



## CHAPTER **B**

# Signature Engines

---

This appendix describes the IPS signature engines. It contains the following sections:

- [Understanding Signature Engines, page B-1](#)
- [Master Engine, page B-3](#)
- [AIC Engine, page B-7](#)
- [Atomic Engine, page B-9](#)
- [Fixed Engine, page B-12](#)
- [Flood Engine, page B-15](#)
- [Meta Engine, page B-16](#)
- [Multi String Engine, page B-17](#)
- [Normalizer Engine, page B-19](#)
- [Service Engines, page B-20](#)
- [State Engine, page B-37](#)
- [String Engines, page B-39](#)
- [Sweep Engines, page B-42](#)
- [Traffic Anomaly Engine, page B-44](#)
- [Traffic ICMP Engine, page B-46](#)
- [Trojan Engines, page B-47](#)

## Understanding Signature Engines

A signature engine is a component of the Cisco IPS that is designed to support many signatures in a certain category. An engine is composed of a parser and an inspector. Each engine has a set of parameters that have allowable ranges or sets of values.



### Note

---

The Cisco IPS 6.1 engines support a standardized Regex.

---

Cisco IPS 6.1 contains the following signature engines:

- **AIC**—Provides thorough analysis of web traffic. The AIC engine provides granular control over HTTP sessions to prevent abuse of the HTTP protocol. It allows administrative control over applications, such as instant messaging and gotomypc, that try to tunnel over specified ports. You can also use AIC to inspect FTP traffic and control the commands being issued. There are two AIC engines: AIC FTP and AIC HTTP.
- **Atomic**—The Atomic engines are now combined into two engines with multi-level selections. You can combine Layer 3 and Layer 4 attributes within one signature, for example IP + TCP. The Atomic engine uses the standardized Regex support.
  - **Atomic ARP**—Inspects Layer 2 ARP protocol. The Atomic ARP engine is different because most engines are based on Layer 3 IP protocol.
  - **Atomic IP**—Inspects IP protocol packets and associated Layer 4 transport protocols. This engine lets you specify values to match for fields in the IP and Layer 4 headers, and lets you use Regex to inspect Layer 4 payloads.




---

**Note** All IP packets are inspected by the Atomic IP engine. This engine replaces the 4.x Atomic ICMP, Atomic IP Options, Atomic L3 IP, Atomic TCP, and Atomic UDP engines.

---

- **Atomic IPv6**—Detects two IOS vulnerabilities that are stimulated by malformed IPv6 traffic.
- **Flood**—Detects ICMP and UDP floods directed at hosts and networks. There are two Flood engines: Flood Host and Flood Net.
- **Meta**—Defines events that occur in a related manner within a sliding time interval. This engine processes events rather than packets.
- **Multi String**—Inspects Layer 4 transport protocols and payloads by matching several strings for one signature. This engine inspects stream-based TCP and single UDP and ICMP packets.
- **Normalizer**—Configures how the IP and TCP normalizer functions and provides configuration for signature events related to the IP and TCP normalizer. Allows you to enforce RFC compliance.
- **Service**—Deals with specific protocols. Service engine has the following protocol types:
  - **DNS**—Inspects DNS (TCP and UDP) traffic.
  - **FTP**—Inspects FTP traffic.
  - **Generic**—Decodes custom service and payload.
  - **Generic Advanced**—Analyzes traffic based on the mini-programs that are written to parse the packets.
  - **H225**—Inspects VoIP traffic. Helps the network administrator make sure the SETUP message coming in to the VoIP network is valid and within the bounds that the policies describe. Is also helps make sure the addresses and Q.931 string fields such as url-ids, email-ids, and display information adhere to specific lengths and do not contain possible attack patterns.
  - **HTTP**—Inspects HTTP traffic. The WEBPORTS variable defines inspection port for HTTP traffic.
  - **IDENT**—Inspects IDENT (client and server) traffic.
  - **MSRPC**—Inspects MSRPC traffic.
  - **MSSQL**—Inspects Microsoft SQL traffic.
  - **NTP**—Inspects NTP traffic.

- RPC—Inspects RPC traffic.
- SMB—Inspects SMB traffic.
- SMB Advanced—Processes Microsoft SMB and Microsoft RPC over SMB packets.
- SNMP—Inspects SNMP traffic.
- SSH—Inspects SSH traffic.
- TNS—Inspects TNS traffic.
- State—Stateful searches of strings in protocols such as SMTP. The state engine now has a hidden configuration file that is used to define the state transitions so new state definitions can be delivered in a signature update.
- String—Searches on Regex strings based on ICMP, TCP, or UDP protocol. There are three String engines: String ICMP, String TCP, and String UDP.
- Sweep—Analyzes sweeps from a single host (ICMP and TCP), from destination ports (TCP and UDP), and multiple ports with RPC requests between two nodes. There are two Sweep engines: Sweep and Sweep Other TCP.
- Traffic Anomaly—Inspects TCP, UDP, and other traffic for worms.
- Traffic ICMP—Analyzes nonstandard protocols, such as TFN2K, LOKI, and DDOS. There are only two signatures with configurable parameters.
- Trojan—Analyzes traffic from nonstandard protocols, such as BO2K andTFN2K. There are three Trojan engines: Bo2k, Tfn2k, and UDP. There are no user-configurable parameters in these engines.

## Master Engine

The Master engine provides structures and methods to the other engines and handles input from configuration and alert output. This section describes the Master engine, and contains the following topics:

- [General Parameters, page B-4](#)
- [Alert Frequency, page B-5](#)
- [Event Actions, page B-6](#)

## General Parameters

The following parameters are part of the Master engine and apply to all signatures (if it makes sense for that signature engine).

Table B-1 lists the general master engine parameters.

**Table B-1** Master Engine Parameters

| Parameter                      | Description  | Value                                  |
|--------------------------------|--|--|
| Alert Severity                 | Severity of the alert: <ul style="list-style-type: none"> <li>• Dangerous alert</li> <li>• Medium-level alert</li> <li>• Low-level alert</li> <li>• Informational alert</li> </ul>   | High<br>Medium<br>Low<br>Informational |
| Sig Fidelity Rating            | Rating of the fidelity of this signature.  | 0 to 100                               |
| Promiscuous Delta <sup>1</sup> | Delta value used to determine seriousness of the alert.  | 0 to 30                                |
| Signature Name                 | Name of the signature.   | <i>sig-name</i>                        |
| Alert Notes                    | Additional information about this signature that will be included in the alert message.  | <i>alert-notes</i>                     |
| User Comments                  | Comments about this signature.   | <i>comments</i>                        |
| Alert Traits                   | Traits you want to document about this signature.  | 0 to 65335                             |
| Release                        | The release in which the signature was most recently updated.  | <i>release</i>                         |
| Engine                         | Specifies the engine the signature belongs to.   | —                                      |
| Event Count                    | Number of times an event must occur before an alert is generated.  | 1 to 65535                             |
| Event Count Key                | The storage type on which to count events for this signature: <ul style="list-style-type: none"> <li>• Attacker address</li> <li>• Attacker and victim addresses</li> <li>• Attacker address and victim port</li> <li>• Victim address</li> <li>• Attacker and victim addresses and ports</li> </ul> | Axxx<br>AxBx<br>Axxb<br>xxBx<br>AaBb   |
| Specify Alert Interval         | Enables alert interval.  | yes   no                               |
| Alert Interval                 | Time in seconds before the event count is reset.   | 2 to 1000                              |
| Status                         | Whether the signature is enabled or disabled, active or retired.   | enabled<br>retired                     |

1. Promiscuous Delta lowers the risk rating 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 risk rating) so the administrator can focus on investigating higher risk rating 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 risk rating value. Signatures that are not service, OS, or application-specific have 0 for the Promiscuous 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.

**Caution**

We do not recommend that you change the Promiscuous Delta setting for a signature.

## Alert Frequency

The purpose of the alert frequency parameter is to reduce the volume of the alerts written to the Event Store to counter IDS DoS tools, such as stick. There are four modes: Fire All, Fire Once, Summarize, and Global Summarize. The summary mode is changed dynamically to adapt to the current alert volume. For example, you can configure the signature to Fire All, but after a certain threshold is reached, it starts summarizing.

Table B-2 lists the alert frequency parameters.

**Table B-2** Master Engine Alert Frequency Parameters

| Parameter                        | Description   | Value                                |
|----------------------------------|---|--------------------------------------|
| Alert Frequency                  | Summary options for grouping alerts.  | —                                    |
| Summary Mode                     | Mode used for summarization.  | —                                    |
| 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                        | Summarizes alerts.  | —                                    |
| Specify Summary Threshold        | (Optional) Enables summary threshold.   | Yes   No                             |
| Summary Threshold                | Threshold number of alerts to send signature into summary mode.   | 0 to 65535                           |
| Specify Global Summary Threshold | Enable global summary threshold.  | Yes   No                             |
| Global Summary Threshold         | Threshold number of events to take alerts into global summary.  | 1 to 65535                           |
| Summary Interval                 | Time in seconds used in each summary alert.   | 1 to 1000                            |
| Summary Key                      | The storage type on which to summarize this signature: <ul style="list-style-type: none"> <li>Attacker address</li> <li>Attacker and victim addresses</li> <li>Attacker address and victim port</li> <li>Victim address</li> <li>Attacker and victim addresses and ports</li> </ul> | Axxx<br>AxBx<br>Axxb<br>xxBx<br>AaBb |

## Event Actions


**Note**

Most of the following event actions belong to each signature engine unless they are not appropriate for that particular engine.

The following event action parameters belong to each signature engine (if it makes sense for that signature engine):

- Alert and Log Actions
  - Product Alert—Writes an alert to Event Store.
  - Produce Verbose Alert—Includes an encoded dump (possibly truncated) of the offending packet in the alert.
  - Log Attacker Packets—Starts IP logging of packets containing the attacker address and sends an alert.
  - Log Victim Packets—Starts IP logging of packets containing the victim address and sends an alert.
  - Log Attacker/Victim Pair Packets—(inline mode only) Starts IP logging of packets containing the attacker/victim address pair.
  - Request SNMP Trap—Sends request to NotificationApp to perform SNMP notification.
- Deny Actions
  - Deny Packet Inline—(inline mode only) Does not transmit this packet.



**Note** You cannot delete the event action override for Deny Packet Inline because it is protected. If you do not want to use that override, disable it.

- Deny Connection Inline—(inline mode only) Does not transmit this packet and future packets on the TCP Flow.
- Deny Attacker Victim Pair Inline—(inline mode only) Does not transmit this packet and future packets on the attacker/victim address pair for a specified period of time.
- Deny Attacker Service Pair Inline—(inline mode only) Does not transmit this packet and future packets on the attacker address victim port pair for a specified period of time.
- Deny Attacker Inline—(inline mode only) Does not transmit this packet and future packets from the attacker address for a specified period of time.



**Note** This is the most severe of the deny actions. It denies the current and future packets from a single attacker address. Each deny address times out for *X* seconds from the first event that caused the deny to start, where *X* is the amount of seconds that you configured. You can clear all denied attacker entries by choosing **Monitoring > Properties > Denied Attackers > Clear List**, which permits the addresses back on the network.

- Modify Packet Inline—(inline mode only) Modifies packet data to remove ambiguity about what the end point might do with the packet.

**Note**

---

Modify Packet Inline is part of the Normalizer Engine. It scrubs the packet and corrects irregular issues such as bad checksum, out of range values, and other RFC violations.

---

- Other Actions
  - Request Block Connection—Requests ARC to block this connection.
  - Request Block Host—Requests ARC to block this attacker host.
  - Request Rate Limit—Requests ARC to perform rate limiting.
  - Reset TCP Connection—Sends TCP resets to hijack and terminate the TCP flow.

## AIC Engine

The Application Inspection and Control (AIC) engine inspects HTTP web traffic and enforces FTP commands. This section describes the AIC engine and its parameters, and contains the following topics:

- [Understanding the AIC Engine, page B-7](#)
- [AIC Engine and Sensor Performance, page B-7](#)
- [AIC Engine Parameters, page B-8](#)

## Understanding the AIC Engine

AIC provides thorough analysis of web traffic. It provides granular control over HTTP sessions to prevent abuse of the HTTP protocol. It allows administrative control over applications, such as instant messaging and gotomypc, that try to tunnel over specified ports. Inspection and policy checks for P2P and instant messaging are 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.

**Note**

---

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.

---

## AIC Engine and Sensor Performance

Application policy enforcement is a unique sensor feature. Rather than being based on traditional IPS technologies that inspect for exploits, vulnerabilities, and anomalies, AIC policy enforcement is designed to enforce HTTP and FTP service policies. The inspection work required for this policy enforcement is extreme compared with traditional IPS inspection work. A large performance penalty is associated with using this feature. When AIC is enabled, the overall bandwidth capacity of the sensor is reduced.

AIC policy enforcement is disabled in the IPS default configuration. If you want to activate AIC policy enforcement, we highly recommend that you carefully choose the exact policies of interest and disable those you do not need. Also, if your sensor is near its maximum inspection load capacity, we recommend that you not use this feature since it can oversubscribe the sensor. We recommend that you use the adaptive security appliance firewall to handle this type of policy enforcement.

## AIC Engine Parameters

The AIC engine defines signatures for deep inspection of web traffic. It also defines signatures that authorize and enforce FTP commands.

There are two AIC engines: AIC HTTP and AIC FTP.

The AIC engine has the following features:

- Web traffic:
  - RFC compliance enforcement
  - HTTP request method authorization and enforcement
  - Response message validation
  - MIME type enforcement
  - Transfer encoding type validation
  - Content control based on message content and type of data being transferred
  - URI length enforcement
  - Message size enforcement according to policy configured and the header
  - Tunneling, P2P and instant messaging enforcement.

This enforcement is done using regular expressions. There are predefined signature but you can expand the list.

- FTP traffic:
  - FTP command authorization and enforcement

[Table B-3](#) lists the parameters that are specific to the AIC HTTP engine.

**Table B-3** AIC HTTP Engine Parameters

| Parameter                 | Description  |
|---------------------------|--|
| Signature Type            | Specifies the type of AIC signature.   |
| Content Types             | AIC signature that deals with MIME types: <ul style="list-style-type: none"> <li>• Define Content Type—Associates actions such as denying a specific MIME type (image/gif), defining a message-size violation, and determining that the MIME-type mentioned in the header and body do not match.</li> <li>• Define Recognized Content Types—Lists content types recognized by the sensor.</li> </ul> |
| Define Web Traffic Policy | Specifies the action to take when noncompliant HTTP traffic is seen. Alarm on Non-HTTP Traffic Yes   No enables the signature. This signature is disabled by default.  |

**Table B-3** AIC HTTP Engine Parameters (continued)

| Parameter                        | Description   |
|----------------------------------|---|
| Max Outstanding Requests Overrun | Maximum allowed HTTP requests per connection (1 to 16).   |
| Msg Body Pattern                 | Uses Regex to define signatures that look for specific patterns in the message body.  |
| Request Methods                  | AIC signature that allows actions to be associated with HTTP request methods: <ul style="list-style-type: none"> <li>Define Request Method—get, put, and so forth.</li> <li>Recognized Request Methods—Lists methods recognized by the sensor.</li> </ul>   |
| Transfer Encoding                | AIC signature that deals with transfer encodings: <ul style="list-style-type: none"> <li>Define Transfer Encoding—Associates an action with each method, such as compress, chunked, and so forth.</li> <li>Recognized Transfer Encodings—Lists methods recognized by the sensor.</li> <li>Chunked Transfer Encoding—Error specifies actions to be taken when a chunked encoding error is seen.</li> </ul> |

Table B-4 lists the parameters that are specific to the AIC FTP engine.

**Table B-4** AIC FTP Engine Parameters

| Parameter                | Description  |
|--------------------------|--|
| Signature Type           | Specifies the type of AIC signature.   |
| FTP Commands             | Associates an action with an FTP command: <ul style="list-style-type: none"> <li>FTP Command—Lets you choose the FTP command you want to inspect.</li> </ul> |
| Unrecognized FTP Command | Inspects unrecognized FTP commands.  |

**For More Information**

- For the procedures for configuring AIC engine signatures, see [Configuring Application Policy, page 5-37](#).
- For an example of a custom AIC signature, see [Tuning an AIC Signature, page 5-38](#).

## Atomic Engine

The Atomic engine contains signatures for simple, single packet conditions that cause alerts to be fired. This section describes the Atomic engine, and contains the following topics:

- [Atomic ARP Engine, page B-10](#)
- [Atomic IP Engine, page B-10](#)
- [Atomic IPv6 Engine, page B-11](#)

## Atomic ARP Engine

The Atomic ARP engine defines basic Layer 2 ARP signatures and provides more advanced detection of the ARP spoof tools dsniff and ettercap.

Table B-5 lists the parameters that are specific to the Atomic ARP engine.

**Table B-5 Atomic ARP Engine Parameters**

| Parameter                 | Description  |
|---------------------------|--|
| Specify Mac Flip Times    | Fires an alert when the MAC address changes more than this many times for this IP address.   |
| Specify Type of Arp Sig   | Specifies the type of ARP signatures you want to fire on: <ul style="list-style-type: none"> <li>• Source Broadcast (default)—Fires an alarm for this signature when it sees an ARP source address of 255.255.255.255.</li> <li>• Destination Broadcast—Fires an alarm for this signature when it sees an ARP destination address of 255.255.255.255.</li> <li>• Same Source and Destination—Fires an alarm for this signature when it sees an ARP destination address with the same source and destination MAC address</li> <li>• Source Multicast—Fires an alarm for this signature when it sees an ARP source MAC address of 01:00:5e:(00-7f).</li> </ul> |
| Specify Request Inbalance | Fires an alert when there are this many more requests than replies on the IP address.  |
| Specify ARP Operation     | The ARP operation code for this signature.   |

## Atomic IP Engine

The Atomic IP engine defines signatures that inspect IP protocol headers and associated Layer 4 transport protocols (TCP, UDP, and ICMP) and payloads.



### Note

The Atomic engines do not store persistent data across packets. Instead they can fire an alert from the analysis of a single packet.

Table B-6 lists the parameters that are specific to the Atomic IP engine.

**Table B-6 Atomic IP Engine Parameters**

| Parameter                  | Description                                    |
|----------------------------|--|
| Fragment Status            | Specifies whether or not fragments are wanted. |
| Specify Layer 4 Protocol   | Specifies Layer 4 protocol.                    |
| Specify IP Payload Length  | Specifies IP datagram payload length.          |
| Specify IP Header Length   | Specifies IP datagram header length.           |
| Specify IP Type of Service | Specifies type of service.                     |
| Specify IP Time-to-Live    | Specifies time to live.                        |

**Table B-6 Atomic IP Engine Parameters (continued)**

| Parameter                    | Description   |
|------------------------------|---|
| Specify IP Version           | Specifies IP protocol version.  |
| Specify IP Identifier        | Specifies IP identifier.  |
| Specify IP Total Length      | Specifies IP datagram total length.   |
| Specify IP Option Inspection | Specifies IP options inspection.  |
| Specify IP Addr Options      | Specifies IP addresses.   |
| Swap Attacker Victim         | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default). |

## Atomic IPv6 Engine

The Atomic IPv6 engine detects two IOS vulnerabilities that are stimulated by malformed IPv6 traffic. These vulnerabilities can lead to router crashes and other security issues. One IOS vulnerability deals with multiple first fragments, which cause a buffer overflow. The other one deals with malformed ICMPv6 Neighborhood Discovery options, which also cause a buffer overflow.


**Note**

IPv6 increases the IP address size from 32 bits to 128 bits, which supports more levels of addressing hierarchy, a much greater number of addressable nodes, and autoconfiguration of addresses.

There are eight Atomic IPv6 signatures. The Atomic IPv6 inspects Neighborhood Discovery protocol of the following types:

- Type 133—Router Solicitation
- Type 134—Router Advertisement
- Type 135—Neighbor Solicitation
- Type 136—Neighbor Advertisement
- Type 137—Redirect


**Note**

Hosts and routers use Neighborhood Discovery to determine the link-layer addresses for neighbors known to reside on attached links and to quickly purge cached values that become invalid. Hosts also use Neighborhood Discovery to find neighboring routers that will forward packets on their behalf.

Each Neighborhood Discovery type can have one or more Neighborhood Discovery options. The Atomic IPv6 engine inspects the length of each option for compliance with the legal values stated in RFC 2461. Violations of the length of an option results in an alert corresponding to the option type where the malformed length was encountered (signatures 1601 to 1605).


**Note**

The Atomic IPv6 signatures do not have any specific parameters to configure.

Table B-7 lists the Atomic IPv6 signatures.

**Table B-7 Atomic IPv6 Signatures**

| Signature ID | Subsignature ID | Name                                   | Description  |
|--------------|-----------------|--|--|
| 1600         | 0               | ICMPv6 zero length option              | For any option type that has ZERO stated as its length   |
| 1601         | 0               | ICMPv6 option type 1 violation         | Violation of the valid length of 8 or 16 bytes.  |
| 1602         | 0               | ICMPv6 option type 2 violation         | Violation of the valid length of 8 or 16 bytes.  |
| 1603         | 0               | ICMPv6 option type 3 violation         | Violation of the valid length of 32 bytes.   |
| 1604         | 0               | ICMPv6 option type 4 violation         | Violation of the valid length of 80 bytes.   |
| 1605         | 0               | ICMPv6 option type 5 violation         | Violation of the valid length of 8 bytes.  |
| 1606         | 0               | ICMPv6 short option data               | Not enough data signature (when the packet states there is more data for an option than is available in the real packet) |
| 1607         | 0               | IPv6 multiple-crafted fragment packets | Produces an alert when more than one first fragment is seen in a 30-second period.                                       |

## Fixed Engine

This section describes the Fixed engine, and contains the following topics:

- [Understanding the Fixed Engine, page B-12](#)
- [Fixed ICMP Engine Parameters, page B-13](#)
- [Fixed TCP Engine Parameters, page B-14](#)
- [Fixed UDP Engine Parameters, page B-15](#)

## Understanding the Fixed Engine

The Fixed engine combines multiple regular expression patterns in to a single pattern matching table that allows a single search through the data. It supports ICMP, TCP, and UDP protocols. After a minimum inspection depth is reached (1 to 100 bytes), inspection stops. There are three Fixed engines: Fixed ICMP, Fixed TCP, and Fixed UDP.



### Note

Fixed TCP and Fixed UDP use the Service Ports parameter as exclusion ports. Fixed ICMP uses the Service Ports parameter as excluded ICMP types.

## Fixed ICMP Engine Parameters

Table B-8 lists the parameters specific to the Fixed ICMP engine.

**Table B-8** Fixed ICMP Engine Parameters

| Parameter                    | Description   | Value                      |
|------------------------------|---|----------------------------|
| Direction                    | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port</li> <li>Traffic from client port destined to service port</li> </ul>      | From Service<br>To Service |
| Max Payload Inspect Length   | Specifies the maximum inspection depth for the signature.   | 1 to 250                   |
| Regex String                 | Specifies the regular expression to search for in a single packet.  | string                     |
| Specify Exact Match Offset   | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the Regex String must report for a match to be valid.</li> </ul> | 0 to 65535                 |
| Specify Minimum Match Length | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Minimum Match Length—Specifies the minimum number of bytes the Regex String must match.</li> </ul>        | 0 to 65535                 |
| Specify ICMP Type            | (Optional) Enables inspection of L4 ICMP type: <ul style="list-style-type: none"> <li>ICMP Type—Specifies the ICMP TYPE value.</li> </ul>   | 0 to 65535                 |
| Swap Attacker Victim         | Yes if address (and ports) source and destination are swapped in the alert message. No for no swap (default).   | Yes   No                   |

## Fixed TCP Engine Parameters

Table B-9 lists the parameters specific to the Fixed TCP engine.

**Table B-9** Fixed TCP Engine Parameters

| Parameter                    | Description   | Value                                |
|------------------------------|---|--------------------------------------|
| Direction                    | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port</li> <li>Traffic from client port destined to service port</li> </ul>      | From Service<br>To Service           |
| Max Payload Inspect Length   | Specifies the maximum inspection depth for the signature.   | 1 to 250                             |
| Regex String                 | Specifies the regular expression to search for in a single packet.  | string                               |
| Specify Exact Match Offset   | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the Regex String must report for a match to be valid.</li> </ul> | 0 to 65535                           |
| Specify Minimum Match Length | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Minimum Match Length—Specifies the minimum number of bytes the Regex String must match.</li> </ul>        | 0 to 65535                           |
| Specify Service Ports        | Enables service ports for use: <ul style="list-style-type: none"> <li>Service Ports—A comma-separated list of ports or port ranges where the target service resides.</li> </ul>           | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Swap Attacker Victim         | Yes if address (and ports) source and destination are swapped in the alert message. No for no swap (default).   | Yes   No                             |

1. The second number in the range must be greater than or equal to the first number.

## Fixed UDP Engine Parameters

Table B-10 lists the parameters specific to the Fixed UDP engine.

**Table B-10** Fixed UDP Engine Parameters

| Parameter                    | Description   | Value                                |
|------------------------------|---|--------------------------------------|
| Direction                    | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port</li> <li>Traffic from client port destined to service port</li> </ul>      | From Service<br>To Service           |
| Max Payload Inspect Length   | Specifies the maximum inspection depth for the signature.   | 1 to 250                             |
| Regex String                 | Specifies the regular expression to search for in a single packet.  | string                               |
| Specify Exact Match Offset   | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the Regex String must report for a match to be valid.</li> </ul> | 0 to 65535                           |
| Specify Minimum Match Length | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Minimum Match Length—Specifies the minimum number of bytes the Regex String must match.</li> </ul>        | 0 to 65535                           |
| Specify Service Ports        | Enables service ports for use: <ul style="list-style-type: none"> <li>Service Ports—A comma-separated list of ports or port ranges where the target service resides.</li> </ul>           | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Swap Attacker Victim         | Yes if address (and ports) source and destination are swapped in the alert message. No for no swap (default).   | Yes   No                             |

1. The second number in the range must be greater than or equal to the first number.

## Flood Engine

The Flood engine defines signatures that watch for any host or network sending multiple packets to a single host or network. For example, you can create a signature that fires when 150 or more packets per second (of the specific type) are found going to the victim host. There are two types of Flood engines: Flood Host and Flood Net.

Table B-11 lists the parameters specific to the Flood Host engine.

**Table B-11 Flood Host Engine Parameters**

| Parameter | Description   | Value                                |
|-----------|---|--------------------------------------|
| Protocol  | Which kind of traffic to inspect.                             | ICMP<br>UDP                          |
| Rate      | Threshold number of packets per second.                       | 0 to 65535 <sup>1</sup>              |
| ICMP Type | Specifies the value for the ICMP header type.                 | 0 to 65535                           |
| Dst Ports | Specifies the destination ports when you choose UDP protocol. | 0 to 65535 <sup>2</sup><br>a-b[,c-d] |
| Src Ports | Specifies the source ports when you choose UDP protocol.      | 0 to 65535 <sup>3</sup><br>a-b[,c-d] |

1. An alert fires when the rate is greater than the packets per second.
2. The second number in the range must be greater than or equal to the first number.
3. The second number in the range must be greater than or equal to the first number.

Table B-12 lists the parameters specific to the Flood Net engine.

**Table B-12 Flood Net Engine Parameters**

| Parameter         | Description   | Value                   |
|-------------------|---|-------------------------|
| Gap               | Gap of time allowed (in seconds) for a flood signature. | 0 to 65535              |
| Peaks             | Number of allowed peaks of flood traffic.               | 0 to 65535              |
| Protocol          | Which kind of traffic to inspect.                       | ICMP<br>TCP<br>UDP      |
| Rate              | Threshold number of packets per second.                 | 0 to 65535 <sup>1</sup> |
| Sampling Interval | Interval used for sampling traffic.                     | 1 to 3600               |
| ICMP Type         | Specifies the value for the ICMP header type.           | 0 to 65535              |

1. An alert fires when the rate is greater than the packets per second.

## Meta Engine

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 the Signature Event Action Processor. The Signature Event Action Processor 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.



### Caution

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

Table B-13 lists the parameters specific to the Meta engine.

**Table B-13 Meta Engine Parameters**

| Parameter               | Description   | Value                        |
|-------------------------|---|------------------------------|
| Swap Attacker Victim    | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No                     |
| Meta Reset Interval     | Time in seconds to reset the Meta signature.  | 0 to 3600                    |
| Component List          | List of Meta components: <ul style="list-style-type: none"> <li>• edit—Edits an existing entry</li> <li>• insert—Inserts a new entry into the list: <ul style="list-style-type: none"> <li>– begin—Places the entry at the beginning of the active list</li> <li>– end—Places the entry at the end of the active list</li> <li>– inactive—Places the entry into the inactive list</li> <li>– before—Places the entry before the specified entry</li> <li>– after—Places the entry after the specified entry</li> </ul> </li> <li>• move—Moves an entry in the list</li> </ul> | <i>name1</i>                 |
| Meta Key                | Storage type for the Meta signature: <ul style="list-style-type: none"> <li>• Attacker address</li> <li>• Attacker and victim addresses</li> <li>• Attacker and victim addresses and ports</li> <li>• Victim address</li> </ul>   | AaBb<br>AxBx<br>Axxx<br>xxBx |
| Unique Victims          | Number of unique victims ports required per Meta signature.   | 1 to 256                     |
| Component List In Order | Whether to fire the component list in order.  | Yes   No                     |

#### For More Information

For an example of a custom Meta engine signature, see [Example Meta Engine Signature, page 5-22](#).

## Multi String Engine

The Multi String engine lets you define signatures that inspect Layer 4 transport protocol (ICMP, TCP, and UDP) payloads using multiple string matches for one signature. You can specify a series of regular expression patterns that must be matched to fire the signature. For example, you can define a signature that looks for regex 1 followed by regex 2 on a UDP service. For UDP and TCP you can specify port numbers and direction. You can specify a single source port, a single destination port, or both ports. The string matching takes place in both directions.

Use the Multi String engine when you need to specify more than one regex pattern. Otherwise, you can use the String ICMP, String TCP, or String UDP engine to specify a single Regex pattern for one of those protocols.

Table B-14 lists the parameters specific to the Multi String Engine.

**Table B-14 Multi String Engine Parameters**

| Parameter            | Description  | Value                                    |
|----------------------|--|--|
| Inspect Length       | Length of stream or packet that must contain all offending strings for the signature to fire.  | 0 to 4294967295                          |
| Protocol             | Layer 4 protocol selection.  | ICMP<br>TCP<br>UDP                       |
| Regex Component      | List of regex components: <ul style="list-style-type: none"> <li>Regex String—The string to search for.</li> <li>Spacing Type—Type of spacing required from the match before or from the beginning of the stream/packet if it is the first entry in the list.</li> </ul> | list (1 to 16 items)<br>exact<br>minimum |
| Port Selection       | Type of TCP or UDP port to inspect: <ul style="list-style-type: none"> <li>Both Ports—Specifies both source and destination port.</li> <li>Destination—Specifies a range of destination ports.</li> <li>Source—Specifies a range of source ports.<sup>1</sup></li> </ul> | 0 to 65535 <sup>2</sup>                  |
| Extra Spacing        | Exact number of bytes that must be between this regex string and the one before, or from the beginning of the stream/packet if it is the first entry in the list.  | 0 to 4294967296                          |
| Minimum Spacing      | Minimum number of bytes that must be between this regex string and the one before, or from the beginning of the stream/packet if it is the first entry in the list.  | 0 to 4294967296                          |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).  | Yes   No                                 |

- Port matching is performed bidirectionally for both the client-to-server and server-to-client traffic flow directions. For example, if the source-ports value is 80, in a client-to-server traffic flow direction, inspection occurs if the client port is 80. In a server-to-client traffic flow direction, inspection occurs if the server port is port 80.
- A valid value is a comma-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.



**Caution**

The Multi String engine can have a significant impact on memory usage.

# Normalizer Engine

The Normalizer engine deals with IP fragmentation and TCP normalization. This section describes the Normalizer engine, and contains the following topics:

- [Understanding the Normalizer Engine, page B-19](#)
- [Normalizer Engine Parameters, page B-20](#)

## Understanding the Normalizer Engine

**Note**

---

You cannot add custom signatures to the Normalizer engine. You can tune the existing ones.

---

The Normalizer engine deals with IP fragment reassembly and TCP stream reassembly. With the Normalizer engine you can set limits on system resource usage, for example, the maximum number of fragments the sensor tries to track at the same time. Sensors in promiscuous mode report alerts on violations. Sensors in inline mode perform the action specified in the event action parameter, such as produce alert, deny packet inline, and modify packet inline.

**Caution**

---

For signature 3050 Half Open SYN Attack, if you choose modify packet inline as the action, you can see as much as 20 to 30% performance degradation while the protection is active. The protection is only active during an actual SYN flood.

---

### IP Fragmentation Normalization

Intentional or unintentional fragmentation of IP datagrams can hide exploits making them difficult or impossible to detect. Fragmentation can also be used to circumvent access control policies like those found on firewalls and routers. And different operating systems use different methods to queue and dispatch fragmented datagrams. If the sensor has to check for all possible ways that the end host can reassemble the datagrams, the sensor becomes vulnerable to DoS attacks. Reassembling all fragmented datagrams inline and only forwarding completed datagrams, refragmenting the datagram if necessary, prevents this. The IP Fragmentation Normalization unit performs this function.

### TCP Normalization

Through intentional or natural TCP session segmentation, some classes of attacks can be hidden. To make sure policy enforcement can occur with no false positives and false negatives, the state of the two TCP endpoints must be tracked and only the data that is actually processed by the real host endpoints should be passed on. Overlaps in a TCP stream can occur, but are extremely rare except for TCP segment retransmits. Overwrites in the TCP session should not occur. If overwrites do occur, someone is intentionally trying to elude the security policy or the TCP stack implementation is broken. Maintaining full information about the state of both endpoints is not possible unless the sensor acts as a TCP proxy. Instead of the sensor acting as a TCP proxy, the segments are ordered properly and the normalizer looks for any abnormal packets associated with evasion and attacks.

### For More Information

For the procedures for configuring signatures in the Normalizer engine, see [Configuring IP Fragment Reassembly Signatures, page 5-38](#), and [Configuring TCP Stream Reassembly Signatures, page 5-42](#).

## Normalizer Engine Parameters

Table B-15 lists the parameters that are specific to the Normalizer engine.

**Table B-15** Normalizer Engine Parameters

| Parameter                           | Description  |
|-------------------------------------|--|
| Edit Defaults                       | Editable signatures.                               |
| Specify Fragment Reassembly Timeout | (Optional) Enables fragment reassembly timeout.    |
| Specify Hijack Max Old Ack          | (Optional) Enables hijack-max-old-ack.             |
| Specify Max Datagram Size           | (Optional) Enables maximum datagram size.          |
| Specify Max Fragments               | (Optional) Enables maximum fragments.              |
| Specify Max Fragments per Datagram  | (Optional) Enables maximum fragments per datagram. |
| Specify Max Last Fragments          | (Optional) Enables maximum last fragments.         |
| Specify Max Partial Datagrams       | (Optional) Enables maximum partial datagrams.      |
| Specify Max Small Frags             | (Optional) Enables maximum small fragments.        |
| Specify Min Fragment Size           | (Optional) Enables minimum fragment size.          |
| Specify Service Ports               | (Optional) Enables service ports.                  |
| Specify SYN Flood Max Embryonic     | (Optional) Enables SYN flood maximum embryonic.    |
| Specify TCP Closed Timeout          | (Optional) Enables TCP closed timeout.             |
| Specify TCP Embryonic Timeout       | (Optional) Enables TCP embryonic timeout.          |
| Specify TCP Idle Timeout            | (Optional) Enables TCP idle timeout.               |
| Specify TCP Max MSS                 | (Optional) Enables TCP maximum mss.                |
| Specify TCP Max Queue               | (Optional) Enables TCP maximum queue.              |
| Specify TCP Min MSS                 | (Optional) Enables TCP minimum mss.                |
| Specify TCP Option Number           | (Optional) Enables TCP option number.              |

## Service Engines

The Service engines analyze Layer 5+ traffic between two hosts. These are one-to-one signatures that track persistent data. The engines analyze the Layer 5+ payload in a manner similar to the live service.

The Service engines have common characteristics but each engine has specific knowledge of the service that it is inspecting. The Service engines supplement the capabilities of the generic string engine specializing in algorithms where using the string engine is inadequate or undesirable.

This section describes the Service engines, and contains the following topics:

- [Service DNS Engine, page B-21](#)
- [Service FTP Engine, page B-22](#)
- [Service Generic Engine, page B-23](#)
- [Service H225 Engine, page B-24](#)
- [Service HTTP Engine, page B-27](#)

- Service IDENT Engine, page B-29
- Service MSRPC Engine, page B-29
- Service MSSQL Engine, page B-30
- Service NTP Engine, page B-31
- Service P2P Engine, page B-31
- Service RPC Engine, page B-32
- Service SMB Advanced Engine, page B-33
- Service SNMP Engine, page B-35
- Service SSH Engine, page B-36
- Service TNS Engine, page B-36

## Service DNS Engine

The Service DNS engine specializes in advanced DNS decode, which includes anti-evasive techniques, such as following multiple jumps. It has many parameters such as lengths, opcodes, strings, and so forth. The Service DNS engine is a biprotocol inspector operating on both TCP and UDP port 53. It uses the stream for TCP and the quad for UDP.

Table B-16 lists the parameters specific to the Service DNS engine.

**Table B-16** Service DNS Engine Parameters

| Parameter                         | Description   | Value                     |
|-----------------------------------|---|---------------------------|
| Protocol                          | Protocol of interest for this inspector.  | TCP<br>UDP                |
| Specify query Chaos String        | (Optional) Enables the DNS Query Class Chaos String.  | <i>query-chaos-string</i> |
| Specify Query Class               | (Optional) Enables the query class: <ul style="list-style-type: none"> <li>• Query Class—DNS Query Class 2 Byte Value</li> </ul>                              | 0 to 65535                |
| Specify query Invalid Domain Name | (Optional) Enables query invalid domain name: <ul style="list-style-type: none"> <li>• Query Invalid Domain Name—DNS Query Length greater than 255</li> </ul> | Yes   No                  |
| Specify Query Jump Count Exceeded | (Optional) Enables query jump count exceeded: <ul style="list-style-type: none"> <li>• Query Jump Count Exceeded—DNS compression counter</li> </ul>           | Yes   No                  |
| Specify Query Opcode              | (Optional) Enables query opcode: <ul style="list-style-type: none"> <li>• Query Opcode—DNS Query Opcode 1 byte Value</li> </ul>                               | 0 to 65535                |

Table B-16 Service DNS Engine Parameters (continued)

| Parameter                         | Description   | Value      |
|-----------------------------------|---|------------|
| Specify Query Record Data Invalid | (Optional) Enables query record data invalid: <ul style="list-style-type: none"> <li>Query Record Data Invalid—DNS Record Data incomplete</li> </ul>        | Yes   No   |
| Specify Query Record Data Length  | (Optional) Enables the query record data length: <ul style="list-style-type: none"> <li>Query Record Data Length—DNS Response Record Data Length</li> </ul> | 0 to 65535 |
| Specify Query Src Port 53         | (Optional) Enables the query source port 53: <ul style="list-style-type: none"> <li>Query Src Port 53—DNS packet source port 53</li> </ul>                  | Yes   No   |
| Specify Query Stream Length       | (Optional) Enables the query stream length: <ul style="list-style-type: none"> <li>Query Record Data Length—DNS Packet Length</li> </ul>                    | 0 to 65535 |
| Specify Query Type                | (Optional) Enables the query type: <ul style="list-style-type: none"> <li>Query Type—DNS Query Type 2 Byte Value</li> </ul>                                 | 0 to 65535 |
| Specify Query Value               | (Optional) Enables the query value: <ul style="list-style-type: none"> <li>Query Value—Query 0 Response 1</li> </ul>  | Yes   No   |

## Service FTP Engine

The Service FTP engine specializes in FTP port command decode, trapping invalid **port** commands and the PASV port spoof. It fills in the gaps when the String engine is not appropriate for detection. The parameters are Boolean and map to the various error trap conditions in the **port** command decode. The Service FTP engine runs on TCP ports 20 and 21. Port 20 is for data and the Service FTP engine does not do any inspection on this. It inspects the control transactions on port 21.

Table B-17 lists the parameters that are specific to the Service FTP engine.

**Table B-17 Service FTP Engine Parameters**

| Parameter               | Description   | Value  |
|-------------------------|---|--|
| Direction               | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port</li> <li>Traffic from client port destined to service port</li> </ul>  | From Service<br>To Service   |
| FTP Inspection Type     | Type of inspection to perform: <ul style="list-style-type: none"> <li>Looks for an invalid address in the FTP port command</li> <li>Looks for an invalid port in the FTP port command</li> <li>Looks for the PASV port spoof</li> </ul> | Invalid Address in<br>PORT Command<br>Invalid Port in PORT<br>Command<br>PASV Port Spoof |
| Service Ports           | A comma-separated list of ports or port ranges where the target service resides.  | 0 to 65535 <sup>1</sup><br>a-b[,c-d]   |
| Swap Attacker<br>Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No   |

1. The second number in the range must be greater than or equal to the first number.

## Service Generic Engine

The Service Generic engine allows programmatic signatures to be issued in a config-file-only signature update. It has a simple machine and assembly language that is defined in the configuration file. It runs the machine code (distilled from the assembly language) through its virtual machine, which processes the instructions and pulls the important pieces of information out of the packet and runs them through the comparisons and operations specified in the machine code. It is intended as a rapid signature response engine to supplement the String and State engines.

New functionality adds the Regex parameter to the Service Generic engine and enhanced instructions. The Service Generic engine can analyze traffic based on the mini-programs that are written to parse the packets. These mini-programs are composed of commands, which dissect the packet and look for certain conditions.



### Note

You cannot use the Service Generic engine to create custom signatures.



### Caution

Due to the proprietary nature of this complex language, we do not recommend that you edit the Service Generic engine signature parameters other than severity and event action.

Table B-18 lists the parameters specific to the Service Generic engine.

**Table B-18 Service Generic Engine Parameters**

| Parameter              | Description   | Value  |
|------------------------|---|--|
| Specify Dst Port       | (Optional) Enables the destination port: <ul style="list-style-type: none"> <li>Dst Port—Destination port of interest for this signature</li> </ul>   | 0 to 65535   |
| Specify IP Protocol    | (Optional) Enables IP protocol: <ul style="list-style-type: none"> <li>IP Protocol—The IP protocol this inspector should examine</li> </ul>   | 0 to 255   |
| Specify Payload Source | (Optional) Enables payload source inspection: <ul style="list-style-type: none"> <li>Payload Source—Payload source inspection for the following types:               <ul style="list-style-type: none"> <li>Inspects ICMP data</li> <li>Inspects Layer 2 headers</li> <li>Inspects Layer 3 headers</li> <li>Inspects Layer 4 headers</li> <li>Inspects TCP data</li> <li>Inspects UDP data</li> </ul> </li> </ul> | ICMP Data<br>12 Header<br>13 Header<br>14 Header<br>TCP Data<br>UDP Data |
| Specify Src Port       | (Optional) Enables the source port: <ul style="list-style-type: none"> <li>Src Port—Source port of interest for this signature</li> </ul>   | 0 to 65535   |
| Specify Regex String   | The regular expression to look for when the policy type is regex: <ul style="list-style-type: none"> <li>A regular expression to search for in a single TCP packet</li> <li>(Optional) Enables min match length for use. The minimum length of the Regex match required to constitute a match.</li> </ul>   | Regex String<br>Specify Min Match Length                                 |
| Swap Attacker Victim   | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No   |

## Service H225 Engine

The Service H225 engine analyzes H225.0 protocol, which consists of many subprotocols and is part of the H.323 suite. H.323 is a collection of protocols and other standards that together enable conferencing over packet-based networks.

H.225.0 call signaling and status messages are part of the H.323 call setup. Various H.323 entities in a network, such as the gatekeeper and endpoint terminals, run implementations of the H.225.0 protocol stack. The Service H225 engine analyzes H225.0 protocol for attacks on multiple H.323 gatekeepers, VoIP gateways, and endpoint terminals. It provides deep packet inspection for call signaling messages that are exchanged over TCP PDUs. The Service H225 engine analyzes the H.225.0 protocol for invalid H.225.0 messages, and misuse and overflow attacks on various protocol fields in these messages.

H.225.0 call signaling messages are based on Q.931 protocol. The calling endpoint sends a Q.931 setup message to the endpoint that it wants to call, the address of which it procures from the admissions procedure or some lookup means. The called endpoint either accepts the connection by transmitting a

Q.931 connect message or rejects the connection. When the H.225.0 connection is established, either the caller or the called endpoint provides an H.245 address, which is used to establish the control protocol (H.245) channel.

Especially important is the SETUP call signaling message because this is the first message exchanged between H.323 entities as part of the call setup. The SETUP message uses many of the commonly found fields in the call signaling messages, and implementations that are exposed to probable attacks will mostly also fail the security checks for the SETUP messages. Therefore, it is highly important to check the H.225.0 SETUP message for validity and enforce checks on the perimeter of the network.

The Service H225 engine has built-in signatures for TPKT validation, Q.931 protocol validation, and ASN.1PER validations for the H225 SETUP message. ASN.1 is a notation for describing data structures. PER uses a different style of encoding. It specializes the encoding based on the data type to generate much more compact representations.

You can tune the Q.931 and TPKT length signatures and you can add and apply granular signatures on specific H.225 protocol fields and apply multiple pattern search signatures of a single field in Q.931 or H.225 protocol.

The Service H225 engine supports the following features:

- TPKT validation and length check
- Q.931 information element validation
- Regular expression signatures on text fields in Q.931 information elements
- Length checking on Q.931 information elements
- SETUP message validation
- ASN.1 PER encode error checks
- Configuration signatures for fields like ULR-ID, E-mail-ID, h323-id, and so forth for both regular expression and length.

There is a fixed number of TPKT and ASN.1 signatures. You cannot create custom signatures for these types. For TPKT signatures, you should only change the value-range for length signatures. You should not change any parameters for ASN.1. For Q.931 signatures, you can add new regular expression signatures for text fields. For SETUP signatures, you can add signatures for length and regular expression checks on various SETUP message fields.

Table B-19 lists parameters specific to the Service H225 engine.

**Table B-19 Service H.225 Engine Parameters**

| Parameter                    | Description  | Value  |
|------------------------------|--|--|
| Message Type                 | Type of H225 message to which the signature applies: <ul style="list-style-type: none"> <li>• SETUP</li> <li>• ASN.1-PER</li> <li>• Q.931</li> <li>• TPKT</li> </ul>   | asn.1-per<br>q.931<br>setup<br>tpkt              |
| Policy Type                  | Type of H225 policy to which the signature applies: <ul style="list-style-type: none"> <li>• Inspects field length.</li> <li>• Inspects presence. If certain fields are present in the message, an alert is sent.</li> <li>• Inspects regular expressions.</li> <li>• Inspects field validations.</li> <li>• Inspects values.</li> </ul> Regex and presence are not valid for TPKT signatures. | length<br>presence<br>regex<br>validate<br>value |
| Specify Field Name           | (Optional) Enables field name for use. Only valid for SETUP and Q.931 message types. Gives a dotted representation of the field name that this signature applies to. <ul style="list-style-type: none"> <li>• Field Name—Field name to inspect.</li> </ul>   | 1 to 512   |
| Specify Invalid Packet Index | (Optional) Enables invalid packet index for use for specific errors in ASN, TPKT, and other errors that have fixed mapping. <ul style="list-style-type: none"> <li>• Invalid Packet Index—Inspection for invalid packet index.</li> </ul>  | 0 to 255   |
| Value Range Regex String     | The regular expression to look for when the policy type is regex. This is never set for TPKT signatures: <ul style="list-style-type: none"> <li>• A regular expression to search for in a single TCP packet</li> <li>• (Optional) Enables min match length for use. The minimum length of the Regex match required to constitute a match. This is never set for TPKT signatures.</li> </ul>    | Regex String<br>Specify Min Match Length         |

**Table B-19** Service H.225 Engine Parameters (continued)

| Parameter            | Description   | Value                          |
|----------------------|---|--------------------------------|
| Specify Value Range  | Valid for the length or value policy types (0x00 to 6535). Not valid for other policy types. <ul style="list-style-type: none"> <li>Value Range—Range of values.</li> </ul> | 0 to 65535 <sup>1</sup><br>a-b |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No                       |

1. The second number in the range must be greater than or equal to the first number.

## Service HTTP Engine

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 networks of today. In addition, it requires the most amount of preprocessing time and has the most number of signatures requiring inspection making it critical to the overall performance of the system.

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.

Table B-20 lists the parameters specific the Service HTTP engine.

**Table B-20** Service HTTP Engine Parameters

| Parameter                       | Description   | Value      |
|---------------------------------|---|------------|
| De Obfuscate                    | Applies anti-evasive deobfuscation before searching.  | Yes   No   |
| Max Field Sizes                 | Maximum field sizes grouping.   | —          |
| Specify Max Arg Field Length    | (Optional) Enables maximum argument field length: <ul style="list-style-type: none"> <li>Max Arg Field Length—Maximum length of the arguments field.</li> </ul> | 0 to 65535 |
| Specify Max Header Field Length | (Optional) Enables maximum header field length: <ul style="list-style-type: none"> <li>Max Header Field Length—Maximum length of the header field.</li> </ul>   | 0 to 65535 |



## Service IDENT Engine

The Service IDENT engine inspects TCP port 113 traffic. It has basic decode and provides parameters to specify length overflows.

For example, when a user or program at computer A makes an ident request of computer B, it may only ask for the identity of users of connections between A and B. The ident server on B listens for connections on TCP port 113. The client at A establishes a connection, then specifies which connection it wants identification for by sending the numbers of the ports on A and B that the connection is using. The server at B determines what user is using that connection, and replies to A with a string that names that user. The Service IDENT engine inspects the TCP port 113 for ident abuse.

Table B-21 lists the parameters specific to the Service IDENT engine.

**Table B-21** Service IDENT Engine Parameters

| Parameter       | Description   | Value                                |
|-----------------|---|--------------------------------------|
| Inspection Type | Type of inspection to perform: <ul style="list-style-type: none"> <li>• Has Newline—Inspects payload for a nonterminating new line character.</li> <li>• Has Bad Port—Inspects payload for a bad port.</li> <li>• Payload Size—Inspects for payload length longer than this.</li> </ul> | —                                    |
| Service Ports   | A comma-separated list of ports or port ranges where the target service resides.  | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Direction       | Direction of the traffic: <ul style="list-style-type: none"> <li>• Traffic from service port destined to client port.</li> <li>• Traffic from client port destined to service port.</li> </ul>  | From Service<br>To Service           |

1. The second number in the range must be greater than or equal to the first number.

## Service MSRPC Engine

The Service MSRPC engine processes MSRPC packets. MSRPC allows for cooperative processing between multiple computers and their application software in a networked environment. It is a transaction-based protocol, implying that there is a sequence of communications that establish the channel and pass processing requests and replies.

MSRPC is an ISO Layer 5-6 protocol and is layered on top of other transport protocols such as UDP, TCP, and SMB. The MSRPC engine contains facilities to allow for fragmentation and reassembly of the MSRPC PDUs.

This communication channel is the source of recent Windows NT, Windows 2000, and Window XP security vulnerabilities.

The Service MSRPC engine only decodes the DCE and RPC protocol for the most common transaction types.

Table B-22 lists the parameters specific to the Service MSRPC engine.

**Table B-22 Service MSRPC Engine Parameters**

| Parameter            | Description   | Value  |
|----------------------|---|--|
| Protocol             | Protocol of interest for this inspector: <ul style="list-style-type: none"> <li>Type—UDP or TCP</li> </ul>  | TCP<br>UDP   |
| Specify Flags        | Flags to set: <ul style="list-style-type: none"> <li>MSRPC TCP Flags</li> <li>MSRPC TCP Flags Mask</li> </ul>   | Concurrent Execution<br>Did Not Execute<br>First Fragment<br>Last Fragment<br>Maybe Semantics<br>Object UUID<br>Pending Cancel<br>Reserved |
| Specify Operation    | (Optional) Enables using MSRPC operation: <ul style="list-style-type: none"> <li>Operation—MSRPC operation requested. Required for SMB_COM_TRANSACTION commands. Exact match.</li> </ul>  | 0 to 65535   |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No   |
| Specify Regex String | (Optional) Enables using a regular expression string: <ul style="list-style-type: none"> <li>Specify Exact Match Offset—Enables the exact match offset:               <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> </li> <li>Specify Min Match Length—Enables the minimum match length:               <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul> </li> </ul> | 0 to 65535   |
| Specify UUID         | (Optional) Enables UUID: <ul style="list-style-type: none"> <li>UUID—MSRPC UUID field</li> </ul>  | 000001a000000000c0000<br>00000000046   |

## Service MSSQL Engine

The Service MSSQL engine inspects the protocol used by the Microsoft SQL server. There is one MSSQL signature. It fires an alert when it detects an attempt to log in to an MSSQL server with the default sa account. You can add custom signatures based on MSSQL protocol values, such as login username and whether a password was used.

Table B-23 lists the parameters specific to the Service MSSQL engine.

**Table B-23 Service MSSQL Engine Parameters**

| Parameter            | Description   | Value    |
|----------------------|---|----------|
| Password Present     | Whether or not a password was used in an MS SQL login.  | Yes   No |
| Specify SQL Username | (Optional) Enables using an SQL username: <ul style="list-style-type: none"> <li>SQL Username—Username (exact match) of user logging in to MS SQL service.</li> </ul> | sa       |

## Service NTP Engine

The Service NTP engine inspects NTP protocol. There is one NTP signature, the NTP readvar overflow signature, which fires an alert if a readvar command is seen with NTP data that is too large for the NTP service to capture. You can tune this signature and create custom signatures based on NTP protocol values, such as mode and size of control packets.

Table B-24 lists the parameters specific to the Service NTP engine.

**Table B-24 Service NTP Engine Parameters**

| Parameter              | Description   | Value      |
|------------------------|---|------------|
| Inspection Type        | Type of inspection to perform.  |            |
| Inspect NTP Packets    | Inspects NTP packets: <ul style="list-style-type: none"> <li>Control Opcode—Opcode number of an NTP control packet according to RFC1305, Appendix B.</li> <li>Max Control Data Size—Maximum allowed amount of data sent in a control packet.</li> <li>Operation Mode—Mode of operation of the NTP packet per RFC 1305.</li> </ul> | 0 to 65535 |
| IS Invalid Data Packet | Looks for invalid NTP data packets. Checks the structure of the NTP data packet to make sure it is the correct size.  | Yes   No   |
| Is Non NTP Traffic     | Checks for nonNTP packets on an NTP port.   | Yes   No   |

## Service P2P Engine

P2P networks use nodes that can simultaneously function as both client and server for the purpose of file sharing. P2P networks often contain copyrighted material and their use on a corporate network can violate company policy. The Service P2P engine monitors such networks and provides optimized TCP and UDP P2P protocol identification. The Service P2P engine has the following characteristics:

- Listens on all TCP and UDP ports
- Increased performance through the use of hard-coded signatures rather than regular expressions
- Ignores traffic once P2P protocol is identified or after seeing 10 packets without a P2P protocol being identified

Because the P2P signatures are hard coded, the only parameters that you can edit are the Master engine parameters.

## Service RPC Engine

The Service RPC engine specializes in RPC protocol and has full decode as an anti-evasive strategy. It can handle fragmented messages (one message in several packets) and batch messages (several messages in a single packet).

The RPC portmapper operates on port 111. Regular RPC messages can be on any port greater than 550. RPC sweeps are like TCP port sweeps, except that they only count unique ports when a valid RPC message is sent. RPC also runs on UDP.

Table B-25 lists the parameters specific to the Service RPC engine.

**Table B-25** Service RPC Engine Parameters

| Parameter                | Description   | Value                                |
|--------------------------|---|--------------------------------------|
| Direction                | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port.</li> <li>Traffic from client port destined to service port.</li> </ul>  | From Service<br>To Service           |
| Protocol                 | Protocol of interest.   | TCP<br>UDP                           |
| Service Ports            | A comma-separated list of ports or port ranges where the target service resides.  | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Specify Regex String     | (Optional) Enables using a regular expression string: <ul style="list-style-type: none"> <li>Specify Exact Match Offset—Enables the exact match offset:               <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> </li> <li>Specify Min Match Length—Enables the minimum match length:               <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul> </li> </ul> | 0 to 65535                           |
| Specify Spoof Src        | (Optional) Enables the spoof source address: <ul style="list-style-type: none"> <li>Is Spoof Src—Fires an alert when the source address is 127.0.0.1.</li> </ul>  | Yes   No                             |
| Specify Port Map Program | (Optional) Enables the portmapper program: <ul style="list-style-type: none"> <li>Port Map Program—The program number sent to the portmapper for this signature.</li> </ul>   | 0 to 999999999                       |
| Specify RPC Max Length   | (Optional) Enables RPC maximum length: <ul style="list-style-type: none"> <li>RPC Max Length—Maximum allowed length of the entire RPC message. Lengths longer than what you specify fire an alert.</li> </ul>   | 0 to 65535                           |
| Specify RPC Procedure    | (Optional) Enables RPC procedure: <ul style="list-style-type: none"> <li>RPC Procedure—RPC procedure number for this signature.</li> </ul>  | 0 to 1000000                         |

**Table B-25** Service RPC Engine Parameters (continued)

| Parameter            | Description  | Value        |
|----------------------|--|--------------|
| Specify RPC Program  | (Optional) Enables RPC program: <ul style="list-style-type: none"> <li>RPC Program—RPC program number for this signature.</li> </ul> | 0 to 1000000 |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).                    | Yes   No     |

1. The second number in the range must be greater than or equal to the first number.

## Service SMB Advanced Engine



### Caution

The SMB engine has been replaced by the SMB Advanced engine. Even though the SMB engine is still visible in IDM, IME, and the CLI, its signatures have been obsoleted; that is, the new signatures have the obsoletes parameter set with the IDs of their corresponding old signatures. Use the new SMB Advanced engine to rewrite any custom signature that were in the SMB engine.

The Service SMB Advanced engine processes Microsoft SMB and Microsoft RPC over SMB packets. The Service SMB Advanced engine uses the same decoding method for connection-oriented MSRPC as the MSRPC engine with the requirement that the MSRPC packet must be over the SMB protocol. The Service SMB Advanced engine supports MSRPC over SMB on TCP ports 139 and 445. It uses a copy of the connection-oriented DCS/RPC code from the MSRPC engine.

Table B-26 lists the parameters specific to the Service SMB Advanced engine.

**Table B-26** Service SMB Advanced Engine Parameters

| Parameter         | Description  | Value                                |
|-------------------|--|--------------------------------------|
| Service Ports     | A comma-separated list of ports or port ranges where the target service resides.   | 0 to 65535<br>a-b[,c-d] <sup>1</sup> |
| Specify Command   | (Optional) Enables SMB commands: <ul style="list-style-type: none"> <li>Command—SMB command value; exact match required; defines the SMB packet type.<sup>2</sup></li> </ul>   | 0 to 255                             |
| Specify Direction | (Optional) Enables traffic direction: <ul style="list-style-type: none"> <li>Direction—Lets you specify the direction of traffic:               <ul style="list-style-type: none"> <li>from-service—Traffic from service port destined to client port.</li> <li>to-service—Traffic from client port destined to service port.</li> </ul> </li> </ul> | from service<br>to service           |

Table B-26 Service SMB Advanced Engine Parameters (continued)

| Parameter                  | Description   | Value  |
|----------------------------|---|--|
| Specify Operation          | (Optional) Enables MSRPC over SMB: <ul style="list-style-type: none"> <li>MSRPC Over SMB Operation—Required for SMB_COM_TRANSACTION commands, exact match required.</li> </ul>        | 0 to 65535   |
| Specify Regex String       | (Optional) Enables searching for regex strings: <ul style="list-style-type: none"> <li>Regex String—A regular expression to search for in a single TCP packet.</li> </ul>             |  |
| Specify Exact Match Offset | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the Regex string must report a match to be valid.</li> </ul> |  |
| Specify Min Match Length   | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the Regex string must match.</li> </ul>                      |  |
| Specify Payload Source     | (Optional) Enables payload source: <ul style="list-style-type: none"> <li>Payload Source—Payload source inspection.<sup>3</sup></li> </ul>  |  |
| Specify Scan Interval      | (Optional) Enables scan interval: <ul style="list-style-type: none"> <li>Scan Interval—The interval in seconds used to calculate alert rates.</li> </ul>                              | 1 to 131071  |
| Specify TCP Flags          | (Optional) Enables TCP flags: <ul style="list-style-type: none"> <li>MSRPC TCP Flags</li> <li>MSRPC TCP Flags Mask</li> </ul>   | <ul style="list-style-type: none"> <li>concurrent execution</li> <li>did not execute</li> <li>first fragment</li> <li>last fragment</li> <li>maybe</li> <li>object UUID</li> <li>pending cancel</li> <li>reserved</li> </ul> |
| Specify Type               | (Optional) Enables type of MSRPC over SMB packet: <ul style="list-style-type: none"> <li>Type—Type field of MSRPC over SMB packet</li> </ul>  | <ul style="list-style-type: none"> <li>0 = Request</li> <li>2 = Response</li> <li>11 = Bind</li> <li>12 = Bind Ack</li> </ul>  |
| Specify UUID               | (Optional) Enables MSRPC over UUID: <ul style="list-style-type: none"> <li>UUID—MSRPC UUID field</li> </ul>   | 32-character string composed of hexadecimal characters 0-9, a-f, A-F.  |

**Table B-26** Service SMB Advanced Engine Parameters (continued)

| Parameter            | Description   | Value        |
|----------------------|---|--------------|
| Specify Hit Count    | (Optional) Enables hit counting: <ul style="list-style-type: none"> <li>Hit Count—The threshold number of occurrences in scan-interval to fire alerts.</li> </ul> | 1 to 65535   |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | True   False |

- The second number in the range must be greater than or equal to the first number.
- Currently supporting 37 (0x25) SMB\_COM\_TRANSACTION command & 162 (0xA2) SMB\_COM\_NT\_CREATE\_ANDX command.
- TCP\_Data performs regex over entire packet, SMB\_Data performs regex on SMB payload only, Resource\_DATA performs regex on SMB\_Resource.

## Service SNMP Engine

The Service SNMP engine inspects all SNMP packets destined for port 161. You can tune SNMP signatures and create custom SNMP signatures based on specific community names and object identifiers.

Instead of using string comparison or regular expression operations to match the community name and object identifier, all comparisons are made using the integers to speed up the protocol decode and reduce storage requirements.

Table B-27 lists the parameters specific to the Service SNMP engine.

**Table B-27** Service SNMP Engine Parameters

| Parameter                   | Description  | Value                                     |
|-----------------------------|--|---|
| Inspection Type             | Type of inspection to perform.   | —   |
| Brute Force Inspection      | Inspects for brute force attempts: <ul style="list-style-type: none"> <li>Bruce Force Count—The number of unique SNMP community names that constitute a brute force attempt.</li> </ul>  | 0 to 65535                                |
| Invalid Packet Inspection   | Inspects for SNMP protocol violations.   | —   |
| Non SNMP Traffic Inspection | Inspects for non-SNMP traffic destined for UDP port 161.   | —   |
| SNMP Inspection             | Inspects SNMP traffic: <ul style="list-style-type: none"> <li>Specify Community Name [yes   no]:               <ul style="list-style-type: none"> <li>Community Name—Searches for the SNMP community name, that is, the SNMP password.</li> </ul> </li> <li>Specify Object ID [yes   no]:               <ul style="list-style-type: none"> <li>Object ID—Searches for the SNMP object identifier.</li> </ul> </li> </ul> | <i>community-name</i><br><i>object-id</i> |

## Service SSH Engine

The Service SSH engine specializes in port 22 SSH traffic. Because all but the setup of an SSH session is encrypted, the engine only looks at the fields in the setup. There are two default signatures for SSH. You can tune these signatures, but you cannot create custom signatures.

Table B-28 lists the parameters specific to the Service SSH engine.

**Table B-28** Service SSH Engine Parameters

| Parameter            | Description   | Value                                |
|----------------------|---|--------------------------------------|
| SSH Version          |   |                                      |
| Length Type          | Inspects for one of the following SSH length types: <ul style="list-style-type: none"> <li>• Key Length—Length of the SSH key to inspect for:               <ul style="list-style-type: none"> <li>– Length—Keys larger than this fire the RSAREF overflow.</li> </ul> </li> <li>• User Length—User length SSH inspection:               <ul style="list-style-type: none"> <li>– Length—Keys larger than this fire the RSAREF overflow.</li> </ul> </li> </ul> | 0 to 65535                           |
| Service Ports        | A comma-separated list of ports or port ranges where the target service resides.  | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Specify Packet Depth | (Optional) Enables packet depth: <ul style="list-style-type: none"> <li>• Packet Depth—Number of packets to watch before determining the session key was missed.</li> </ul>   | 0 to 65535                           |

1. The second number in the range must be greater than or equal to the first number.

## Service TNS Engine

The Service TNS engine inspects TNS protocol. TNS provides database applications with a single common interface to all industry-standard network protocols. With TNS, applications can connect to other database applications across networks with different protocols. The default TNS listener port is TCP 1521. TNS also supports REDIRECT frames that redirect the client to another host and/or another TCP port. To support REDIRECT packets, the TNS engine listens on all TCP ports and has a quick TNS frame header validation routine to ignore non-TNS streams.

Table B-29 lists the parameters specific to the Service TNS engine.

**Table B-29 Service TNS Engine Parameters**

| Parameter                    | Description   | Value                             |
|------------------------------|---|-----------------------------------|
| Direction                    | Direction of traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port</li> <li>Traffic from client port destined to service port</li> </ul>  | From Service<br>To Service        |
| Type Frame Type              | Specifies the TNS frame value type: <ul style="list-style-type: none"> <li>1—Connect</li> <li>2—Accept</li> <li>4—Refuse</li> <li>5—Redirect</li> <li>6—Data</li> <li>11—Resend</li> <li>12—Marker</li> </ul>   | 1<br>2<br>4<br>5<br>6<br>11<br>12 |
| Specify Regex String         | (Optional) Enables using a regular expression string: <ul style="list-style-type: none"> <li>Specify Exact Match Offset—Enables the exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> </li> <li>Specify Min Match Length—Enables the minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul> </li> </ul> | 0 to 65535                        |
| Specify Regex Payload Source | Specifies which protocol to inspect:<br>Payload Source: <ul style="list-style-type: none"> <li>TCP Data—Performs Regex over the data portion of the TCP packet.</li> <li>TNS Data—Performs Regex only over the TNS data (with all white space removed).</li> </ul>  | TCP<br>TNS                        |

## State Engine

The State engine provides state-based regular expression-based pattern inspection of TCP streams. A state engine is a device that stores the state of something and at a given time can operate on input to transition from one state to another and/or cause an action or output to take place. State machines are used to describe a specific event that causes an output or alarm. There are three state machines in the State engine: SMTP, Cisco Login, and LPR Format String.

Table B-30 lists the parameters specific to the State engine.

**Table B-30 State Engine Parameters**

| Parameter                  | Description  | Value   |
|----------------------------|--|---|
| State Machine              | State machine grouping.  | <ul style="list-style-type: none"> <li>• SMPT</li> <li>• LPR Format String</li> <li>• Cisco Login</li> </ul>  |
| Cisco Login                | Specifies the state machine for Cisco login: <ul style="list-style-type: none"> <li>• State Name—Name of the state required before the signature fires an alert:               <ul style="list-style-type: none"> <li>– Cisco device state</li> <li>– Control-C state</li> <li>– Password prompt state</li> <li>– Start state</li> </ul> </li> </ul>   | <ul style="list-style-type: none"> <li>• Cisco Device</li> <li>• Control C</li> <li>• Pass Prompt</li> <li>• Start Cisco</li> </ul>                   |
| LPR Format String          | Specifies the state machine to inspect for the LPR format string vulnerability: <ul style="list-style-type: none"> <li>• State Name—Name of the state required before the signature fires an alert:               <ul style="list-style-type: none"> <li>– Abort state to end LPR Format String inspection</li> <li>– Format character state</li> <li>– State state</li> </ul> </li> </ul>                         | <ul style="list-style-type: none"> <li>• Abort</li> <li>• Format Char</li> <li>• Start</li> </ul>   |
| State Name                 | Specifies the state machine for the SMTP protocol: <ul style="list-style-type: none"> <li>• State Name—Name of the state required before the signature fires an alert:               <ul style="list-style-type: none"> <li>– Abort state to end LPR Format String inspection</li> <li>– Mail body state</li> <li>– Mail header state</li> <li>– SMTP commands state</li> <li>– Start state</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Abort</li> <li>• Mail Body</li> <li>• Mail Header</li> <li>• SMPT Commands</li> <li>• Start Abort</li> </ul> |
| Direction                  | Direction of the traffic: <ul style="list-style-type: none"> <li>• Traffic from service port destined to client port.</li> <li>• Traffic from client port destined to service port.</li> </ul>   | From Service<br>To Service  |
| Service Ports              | A comma-separated list of ports or port ranges where the target service resides.   | 0 to 65535 <sup>1</sup><br>a-b[,c-d]  |
| Specify Exact Match Offset | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>• Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul>   | 0 to 65535  |

**Table B-30 State Engine Parameters (continued)**

| Parameter                | Description  | Value      |
|--------------------------|--|------------|
| Specify Max Match Offset | (Optional) Enables maximum match offset: <ul style="list-style-type: none"> <li>Max Match Offset—The maximum stream offset the regular expression string must report for a match to be valid.</li> </ul> | 0 to 65535 |
| Specify Min Match Offset | (Optional) Enables minimum match offset: <ul style="list-style-type: none"> <li>Min Match Offset—The minimum stream offset the regular expression string must report for a match to be valid.</li> </ul> | 0 to 65535 |
| Specify Min Match Length | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul>                            | 0 to 65535 |
| Swap Attacker Victim     | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).  | Yes   No   |

1. The second number in the range must be greater than or equal to the first number.

## String Engines

This section describes the String engine, and contains the following topics:

- [Understanding String Engines, page B-39](#)
- [String ICMP Engine Parameters, page B-40](#)
- [String TCP Engine Parameters, page B-40](#)
- [String UDP Engine Parameters, page B-41](#)

## Understanding String Engines

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.

## String ICMP Engine Parameters

Table B-31 lists the parameters specific to the String ICMP engine.

**Table B-31** String ICMP Engine Parameters

| Parameter                  | Description  | Value                             |
|----------------------------|--|-----------------------------------|
| Direction                  | Direction of the traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port.</li> <li>Traffic from client port destined to service port.</li> </ul>             | From Service<br>To Service        |
| ICMP Type                  | ICMP header TYPE value.  | 0 to 18 <sup>1</sup><br>a-b[,c-d] |
| Specify Exact Match Offset | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> | 0 to 65535                        |
| Specify Min Match Length   | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul>                          | 0 to 65535                        |
| Swap Attacker Victim       | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).  | Yes   No                          |

1. The second number in the range must be greater than or equal to the first number.

### For More Information

For an example custom String engine signature, see [Example String TCP Signature, page 7-21](#).

## String TCP Engine Parameters

Table B-32 lists the parameters specific to the String TCP engine.

**Table B-32** String TCP Engine

| Parameter                  | Description  | Value                                |
|----------------------------|--|--------------------------------------|
| Direction                  | Direction of the traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port.</li> <li>Traffic from client port destined to service port.</li> </ul>             | From Service<br>To Service           |
| Service Ports              | A comma-separated list of ports or port ranges where the target service resides.   | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Specify Exact Match Offset | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> | 0 to 65535                           |

**Table B-32** String TCP Engine (continued)

| Parameter                | Description   | Value        |
|--------------------------|---|--------------|
| Specify Min Match Length | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul> | 0 to 65535   |
| Strip Telnet Options     | Strips the Telnet option characters from the data before the pattern is searched. <sup>2</sup>  | True   False |
| Swap Attacker Victim     | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).   | Yes   No     |

- The second number in the range must be greater than or equal to the first number.
- This parameter is primarily used as an IPS anti-evasion tool.

**For More Information**

For an example custom String engine signature, see [Example String TCP Signature, page 7-21](#).

## String UDP Engine Parameters

[Table B-33](#) lists the parameters specific to the String UDP engine.

**Table B-33** String UDP Engine

| Parameter                  | Description  | Value                                |
|----------------------------|--|--------------------------------------|
| Direction                  | Direction of the traffic: <ul style="list-style-type: none"> <li>Traffic from service port destined to client port.</li> <li>Traffic from client port destined to service port.</li> </ul>             | From Service To Service              |
| Service Ports              | A comma-separated list of ports or port ranges where the target service resides.   | 0 to 65535 <sup>1</sup><br>a-b[,c-d] |
| Specify Exact Match Offset | (Optional) Enables exact match offset: <ul style="list-style-type: none"> <li>Exact Match Offset—The exact stream offset the regular expression string must report for a match to be valid.</li> </ul> | 0 to 65535                           |
| Specify Min Match Length   | (Optional) Enables minimum match length: <ul style="list-style-type: none"> <li>Min Match Length—Minimum number of bytes the regular expression string must match.</li> </ul>                          | 0 to 65535                           |
| Swap Attacker Victim       | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).  | Yes   No                             |

- The second number in the range must be greater than or equal to the first number.

**For More Information**

For an example custom String engine signature, see [Example String TCP Signature, page 7-21](#).

# Sweep Engines

This section describes the Sweep engines, and contains the following topics:

- [Sweep Engine, page B-42](#)
- [Sweep Other TCP Engine, page B-43](#)

## Sweep Engine

The Sweep engine analyzes traffic between two hosts or from one host to many hosts. You can tune the existing signatures or create custom signatures. The Sweep engine has protocol-specific parameters for ICMP, UDP, and TCP.

The alert conditions of the Sweep engine ultimately depend on the count of the unique parameter. The unique parameter is the threshold number of distinct hosts or ports depending on the type of sweep. The unique parameter triggers the alert when more than the unique number of ports or hosts is seen on the address set within the time period. The processing of unique port and host tracking is called counting.

A unique parameter must be specified for all signatures in the Sweep engine. A limit of 2 through 40 (inclusive) is enforced on the sweeps. 2 is the absolute minimum for a sweep, otherwise, it is not a sweep (of one host or port). 40 is a practical maximum that must be enforced so that the sweep does not consume excess memory. More realistic values for unique range between 5 and 15.

TCP sweeps must have a TCP flag and mask specified to determine which sweep inspector slot in which to count the distinct connections.

The ICMP sweeps must have an ICMP type specified to discriminate among the various types of ICMP packets.

[Table B-34](#) lists the parameters specific to the Sweep engine.

**Table B-34** Sweep Engine Parameters

| Parameter          | Description  | Value   |
|--------------------|--|---|
| Protocol           | Protocol of interest for this inspector.   | <ul style="list-style-type: none"> <li>• ICMP</li> <li>• UDP</li> <li>• TCP</li> </ul>                  |
| Specify ICMP Type  | (Optional) Enables the ICMP header type: <ul style="list-style-type: none"> <li>• ICMP Type—ICMP header TYPE value.</li> </ul>   | 0 to 255  |
| Specify Port Range | (Optional) Enables using a port range for inspection: <ul style="list-style-type: none"> <li>• Port Range—UDP port range used in inspection.</li> </ul>                                    | 0 to 65535<br>a-b[,c-d]   |
| Fragment Status    | Specifies whether fragments are wanted or not: <ul style="list-style-type: none"> <li>• Any fragment status.</li> <li>• Do not inspect fragments.</li> <li>• Inspect fragments.</li> </ul> | <ul style="list-style-type: none"> <li>• Any</li> <li>• No Fragment</li> <li>• Want Fragment</li> </ul> |
| Inverted Sweep     | Uses source port instead of destination port for unique counting.  | True   False  |

Table B-34 Sweep Engine Parameters (continued)

| Parameter            | Description  | Value  |
|----------------------|--|--|
| Mask                 | Mask used in TCP flags comparison: <ul style="list-style-type: none"> <li>• URG bit</li> <li>• ACK bit</li> <li>• PSH bit</li> <li>• RST bit</li> <li>• SYN bit</li> <li>• FIN bit</li> </ul>                | <ul style="list-style-type: none"> <li>• URG</li> <li>• ACK</li> <li>• PSH</li> <li>• RST</li> <li>• SYN</li> <li>• FIN</li> </ul> |
| Storage Key          | Type of address key used to store persistent data: <ul style="list-style-type: none"> <li>• Attacker address</li> <li>• Attacker and victim addresses</li> <li>• Attacker address and victim port</li> </ul> | Axxx<br>AxBx<br>Axxb   |
| Suppress Reverse     | Does not fire when a sweep has fired in the reverse direction on this address set.   | Yes   No   |
| Swap Attacker Victim | True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).  | Yes   No   |
| TCP Flags            | TCP flags to match when masked by mask: <ul style="list-style-type: none"> <li>• URG bit</li> <li>• ACK bit</li> <li>• PSH bit</li> <li>• RST bit</li> <li>• SYN bit</li> <li>• FIN bit</li> </ul>           | <ul style="list-style-type: none"> <li>• URG</li> <li>• ACK</li> <li>• PSH</li> <li>• RST</li> <li>• SYN</li> <li>• FIN</li> </ul> |
| Unique               | Threshold number of unique port connections between the two hosts.   | 0 to 65535   |

## Sweep Other TCP Engine

The Sweep Other TCP engine analyzes traffic between two hosts looking for abnormal packets typically used to fingerprint a victim. You can tune the existing signatures or create custom signatures.

TCP sweeps must have a TCP flag and mask specified. You can specify multiple entries in the set of TCP flags. And you can specify an optional port range to filter out certain packets.

Table B-35 lists the parameters specific to the Sweep Other TCP engine.

**Table B-35 Sweep Other TCP Engine Parameters**

| Parameter          | Description   | Value  |
|--------------------|---|--|
| Specify Port Range | (Optional) Enables using a port range for inspection: <ul style="list-style-type: none"> <li>Port Range—UDP port range used in inspection.</li> </ul>   | 0 to 65535<br>a-b[,c-d]  |
| Set TCP Flags      | Lets you set TCP flags to match. <ul style="list-style-type: none"> <li>TCP Flags—TCP flags used in this inspection:                             <ul style="list-style-type: none"> <li>URG bit</li> <li>ACK bit</li> <li>PSH bit</li> <li>RST bit</li> <li>SYN bit</li> <li>FIN bit</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>URG</li> <li>ACK</li> <li>PSH</li> <li>RST</li> <li>SYN</li> <li>FIN</li> </ul> |

## Traffic Anomaly Engine

The Traffic Anomaly engine contains nine anomaly detection signatures covering the three protocols (TCP, UDP, and other). Each signature has two subsignatures, one for the scanner and the other for the worm-infected host (or a scanner under worm attack). When anomaly detection discovers an anomaly, it triggers an alert for these signatures. All anomaly detection signatures are enabled by default and the alert severity for each one is set to high.

When a scanner is detected but no histogram anomaly occurred, the scanner signature fires for that attacker (scanner) IP address. If the histogram signature is triggered, the attacker addresses that are doing the scanning each trigger the worm signature (instead of the scanner signature). The alert details state which threshold is being used for the worm detection now that the histogram has been triggered.

From that point on, all scanners are detected as worm-infected hosts.

The following anomaly detection event actions are possible:

- Produce alert—Writes the event to the Event Store.
- Deny attacker inline—(inline only) Does not transmit this packet and future packets originating from the attacker address for a specified period of time.
- Log attacker pairs—Starts IP logging for packets that contain the attacker address.
- Log pair packets—Starts IP logging for packets that contain the attacker and victim address pair.
- Deny attacker service pair inline—Blocks the source IP address and the destination port.
- Request SNMP trap—Sends a request to NotificationApp to perform SNMP notification.
- Request block host—Sends a request to ARC to block this host (the attacker).



**Note**

You can edit or tune anomaly detection signatures but you cannot create custom anomaly detection signatures.

Table 36 lists the anomaly detection worm signatures.

**Table 36** *Anomaly Detection Worm Signatures*

| <b>Signature ID</b> | <b>Subsignature ID</b> | <b>Name</b>            | <b>Description</b>   |
|---------------------|------------------------|------------------------|--|
| 13000               | 0                      | Internal TCP Scanner   | Identified a single scanner over a TCP protocol in the internal zone.  |
| 13000               | 1                      | Internal TCP Scanner   | Identified a worm attack over a TCP protocol in the internal zone; the TCP histogram threshold was crossed and a scanner over a TCP protocol was identified.         |
| 13001               | 0                      | Internal UDP Scanner   | Identified a single scanner over a UDP protocol in the internal zone.  |
| 13001               | 1                      | Internal UDP Scanner   | Identified a worm attack over a UDP protocol in the internal zone; the UDP histogram threshold was crossed and a scanner over a UDP protocol was identified.         |
| 13002               | 0                      | Internal Other Scanner | Identified a single scanner over an Other protocol in the internal zone.   |
| 13002               | 1                      | Internal Other Scanner | Identified a worm attack over an Other protocol in the internal zone; the Other histogram threshold was crossed and a scanner over an Other protocol was identified. |
| 13003               | 0                      | External TCP Scanner   | Identified a single scanner over a TCP protocol in the external zone.  |
| 13003               | 1                      | External TCP Scanner   | Identified a worm attack over a TCP protocol in the external zone; the TCP histogram threshold was crossed and a scanner over a TCP protocol was identified.         |
| 13004               | 0                      | External UDP Scanner   | Identified a single scanner over a UDP protocol in the external zone.  |
| 13004               | 1                      | External UDP Scanner   | Identified a worm attack over a UDP protocol in the external zone; the UDP histogram threshold was crossed and a scanner over a UDP protocol was identified.         |
| 13005               | 0                      | External Other Scanner | Identified a single scanner over an Other protocol in the external zone.   |
| 13005               | 1                      | External Other Scanner | Identified a worm attack over an Other protocol in the external zone; the Other histogram threshold was crossed and a scanner over an Other protocol was identified. |
| 13006               | 0                      | Illegal TCP Scanner    | Identified a single scanner over a TCP protocol in the external zone.  |
| 13006               | 1                      | Illegal TCP Scanner    | Identified a worm attack over a TCP protocol in the external zone; the TCP histogram threshold was crossed and a scanner over a TCP protocol was identified.         |

**Table 36** Anomaly Detection Worm Signatures (continued)

| Signature ID | Subsignature ID | Name                  | Description  |
|--------------|-----------------|-----------------------|--|
| 13007        | 0               | Illegal UDP Scanner   | Identified a single scanner over a UDP protocol in the external zone.  |
| 13007        | 1               | Illegal UDP Scanner   | Identified a worm attack over a UDP protocol in the external zone; the UDP histogram threshold was crossed and a scanner over a UDP protocol was identified.         |
| 13008        | 0               | Illegal Other Scanner | Identified a single scanner over an Other protocol in the external zone.   |
| 13008        | 1               | Illegal Other Scanner | Identified a worm attack over an Other protocol in the external zone; the Other histogram threshold was crossed and a scanner over an Other protocol was identified. |

## Traffic ICMP Engine

The Traffic ICMP engine analyzes nonstandard protocols, such as TFN2K, LOKI, and DDoS. There are only two signatures (based on the LOKI protocol) with user-configurable parameters.

TFN2K is the newer version of the TFN. It is a DDoS agent that is used to control coordinated attacks by infected computers (zombies) to target a single computer (or domain) with bogus traffic floods from hundreds or thousands of unknown attacking hosts. TFN2K sends randomized packet header information, but it has two discriminators that can be used to define signatures. One is whether the L3 checksum is incorrect and the other is whether the character 64 'A' is found at the end of the payload. TFN2K can run on any port and can communicate with ICMP, TCP, UDP, or a combination of these protocols.

LOKI is a type of back door Trojan. When the computer is infected, the malicious code creates an ICMP Tunnel that can be used to send small payload in ICMP replies (which may go straight through a firewall if it is not configured to block ICMP.) The LOKI signatures look for an imbalance of ICMP echo requests to replies and simple ICMP code and payload discriminators.

The DDoS category (excluding TFN2K) targets ICMP-based DDoS agents. The main tools used here are TFN and Stacheldraht. They are similar in operation to TFN2K, but rely on ICMP only and have fixed commands: integers and strings.

[Table B-37](#) lists the parameters specific to the Traffic ICMP engine.

**Table B-37** Traffic ICMP Engine Parameters

| Parameter             | Description   | Value                  |
|-----------------------|---|------------------------|
| Parameter Tunable Sig | Whether this signature has configurable parameters.   | Yes   No               |
| Inspection Type       | Type of inspection to perform: <ul style="list-style-type: none"> <li>Inspects for original LOKI traffic.</li> <li>Inspects for modified LOKI traffic.</li> </ul> | Is Loki<br>Is Mod Loki |

**Table B-37** Traffic ICMP Engine Parameters (continued)

| Parameter    | Description  | Value        |
|--------------|--|--------------|
| Reply Ratio  | Inbalance of replies to requests. The alert fires when there are this many more replies than requests. | 0 to 65535   |
| Want Request | Requires an ECHO REQUEST be seen before firing the alert.  | True   False |

## Trojan Engines

The Trojan engines analyze nonstandard protocols, such as BO2K and TFN2K. There are three Trojan engines: Trojan BO2K, TrojanTFN2K, and Trojan UDP.

BO was the original Windows back door Trojan that ran over UDP only. It was soon superseded by BO2K. BO2K supported UDP and TCP both with basic XOR encryption. They have plain BO headers that have certain cross-packet characteristics.

BO2K also has a stealthy TCP module that was designed to encrypt the BO header and make the cross-packet patterns nearly unrecognizable.

The UDP modes of BO and BO2K are handled by the Trojan UDP engine. The TCP modes are handled by the Trojan BO2K engine.

There are no specific parameters to the Trojan engines, except for swap-attacker-victim in the Trojan UDP engine.

