

Step 6 Specify the signature ID of the signature on which to match this component:

```
sensor(config-sig-sig-met-com)# component-sig-id 2000
```

Step 7 Exit component list submode:

```
sensor(config-sig-sig-met-com)# exit
```

Step 8 Insert another MEG signature (named c2) at the end of the list:

```
sensor(config-sig-sig-met)# component-list insert c2 end
```

Step 9 Specify the signature ID of the signature on which to match this component

```
sensor(config-sig-sig-met-com)# component-sig-id 3000
```

Step 10 Verify the settings:

```
sensor(config-sig-sig-met-com)# exit
sensor(config-sig-sig-met)# show settings
```

```
   meta
   ------------------------------------------------
      event-action: produce-alert <defaulted>
      meta-reset-interval: 60 <defaulted>
      component-list (min: 1, max: 8, current: 2 - 2 active, 0 inactive)
      ------------------------------------------------
      ACTIVE list-contents
      ------------------------------------------------
         NAME: c1
         ------------------------------------------------
            component-sig-id: 2000
            component-subsig-id: 0 <defaulted>
            component-count: 1 <defaulted>
         ------------------------------------------------
         ------------------------------------------------
         NAME: c2
         ------------------------------------------------
            component-sig-id: 3000
            component-subsig-id: 0 <defaulted>
            component-count: 1 <defaulted>
         ------------------------------------------------
         ------------------------------------------------
      ------------------------------------------------
      meta-key
      ------------------------------------------------
         Axxx
         ------------------------------------------------
            unique-victims: 1 <defaulted>
         ------------------------------------------------
      ------------------------------------------------
      component-list-in-order: false <defaulted>
   ------------------------------------------------
sensor(config-sig-sig-met)#
```

Step 11 Exit signature definition submode:

```
sensor(config-sig-sig-met)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 12 Press **Enter** to apply the changes or enter **no** to discard them.

Example AIC MIME-Type Signature

The following example demonstrates how to create a MIME-type signature based on the AIC engine.

The following options apply:

- **event-action**—Action(s) to perform when alert is triggered.
  - **deny-attacker-inline** —(Inline mode only) does not transmit this packet and future packets from the attacker address for a specified period of time.
  - **deny-attacker-service-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
  - **deny-attacker-victim-pair-inline**—(Inline only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.

  - **deny-connection-inline**—(Inline mode only) Does not transmit this packet and future packets on the TCP Flow.
  - **deny-packet-inline**—(Inline mode only) Does not transmit this packet.
  - **log-attacker-packets**—Starts IP logging of packets containing the attacker address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-pair-packets**—Starts IP logging of packets containing the attacker-victim address pair. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **log-victim-packets**—Starts IP logging of packets containing the victim address. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **produce-alert** —Writes the event to the Event Store as an alert.
  - **produce-verbose-alert**—Includes an encoded dump (possibly truncated) of the offending packet in the alert. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected.
  - **request-block-connection**—Sends a request to ARC to block this connection. You must have blocking devices configured to implement this action.
  - **request-block-host**—Sends a request to ARC to block this attacker host. You must have blocking devices configured to implement this action.
  - **request-rate-limit**—Sends a rate limit request to ARC to perform rate limiting. You must have rate limiting devices configured to implement this action.
  - **request-snmp-trap**—Sends a request to the Notification Application component of the sensor to perform SNMP notification. This action causes an alert to be written to the Event Store, even if **produce-alert** is not selected. You must have SNMP configured on the sensor to implement this action.
  - **reset-tcp-connection**—Sends TCP resets to hijack and terminate the TCP flow. **Reset TCP Connection** only works on TCP signatures that analyze a single connection. It does not work for sweeps or floods.
  - **modify-packet-inline**— Modifies packet data to remove ambiguity about what the end point might do with the packet.

- **no**—Removes an entry or selection setting
- **signature-type**—Type of signature desired
  - **content-types**—Content-types
  - **define-web-traffic-policy**—Defines web traffic policy
  - **max-outstanding-requests-overrun**—Inspects for large number of outstanding HTTP requests
  - **msg-body-pattern**—Message body pattern
  - **request-methods**—Signature types that deal with request methods
  - **transfer-encodings**—Signature types that deal with transfer encodings

To define a MIME-type policy signature, follow these steps:

Step 1 Log in to the CLI using an account with administrator or operator privileges.

Step 2 Enter application policy enforcement submode:

```
sensor# configure terminal
sensor(config)# service signature-definition sig0
```

```
sensor(config-sig)# signatures 60001 0
sensor(config-sig-sig)# engine application-policy-enforcement-http
```

Step 3 Specify the event action:

```
sensor(config-sig-sig-app)# event-action produce-alert|log-pair-packets
```

Step 4 Define the signature type:

```
sensor(config-sig-sig-app)# signature-type content-type define-content-type
```

Step 5 Define the content type:

```
sensor(config-sig-sig-app-def)# name MyContent
```

Step 6 Verify your settings:

```
sensor(config-sig-sig-app-def)# show settings
-> define-content-type
   -----------------------------------------------
      name: MyContent
*---> content-type-details
      -----------------------------------------------
      -----------------------------------------------
   -----------------------------------------------
sensor(config-sig-sig-app-def)#
```

Step 7 Exit signatures submode:

```
sensor(config-sig-sig-app-def)# exit
sensor(config-sig-sig-app)# exit
sensor(config-sig-sig)# exit
sensor(config-sig)# exit
Apply Changes:?[yes]:
```

Step 8 Press **Enter** to apply the changes or enter **no** to discard them.