Signature Engines

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

About 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 IPS 6.0 engines support a standardized Regex.

IPS 6.0 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 in to 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

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

Table A-1 lists the general master engine parameters.

Table A-1   Master Engine Parameters

Parameter	Description	Value
Signature ID	Specifies the ID of this signature.	*number*
Sub Signature ID	Specifies the sub ID of this signature	*number*
Alert Severity	Specifies the severity of the alert: • Dangerous alert • Medium-level alert • Low-level alert • Informational alert	• High • Medium • Low • Informational (default)
Sig Fidelity Rating	Specifies the rating of the fidelity of this signature.	0 to 100 (default = 100)
Promiscuous Delta	Specifies the delta value used to determine the seriousness of the alert.	0 to 30 (default = 5)
Signature Name	Specifies the name of the signature.	*sig-name*
Alert Notes	Provides additional information about this signature that will be included in the alert message.	*alert-notes*
User Comments	Provides comments about this signature.	*comments*
Alert Traits	Specifies traits you want to document about this signature.	0 to 65335
Release	Provides the release in which the signature was most recently updated.	*release*
Signature Creation Date	Specifies the date the signature was created.	—
Signature Type	Specifies the signature category.	• Anomaly • Component • Exploit • Other
Engine	Specifies the engine to which the signature belongs. **Note** The engine-specific parameters appear under the Engine category.	—
Event Count	Specifies the number of times an event must occur before an alert is generated.	1 to 65535 (default = 1)

Table A-1   Master Engine Parameters (continued)

Parameter	Description	Value
Event Count Key	Specifies the storage type on which to count events for this signature: • Attacker address • Attacker and victim addresses • Attacker address and victim port • Victim address • Attacker and victim addresses and ports	• Axxx • AxBx • Axxb • xxBx • AaBb
Specify Alert Interval {Yes | No}	Enables the alert interval: • Alert Interval—Specifies the time in seconds before the event count is reset.	2 to 1000
Status	Specifies whether the signature is enabled or disabled, active or retired.	Enabled | Retired {Yes | No}
Obsoletes	Indicates that a newer signature has disabled an older signature.	—
Vulnerable OS List	When combined with passive OS fingerprinting, it allows the IPS to determine if it is likely a given attack is relevant to the target system.	AIX BSD General OS HP-UX IOS IRIX Linux Mac OS Netware Other Solaris UNIX Windows Windows NT Windows NT/2K/XP
Mars Category {Yes | No}	Maps signatures to a MARS attack category.[1]	—

1. This is a static information category that you can set in the configuration and view in the alerts.Refer to the MARS documentation for more information.

Promiscuous Delta

The 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 so that they never reach the target host, so it does not matter if the target was vulnerable. Because the attack was not allowed on the network, the IPS does 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 recommend that you do NOT change the promiscuous delta setting for a signature.

Obsoletes

The Cisco signature team uses the obsoletes field to indicate obsoleted, older signatures that have been replaced by newer, better signatures, and to indicate disabled signatures in an engine when a better instance of that engine is available.

Vulnerable OS List

When you combine the vulnerable OS setting of a signature with passive OS fingerprinting, the IPS can determine if it is likely that a given attack is relevant to the target system. If the attack is found to be relevant, the risk rating value of the resulting alert receives a boost. If the relevancy is unknown, usually because there is no entry in the passive OS fingerprinting list, then no change is made to the risk rating. If there is a passive OS fingerprinting entry and it does not match the vulnerable OS setting of a signature, the risk rating value is decreased. The default value by which to increase or decrease the risk rating is +/- 10 points.

For More Information

- For more information about promiscuous mode, see Promiscuous Mode, page 3-12.

- For more information about passive OS fingerprinting, see Monitoring OS Identifications, page 6-37.

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 A-2 lists the alert frequency parameters.

Table A-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

Table A-2   Master Engine Alert Frequency Parameters (continued)

Parameter	Description	Value
summary-threshold	Threshold number of alerts to send signature in to 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 in to 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: • Attacker address • Attacker and victim addresses • Attacker address and victim port • Victim address • Attacker and victim addresses and ports	Axxx AxBx Axxb xxBx AaBb

Event Actions

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:

- produce-alert—Writes an evIdsAlert to the Event Store.

- produce-verbose-alert—Includes an encoded dump (possibly truncated) of the offending packet in the evIdsAlert.

- deny-attacker-inline—Does not transmit this packet and future packets from the attacker address for a specified period of time (inline only).

> **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 global-deny-timeout in Event Action Rules. You can clear all denied attacker entries with the **clear denied-attackers** command, which permits the addresses back on the network.

- deny-connection-inline—Does not transmit this packet and future packets on the TCP Flow (inline only).

- deny-packet-inline—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.

- log-attacker-packets—Starts IP logging of packets containing the attacker address (inline only).

- log-pair-packets—Starts IP logging of packets containing the attacker-victim address pair.

- log-victim-packets—Starts IP logging of packets containing the victim address.
- request-block-connection—Requests Network Access Controller to block this connection.
- request-block-host—Requests Network Access Controller to block this attacker host.
- request-snmp-trap—Sends request to NotificationApp to perform SNMP action.
- reset-tcp-connection—Sends TCP resets to hijack and terminate the TCP flow.
- modify-packet-inline—Modifies packet contents (inline only).

Note Modify-packet-inline is a new feature from the inline normalizer. It scrubs the packet and corrects irregular issues such as bad checksum, out of range values, and other RFC violations.

Understanding Deny Packet Inline

For signatures that have Deny Packet Inline configured as an action or for an event action override that adds Deny Packet Inline as an action, the following actions may be taken:

- droppedPacket
- deniedFlow
- tcpOneWayResetSent

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

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

Regular Expression Syntax

Regular expressions (Regex) are a powerful and flexible notational language that allow you to describe text. In the context of pattern matching, regular expressions allow a succinct description of any arbitrary pattern.

Table A-3 lists the IPS signature Regex syntax.

Table A-3   *Signature Regular Expression Syntax*

Metacharacter	Name	Description
?	Question mark	Repeat 0 or 1 times.
*	Star, asterisk	Repeat 0 or more times.
+	Plus	Repeat 1 or more times.
{x}	Quantifier	Repeat exactly *X* times.

Table A-3   Signature Regular Expression Syntax (continued)

Metacharacter	Name	Description
{x,}	Minimum quantifier	Repeat at least *X* times.
.	Dot	Any one character except new line (0x0A).
[abc]	Character class	Any character listed.
[^abc]	Negated character class	Any character not listed.
[a-z]	Character range class	Any character listed inclusively in the range.
( )	Parenthesis	Used to limit the scope of other metacharacters.
|	Alternation, or	Matches either expression it separates.
^	caret	The beginning of the line.
char	Escaped character	When *char* is a metacharacter or not, matches the literal *char*.
char	Character	When char is not a metacharacter, matches the literal char.
\r	Carriage return	Matches the carriage return character (0x0D).
\n	New line	Matches the new line character (0x0A).
\t	Tab	Matches the tab character (0x09).
\f	Form feed	Matches the form feed character (0x0C).
\xNN	Escaped hexadecimal character	Matches character with the hexadecimal code 0xNN (0<=N<=F).
\NNN	Escaped octal character	Matches the character with the octal code NNN (0<=N<=8).

All repetition operators will match the shortest possible string as opposed to other operators that consume as much of the string as possible thus giving the longest string match.

lists examples of Regex patterns.

Table A-4   Regex Patterns

To Match	Regular Expression
Hacker	Hacker
Hacker or hacker	[Hh]acker
Variations of bananas, banananas, bananananas	ba(na)+s

Table A-4   Regex Patterns (continued)

To Match	Regular Expression
foo and bar on the same line with anything except a new line between them	foo.*bar
Either foo or bar	foo|bar
Either moon or soon	(m|s)oon

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 A-10
- AIC Engine and Sensor Performance, page A-11
- AIC Engine Parameters, page A-11

Understanding the AIC Engine

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

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

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.

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

⚠️

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

Table A-5 lists the parameters that are specific to the AIC HTTP engine.

Table A-5   AIC HTTP Engine Parameters

Parameter	Description
signature-type	Specifies the type of AIC signature.
content-types	AIC signature that deals with MIME types: • 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. • define-recognized-content-types lists content types recognized by the sensor.
define-web-traffic-policy	Specifies the action to take when noncompliant HTTP traffic is seen. The **alarm-on-non-http-traffic [true | false]** command enables the signature. This signature is disabled by default.

Table A-5   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: • define-request-method, such as get, put, and so forth. • recognized-request-methods lists methods recognized by the sensor.
transfer-encodings	AIC signature that deals with transfer encodings: • define-transfer-encoding associates an action with each method, such as compress, chunked, and so forth. • recognized-transfer-encodings lists methods recognized by the sensor. • chunked-transfer-encoding-error specifies actions to be taken when a chunked encoding error is seen.

Table A-6 lists the parameters that are specific to the AIC FTP engine.

Table A-6   AIC FTP Engine Parameters

Parameter	Description
signature-type	Specifies the type of AIC signature.
ftp-commands	Associates an action with an FTP command: • ftp-command—Lets you choose the FTP command you want to inspect.
unrecognized-ftp-command	Inspects unrecognized FTP commands.

For More Information

- For the procedures for configuring AIC engine signatures, see Configuring Application Policy Signatures, page 5-59.
- For an example of a custom AIC signature, see Example Recognized Define Content Type (MIME) Signature, page 5-68.
- For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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

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

Table A-7 lists the parameters that are specific to the Atomic ARP engine.

Table A-7   Atomic ARP Engine Parameters

Parameter	Description
specify-mac-flip	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: • Source Broadcast (default)—Fires an alarm for this signature when it sees an ARP source address of 255.255.255.255. • Destination Broadcast—Fires an alarm for this signature when it sees an ARP destination address of 255.255.255.255. • 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 • Source Multicast—Fires an alarm for this signature when it sees an ARP source MAC address of 01:00:5e:(00-7f).
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 A-8 lists the parameters that are specific to the Atomic IP engine.

Table A-8   Atomic IP Engine Parameters

Parameter	Description
fragment-status	Specifies whether or not fragments are wanted.
specify-ip-payload-length	Specifies IP datagram payload length.
specify-ip-header-length	Specifies IP datagram header length.
specify-ip-addr-options	Specifies IP addresses.
specify-ip-id	Specifies IP identifier.
specify-ip-total-length	Specifies IP datagram total length.
specify-ip-option-inspection	Specifies IP options inspection.
specify-l4-protocol	Specifies Layer 4 protocol.
specify-ip-tos	Specifies type of server.
specify-ip-ttl	Specifies time to live.
specify-ip-version	Specifies IP protocol version.

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 A-9 lists the Atomic IPv6 signatures.

Table A-9   *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	Multiple first fragment packets	Produces an alert when more than one first fragment is seen in a 30-second period.

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 A-10 lists the parameters specific to the Flood Host engine.

Table A-10   *Flood Host Engine Parameters*

Parameter	Description	Value
protocol	Which kind of traffic to inspect.	ICMP UDP
rate	Threshold number of packets per second.	0 to 65535[1]
icmp-type	Specifies the value for the ICMP header type.	0 to 65535

Table A-10   Flood Host Engine Parameters (continued)

Parameter	Description	Value
dst-ports	Specifies the destination ports when you choose UDP protocol.	0 to 65535[2] a-b[,c-d]
src-ports	Specifies the source ports when you choose UDP protocol.	0 to 65535[3] 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 A-11 lists the parameters specific to the Flood Net engine.

Table A-11   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 TCP UDP
rate	Threshold number of packets per second.	0 to 65535[1]
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 A-12 lists the parameters specific to the Meta engine.

Table A-12   Meta Engine Parameters

Parameter	Description	Value
meta-reset-interval	Time in seconds to reset the META signature.	0 to 3600

component-list	List of Meta components:
	• edit—Edits an existing entry
	• insert—Inserts a new entry in to the list:
	  – begin—Places the entry at the beginning of the active list
	  – end—Places the entry at the end of the active list
	  – inactive—Places the entry in to the inactive list
	  – before—Places the entry before the specified entry
	  – after—Places the entry after the specified entry
	• move—Moves an entry in the list
		name1

meta-key	Storage type for the Meta signature: • Attacker address • Attacker and victim addresses • Attacker and victim addresses and ports • Victim address	AaBb AxBx Axxx xxBx
unique-victim-ports	Number of unique victims ports required per Meta signature.	1 to 256
component-list-in-order	Whether to fire the component list in order.	true | false

For More Information

- For more information about the Signature Event Action Processor, see Signature Event Action Processor, page 6-5.
- For an example of a custom Meta engine signature, see Example Meta Engine Signature, page 5-23.

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 A-13 lists the parameters specific to the Multi String Engine.

Table A-13   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 tcp udp
regex-component	List of regex components: • regex-string—The string to search for. • 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.	list (1 to 16 items) exact minimum
port-selection	Type of TCP or UDP port to inspect: • both-ports—Specifies both source and destination port. • dest-ports—Specifies a range of destination ports. • source-ports—Specifies a range of source ports.[1]	0 to 65535 [2]
exact-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
min-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).	true | false

1. 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.

2. 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.

For More Information

For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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 A-19
- Normalizer Engine Parameters, page A-22

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.

AIP SSM and the Normalizer Engine

The majority of the features in the Normalizer engine are not used on the AIP SSM, because the ASA itself handles the normalization. Packets on the ASA IPS modules go through a special path in the Normalizer that only reassembles fragments and puts packets in the right order for the TCP stream. The Normalizer does not do any of the normalization that is done on an inline IPS appliance, because that causes problems in the way the ASA handles the packets.

The following Normalizer engine signatures are not supported:

- 1300.0
- 1304.0
- 1305.0
- 1307.0
- 1308.0
- 1309.0
- 1311.0
- 1315.0
- 1316.0
- 1317.0
- 1330.0
- 1330.1
- 1330.2
- 1330.9
- 1330.10
- 1330.12
- 1330.14
- 1330.15
- 1330.16
- 1330.17
- 1330.18

Normalizer Engine Signatures

The following Normalizer engine signatures have built-in safeguards that you cannot override through configuration. They have specific actions based on the type of normalization they are performing even if they are disabled.

- Signature 1200—IP Fragmentation Buffer Full

  When disabled, the default values are used and no alert is sent.

- Signature 1202—Datagram Too Long

  When disabled, the default values are used and the packet is dropped.

- Signature 1204—No Initial Fragment

  When disabled, no alert is sent and no inspection is done on the datagram.

- Signature 1205—Too Many Datagrams

    When disabled, the default settings are used to protect the IPS.

- Signature 1207—Too Many Fragments

    When disabled, the default settings are still used to protect the IPS.

- Signature 1208—Incomplete Datagram

    When disabled, the default settings are still used.

- Signature 1330 Subsignature 0—TCP Drop-Bad Checksum

    When disabled, the packet checksum is ignored and the packet is processed even though it has a bad checksum.

- Signature 1330 Subsignature 1—TCP Drop-Bad TCP Flags

    When disabled, it is the same as having no actions. Packets are not sent for inspection regardless of the settings.

- Signature 1330 Subsignature 4—TCP Drop-Bad Option Length

    When disabled, it is the same as having no actions set. The packet is modified and processed for inspection.

- Signature 1330 Subsignature 7—TCP Drop-Bad Window Scale Value

    When disabled, it is the same as having no actions set. The packet is modified and processed for inspection.

- Signature 1330 Subsignature 12—TCP Drop-Segment Out of Order

    When disabled, is the same as having no actions set. Out of order queuing is not prevented.

- Signature 1330 Subsignature 19—TCP Timestamp Option Detected When Not Expected

    When disabled, it is the same as having no actions set. The packet is modified and processed for inspection.

- Signature 1330 Subsignature 20—TCP Window Scale Option Detected When Not Expected

    When disabled, it is the same as having no actions set. The packet is modified and processed for inspection.

- Signature 1330 Subsignature 21—TCP Option SACK Data Detected When Not Expected

    When disabled, it is the same as having no actions set. The packet is modified and processed for inspection.

For More Information

For the procedures for configuring signatures in the Normalizer engine, see Configuring IP Fragment Reassembly Signatures, page 5-69, and Configuring TCP Stream Reassembly Signatures, page 5-73.

Normalizer Engine Parameters

Table A-14 lists the parameters that are specific to the Normalizer engine.

Table A-14   Normalizer Engine Parameters

Parameter	Description
edit-default-sigs-only	Editable signatures.
specify-fragment-reassembly-timeout	(Optional) Enables fragment reassembly timeout.
specify-hijack-max-old-ack	(Optional) Enables hijack-max-old-ack.
specify-max-dgram-size	(Optional) Enables maximum datagram size.
specify-max-fragments	(Optional) Enables maximum fragments.
specify-max-fragments-per-dgram	(Optional) Enables maximum fragments per datagram.
specify-max-last-fragments	(Optional) Enables maximum last fragments.
specify-max-partial-dgrams	(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 contains the following topics:

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 A-15 lists the parameters specific to the Service DNS engine.

Table A-15   Service DNS Engine Parameters

Parameter	Description	Value
protocol	Protocol of interest for this inspector.	TCP UDP
specify-query-chaos-string	(Optional) Enables the DNS Query Class Chaos String.	*query-chaos-string*
specify-query-class	(Optional) Enables the query class: • query-class—DNS Query Class 2 Byte Value	0 to 65535
specify-query-invalid-domain-name	(Optional) Enables query invalid domain name: • query-invalid-domain-name—DNS Query Length greater than 255	true | false
specify-query-jump-count-exceeded	(Optional) Enables query jump count exceeded: • query-jump-count-exceeded—DNS compression counter	true | false
specify-query-opcode	(Optional) Enables query opcode: • query-opcode—DNS Query Opcode 1 byte Value	0 to 65535

Table A-15   Service DNS Engine Parameters (continued)

Parameter	Description	Value
specify-query-record-data-invalid	(Optional) Enables query record data invalid: • query-record-data-invalid—DNS Record Data incomplete	true | false
specify-query-record-data-len	(Optional) Enables the query record data length: • query-record-data-len—DNS Response Record Data Length	0 to 65535
specify-query-src-port-53	(Optional) Enables the query source port 53: • query-src-port-53—DNS packet source port 53	true | false
specify-query-stream-len	(Optional) Enables the query stream length: • query-stream-len—DNS Packet Length	0 to 65535
specify-query-type	(Optional) Enables the query type: • query-type—DNS Query Type 2 Byte Value	0 to 65535
specify-query-value	(Optional) Enables the query value: • query-value—Query 0 Response 1	true | false

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 A-16 lists the parameters that are specific to the Service FTP engine.

Table A-16   Service FTP Engine Parameters

Parameter	Description	Value
direction	Direction of traffic: • Traffic from service port destined to client port • Traffic from client port destined to service port	from-service to-service
ftp-inspection-type	Type of inspection to perform: • Looks for an invalid address in the FTP port command • Looks for an invalid port in the FTP port command • Looks for the PASV port spoof	bad-port-cmd-address bad-port-cmd-port pasv
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
swap-attacker-victim	Swaps the attacker and victim addresses and ports (source and destination) in the alert message and in any actions taken.	yes | no (default)

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

Service Generic Engines

This section describes the Service Generic engines and their parameters. It contains the following topics:

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.

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 A-17 lists the parameters specific to the Service Generic engine.

Table A-17   Service Generic Engine Parameters

Parameter	Description	Value
specify-dst-port	(Optional) Enables the destination port: • dst-port—Destination port of interest for this signature	0 to 65535
specify-ip-protocol	(Optional) Enables IP protocol: • ip-protocol—The IP protocol this inspector should examine	0 to 255

specify-payload-source	(Optional) Enables payload source inspection:
	• payload-source—Payload source inspection for the following types:
	  – Inspects ICMP data
	  – Inspects Layer 2 headers
	  – Inspects Layer 3 headers
	  – Inspects Layer 4 headers
	  – Inspects TCP data
	  – Inspects UDP data
		icmp-data
		l2-header
		l3-header
		l4-header
		tcp-data
		udp-data

specify-src-port	(Optional) Enables the source port: • src-port—Source port of interest for this signature	0 to 65535

Service Generic Advanced Engine

The Service Generic Advanced engine adds the Regex parameter to the functionality of the Service Generic engine and enhanced instructions. The Service Generic Advanced engine analyzes 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 Advanced 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 Advanced engine signature parameters other than severity and event action.

Service H225 Engine

This section describes the Service H225 engine, and contains the following topics:

Understanding the Service H255 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.255.0 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.

Service H255 Engine Parameters

Table A-18 lists parameters specific to the Service H225 engine.

Table A-18   Service H.225 Engine Parameters

Parameter	Description	Value
message-type	Type of H225 message to which the signature applies: • SETUP • ASN.1-PER • Q.931 • TPKT	asn.1-per q.931 setup tpkt
policy-type	Type of H225 policy to which the signature applies: • Inspects field length. • Inspects presence. If certain fields are present in the message, an alert is sent. • Inspects regular expressions. • Inspects field validations. • Inspects values. Regex and presence are not valid for TPKT signatures.	length presence regex validate 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. • field-name—Field name to inspect.	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. • invalid-packet-index—Inspection for invalid packet index.	0 to 255

Table A-18   Service H.225 Engine Parameters (continued)

Parameter	Description	Value
specify-regex-string	The regular expression to look for when the policy type is regex. This is never set for TPKT signatures: • A regular expression to search for in a single TCP packet • (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.	regex-string specify-min-match-length
specify-value-range	Valid for the length or value policy types (0x00 to 6535). Not valid for other policy types. • value-range—Range of values.	0 to 65535[1] a-b

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

For More Information

For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

Service HTTP Engine

This section describes the Service HTTP engine, and contains the following topics:

Understanding the 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 in to 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.

Service HTTP Engine Parameters

Table A-19 lists the parameters specific the Service HTTP engine.

Table A-19   Service HTTP Engine Parameters

Parameter	Description	Value
de-obfuscate	Applies anti-evasive deobfuscation before searching.	true | false
max-field-sizes	Maximum field sizes grouping.	—
specify-max-arg-field-length	(Optional) Enables maximum argument field length: • max-arg-field-length—Maximum length of the arguments field.	0 to 65535
specify-max-header-field-length	(Optional) Enables maximum header field length: • max-header-field-length—Maximum length of the header field.	0 to 65535
specify-max-request-length	(Optional) Enables maximum request field length: • max-request-length—Maximum length of the request field.	0 to 65535
specify-max-uri-field-length	(Optional) Enables the maximum URI field length: • max-uri-field-length—Maximum length of the URI field.	0 to 65535
regex	Regular expression grouping.	—
specify-arg-name-regex	(Optional) Enables searching the Arguments field for a specific regular expression: • arg-name-regex—Regular expression to search for in the HTTP Arguments field (after the ? and in the Entity body as defined by Content-Length).	—
specify-header-regex	(Optional) Enables searching the Header field for a specific regular expression: • header-regex—Regular Expression to search in the HTTP Header field. The Header is defined after the first CRLF and continues until CRLFCRLF.	—
specify-request-regex	(Optional) Enables searching the Request field for a specific regular expression: • request-regex—Regular expression to search in both HTTP URI and HTTP Argument fields. • specify-min-request-match-length—Enables setting a minimum request match length.	0 to 65535

Table A-19   Service HTTP Engine Parameters (continued)

Parameter	Description	Value
specify-uri-regex	(Optional) Regular expression to search in HTTP URI field. The URI field is defined to be after the HTTP method (GET, for example) and before the first CRLF. The regular expression is protected, which means you cannot change the value.	[/\\][a-zA-Z][a-zA-Z][a-zA-Z][a-zA-Z][a-zA-Z][a-zA-Z][a-zA-Z][.]jpeg
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
swap-attacker-victim	True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).	true | false

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

For More Information

- For an example Service HTTP custom signature, see Example Service HTTP Signature, page 5-52.

- For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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 A-20 lists the parameters specific to the Service IDENT engine.

Table A-20   Service IDENT Engine Parameters

Parameter	Description	Value
inspection-type	Type of inspection to perform.	—
has-bad-port	Inspects payload for a bad port.	true | false
has-newline	Inspects payload for a nonterminating new line character.	true | false
size	Inspects for payload length longer than this.	0 to 65535
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
direction	Direction of the traffic: - Traffic from service port destined to client port. - Traffic from client port destined to service port.	from-service to-service

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

Service MSRPC Engine

This section describes the Service MSRPC engine, and contains the following topics:

- Understanding the Service MSRPC Engine, page A-32
- Service MSRPC Engine Parameters, page A-32

Understanding the 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.

Service MSRPC Engine Parameters

Table A-21 lists the parameters specific to the Service MSRPC engine.

Table A-21   Service MSRPC Engine Parameters

Parameter	Description	Value
protocol	Protocol of interest for this inspector.	tcp udp
specify-operation	(Optional) Enables using MSRPC operation: • operation—MSRPC operation requested. Required for SMB_COM_TRANSACTION commands. Exact match.	0 to 65535

specify-regex-string	(Optional) Enables using a regular expression string:
	
	• specify-exact-match-offset—Enables the exact match offset:
	
	  – exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.
	
	• specify-min-match-length—Enables the minimum match length:
	
	  – min-match-length—Minimum number of bytes the regular expression string must match.
		0 to 65535

specify-uuid	(Optional) Enables UUID: • uuid—MSRPC UUID field.	000001a0000 00000c00000 0000000046

For More Information

For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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 A-22 lists the parameters specific to the Service MSSQL engine.

Table A-22   Service MSSQL Engine Parameters

Parameter	Description	Value
password-present	Whether or not a password was used in an MS SQL login.	true | false
specify-sql-username	(Optional) Enables using an SQL username: • sql-username—Username (exact match) of user logging in to MS SQL service.	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 A-23 lists the parameters specific to the Service NTP engine.

Table A-23   Service NTP Engine Parameters

Parameter	Description	Value
inspection-type	Type of inspection to perform.	
inspect-ntp-packets	Inspects NTP packets: • control-opcode—Opcode number of an NTP control packet according to RFC1305, Appendix B. • max-control-data-size—Maximum allowed amount of data sent in a control packet. • mode —Mode of operation of the NTP packet per RFC 1305.	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.	true | false
is-non-ntp-traffic	Checks for nonNTP packets on an NTP port.	true | false

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 A-24 lists the parameters specific to the Service RPC engine.

Table A-24   Service RPC Engine Parameters

Parameter	Description	Value
direction	Direction of traffic: • Traffic from service port destined to client port. • Traffic from client port destined to service port.	from-service to-service
protocol	Protocol of interest.	tcp udp
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
specify-is-spoof-src	(Optional) Enables the spoof source address: • is-spoof-src—Fires an alert when the source address is 127.0.0.1.	true | false
specify-port-map-program	(Optional) Enables the portmapper program: • port-map-program—The program number sent to the portmapper for this signature.	0 to 9999999999
specify-rpc-max-length	(Optional) Enables RPC maximum length: • rpc-max-length—Maximum allowed length of the entire RPC message. Lengths longer than what you specify fire an alert.	0 to 65535
specify-rpc-procedure	(Optional) Enables RPC procedure: • rpc-procedure—RPC procedure number for this signature.	0 to 1000000
specify-rpc-program	(Optional) Enables RPC program: • rpc-program—RPC program number for this signature.	0 to 1000000

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

Service SMB Engine

The Service SMB engine inspects SMB packets. You can tune SMB signatures and create custom SMB signatures based on SMB control transaction exchanges and SMB NT_Create_AndX exchanges.

Table A-25 lists the parameters specific to the Service SMB engine.

Table A-25   Service SMB Engine Parameters

Parameter	Description	Value
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535 a-b[,c-d] [1]
specify-allocation-hint	(Optional) Enables MSRPC allocation hint: • allocation-hint—MSRPC Allocation Hint, which is used in SMB_COM_TRANSACTION command parsing.[2]	0 to 42949677295
specify-byte-count	(Optional) Enables byte count: • byte-count—Byte count from SMB_COM_TRANSACTION structure.[3]	0 to 65535
specify-command	(Optional) Enables SMB commands: • command—SMB command value.[4]	0 to 255
specify-direction	(Optional) Enables traffic direction: • direction—Lets you specify the direction of traffic: – Traffic from service port destined to client port. – Traffic from client port destined to service port.	from service to service
specify-file-id	(Optional) Enables using a transaction file ID: • file-id—Transaction File ID.[5] **Note** This parameter may limit a signature to a specific exploit instance and its use should be carefully considered.	0 to 65535
specify-function	(Optional) Enables named pipe function: • function—Named Pipe function.[6]	0 to 65535
specify-hit-count	(Optional) Enables hit counting: • hit-count—The threshold number of occurrences in scan-interval to fire alerts.[7]	0 to 65535
specify-operation	(Optional) Enables MSRPC operation: • operation—MSRPC operation requested. Required for SMB_COM_TRANSACTION commands. An exact match is required.	0 to 65535

Table A-25   Service SMB Engine Parameters (continued)

Parameter	Description	Value
specify-resource	(Optional) Enables resource: • resource—Specifies that pipe or the SMB filename is used to qualify the alert. In ASCII format. An exact match is required.	*resource*
specify-scan-interval	(Optional) Enables scan interval: • scan-interval—The interval in seconds used to calculate alert rates.[8]	0 to 131071
specify-set-count	(Optional) Enables counting setup words: • set-count—Number of Setup words.[9]	0 to 255
specify-type	(Optional) Enables searching for the Type field of an MSRPC packet: • type —Type Field of MSRPC packet. 0 = Request; 2 = Response; 11 = Bind; 12 = Bind Ack	0 to 255
specify-word-count	(Optional) Enables word counting for command parameters: • word-count—Word count for the SMB_COM_TRANSACTION command parameters.[10]	0 to 255
swap-attacker-victim	True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).	true	false

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

2. An exact match is optional.

3. An exact match is optional.

4. An exact match is required. Currently supporting the 37 (0x25) SMB_COM_TRANSACTION command \x26amp and the 162 (0xA2) SMB_COM_NT_CREATE_ANDX command.

5. An exact match is optional.

6. An exact match is required. Required for SMB_COM_TRANSACTION commands.

7. Valid for signatures 3302 and 6255 only.

8. Valid for signatures 3302 and 6255 only.

9. An exact match is required. Usually two are required for SMB_COM_TRANSACTION commands.

10. An exact match is required. Only 16 word transactions are decoded.

Service SMB Advanced 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.

Note   The Service SMB Advanced engine replaces the Service SMB engine.

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

Table A-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 a-b[,c-d] [1]
specify-command	(Optional) Enables SMB commands: • command—SMB command value; exact match required; defines the SMB packet type.[2]	0 to 255

specify-direction	(Optional) Enables traffic direction:
	• direction—Lets you specify the direction of traffic:
	  – from-service—Traffic from service port destined to client port.
	  – to-service—Traffic from client port destined to service port.
		from service to service

specify-operation	(Optional) Enables MSRPC over SMB: • msrpc-over-smb-operation—Required for SMB_COM_TRANSACTION commands, exact match required.	0 to 65535
specify-regex-string	(Optional) Enables searching for regex strings: • regex-string—A regular expression to search for in a single TCP packet.	
specify-exact-match-offset	(Optional) Enables exact match offset: • exact-match-offset—The exact stream offset the Regex string must report a match to be valid.	
specify-min-match-length	(Optional) Enables minimum match length: • min-match-length—Minimum number of bytes the Regex string must match.	
specify-payload-source	(Optional) Enables payload source: • payload-source—Payload source inspection.[3]	
specify-scan-interval	(Optional) Enables scan interval: • scan-interval—The interval in seconds used to calculate alert rates.	1 to 131071

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

Parameter	Description	Value
specify-tcp-flags	(Optional) Enables TCP flags: • msrpc-tcp-flags • msrpc-tcp-flags-mask	• concurrent execution • did not execute • first fragment • last fragment • maybe • object UUID • pending cancel • reserved
specify-type	(Optional) Enables type of MSRPC over SMB packet: • type—Type field of MSRPC over SMB packet	• 0 = Request • 2 = Response • 11 = Bind • 12 = Bind Ack
specify-uuid	(Optional) Enables MSRPC over UUID: • uuid—MSRPC UUID field	32-character string composed of hexadecimal characters 0-9, a-f, A-F.
specify-hit-count	(Optional) Enables hit counting: • hit-count—The threshold number of occurrences in scan-interval to fire alerts.	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

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

2. Currently supporting 37 (0x25) SMB_COM_TRANSACTION command \x26amp; 162 (0xA2) SMB_COM_NT_CREATE_ANDX command.

3. TCP_Data performs regex over entire packet, SMB_Data performs regex on SMB payload only, Resource_DATA performs regex on SMB_Resource.

For More Information

For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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 A-27 lists the parameters specific to the Service SNMP engine.

Table A-27   Service SNMP Engine Parameters

Parameter	Description	Value
inspection-type	Type of inspection to perform.	—
brute-force-inspection	Inspects for brute force attempts: • brute-force-count—The number of unique SNMP community names that constitute a brute force attempt.	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:
	• specify-community-name [yes | no]:
	  – community-name—Searches for the SNMP community name, that is, the SNMP password.
	• specify-object-id [yes | no]:
	  – object-id—Searches for the SNMP object identifier.
		community-name
		
		object-id

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 A-28 lists the parameters specific to the Service SSH engine.

Table A-28   Service SSH Engine Parameters

Parameter	Description	Value

length-type	Inspects for one of the following SSH length types:
	• key-length—Length of the SSH key to inspect for:
	  – length—Keys larger than this fire the RSAREF overflow.
	• user-length—User length SSH inspection:
	  – length—Keys larger than this fire the RSAREF overflow.
		0 to 65535

service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
specify-packet-depth	(Optional) Enables packet depth: • packet-depth—Number of packets to watch before determining the session key was missed.	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 A-29 lists the parameters specific to the Service TNS engine.

Table A-29   Service TNS Engine Parameters

Parameter	Description	Value
type	Specifies the TNS frame value type: • 1—Connect • 2—Accept • 4—Refuse • 5—Redirect • 6—Data • 11—Resend • 12—Marker	1 2 4 5 6 11 12

specify-regex-string	(Optional) Enables using a regular expression string:
	• specify-exact-match-offset—Enables the exact match offset:
	  – exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.
	• specify-min-match-length—Enables the minimum match length:
	  – min-match-length—Minimum number of bytes the regular expression string must match.
		0 to 65535

specify-regex-payload	Specifies which protocol to inspect: • TCP data—Performs Regex over the data portion of the TCP packet. • TNS data—Performs Regex only over the TNS data (with all white space removed).	TCP TNS

For More Information

For a list of the signature regular expression syntax, see Regular Expression Syntax, page A-8.

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 A-30 lists the parameters specific to the State engine.

Table A-30   State Engine Parameters

Parameter	Description	Value
state-machine	State machine grouping.	—
cisco-login	Specifies the state machine for Cisco login: • state-name—Name of the state required before the signature fires an alert: – Cisco device state – Control-C state – Password prompt state – Start state	cisco-device control-c pass-prompt start
lpr-format-string	Specifies the state machine to inspect for the LPR format string vulnerability: • state-name—Name of the state required before the signature fires an alert: – Abort state to end LPR Format String inspection – Format character state – State state	abort format-char start
smtp	Specifies the state machine for the SMTP protocol: • state-name—Name of the state required before the signature fires an alert: – Abort state to end LPR Format String inspection – Mail body state – Mail header state – SMTP commands state – Start state	abort mail-body mail-header smtp-commands start
direction	Direction of the traffic: • Traffic from service port destined to client port. • Traffic from client port destined to service port.	from-service to-service
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535 a-b[,c-d] [1]

Table A-30   State Engine Parameters (continued)

Parameter	Description	Value
specify-exact-match-offset	(Optional) Enables exact match offset: • exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.	0 to 65535
specify-min-match-length	(Optional) Enables minimum match length: • min-match-length—Minimum number of bytes the regular expression string must match.	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).	true | false

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 the 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 in to 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 A-31 lists the parameters specific to the String ICMP engine.

Table A-31   String ICMP Engine Parameters

Parameter	Description	Value
direction	Direction of the traffic: • Traffic from service port destined to client port. • Traffic from client port destined to service port.	from-service to-service
icmp-type	ICMP header TYPE value.	0 to 18[1] a-b[,c-d]

Table A-31   String ICMP Engine Parameters (continued)

Parameter	Description	Value
specify-exact-match-offset	(Optional) Enables exact match offset: • exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.	0 to 65535
specify-min-match-length	(Optional) Enables minimum match length: • min-match-length—Minimum number of bytes the regular expression string must match.	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).	true | false

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 .

String TCP Engine Parameters

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

Table A-32   String TCP Engine

Parameter	Description	Value
direction	Direction of the traffic: • Traffic from service port destined to client port. • Traffic from client port destined to service port.	from-service to-service
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
specify-exact-match-offset	(Optional) Enables exact match offset: • exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.	0 to 65535
specify-min-match-length	(Optional) Enables minimum match length: • min-match-length—Minimum number of bytes the regular expression string must match.	0 to 65535
strip-telnet-options	Strips the Telnet option characters from the data before the pattern is searched.[2]	true | false
swap-attacker-victim	True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).	true | false

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

2. 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 5-50.

String UDP Engine Parameters

Table A-33 lists the parameters specific to the String UDP engine.

Table A-33   String UDP Engine

Parameter	Description	Value
direction	Direction of the traffic: • Traffic from service port destined to client port. • Traffic from client port destined to service port.	from-service to-service
service-ports	A comma-separated list of ports or port ranges where the target service resides.	0 to 65535[1] a-b[,c-d]
specify-exact-match-offset	(Optional) Enables exact match offset: • exact-match-offset—The exact stream offset the regular expression string must report for a match to be valid.	0 to 65535
specify-min-match-length	(Optional) Enables minimum match length: • min-match-length—Minimum number of bytes the regular expression string must match.	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).	true | false

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 5-50.

Sweep Engines

This section describes the Sweep engines and their parameters. It contains the following topics:

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.

You can configure source and destination address filters, which means the sweep signature will exclude these addresses from the sweep-counting algorithm.

Caution Event action filters based on source and destination IP addresses do not function for the Sweep engine, because they do not filter as regular signatures. To filter source and destination IP addresses in sweep alerts, use the source and destination IP address filter parameters in the Sweep engine signatures.

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.

Data Node

When an activity related to Sweep engine signatures is seen, the IPS uses a Data Node to determine when it should stop monitoring for a particular host. The Data Node contains various persistent counters and variables needed for cross-packet reassembly of streams and for tracking the inspection state on a per-stream/per-source/per-destination basis The Data Node containing the sweep determines when the sweep should expire. The Data Node stops a sweep when the Data Node has not seen any traffic for *x* number of seconds (depending on the protocol).

There are several adaptive timeouts for the Data Nodes. The Data Node expires after 30 seconds of idle time on the address set after all of the contained objects have been removed. Each contained object has various timeouts, for example, TCP Stream has a one-hour timeout for established connections. Most other objects have a much shorter expiration time, such as 5 or 60 seconds.

Table A-34 lists the parameters specific to the Sweep engine.

Table A-34   Sweep Engine Parameters

Parameter	Description	Value
dst-addr-filter	Destination IP address to exclude from the sweep counting algorithm.	<A.B.C.D>-<A.B.C.D> [,<A.B.C.D>-<A.B.C.D>]
src-addr-filter	Source IP address to exclude from the sweep counting algorithm.	<A.B.C.D>-<A.B.C.D> [,<A.B.C.D>-<A.B.C.D>]
protocol	Protocol of interest for this inspector.	icmp udp tcp
specify-icmp-type	(Optional) Enables the ICMP header type: • icmp-type—ICMP header TYPE value.	0 to 255

Table A-34   Sweep Engine Parameters (continued)

Parameter	Description	Value
specify-port-range	(Optional) Enables using a port range for inspection: • port-range—UDP port range used in inspection.	0 to 65535 a-b[,c-d]
fragment-status	Specifies whether fragments are wanted or not: • Any fragment status. • Do not inspect fragments. • Inspect fragments.	any no-fragments want-fragments
inverted-sweep	Uses source port instead of destination port for unique counting.	true | false
mask	Mask used in TCP flags comparison: • URG bit • ACK bit • PSH bit • RST bit • SYN bit • FIN bit	urg ack psh rst syn fin
storage-key	Type of address key used to store persistent data: • Attacker address • Attacker and victim addresses • Attacker address and victim port	Axxx AxBx Axxb
suppress-reverse	Does not fire when a sweep has fired in the reverse direction on this address set.	true | false
swap-attacker-victim	True if address (and ports) source and destination are swapped in the alert message. False for no swap (default).	true | false
tcp-flags	TCP flags to match when masked by mask: • URG bit • ACK bit • PSH bit • RST bit • SYN bit • FIN bit	urg ack psh rst syn fin
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 A-35 lists the parameters specific to the Sweep Other TCP engine.

Table A-35   Sweep Other TCP Engine Parameters

Parameter	Description	Value
specify-port-range	(Optional) Enables using a port range for inspection: • port-range—UDP port range used in inspection.	0 to 65535 a-b[,c-d]
set-tcp-flags	Lets you set TCP flags to match: • tcp-flags—TCP flags used in this inspection: – URG bit – ACK bit – PSH bit – RST bit – SYN bit – FIN bit	urg ack psh rst syn fin

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 A-36 lists the anomaly detection worm signatures.

Table A-36   Anomaly Detection Worm Signatures

Signature ID	Subsignature ID	Name	Description
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.

Table A-36   Anomaly Detection Worm Signatures (continued)

Signature ID	Subsignature ID	Name	Description
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 illegal zone.
13006	1	Illegal TCP Scanner	Identified a worm attack over a TCP protocol in the illegal zone; the TCP histogram threshold was crossed and a scanner over a TCP protocol was identified.
13007	0	Illegal UDP Scanner	Identified a single scanner over a UDP protocol in the illegal zone.
13007	1	Illegal UDP Scanner	Identified a worm attack over a UDP protocol in the illegal 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 illegal zone.
13008	1	Illegal Other Scanner	Identified a worm attack over an Other protocol in the illegal 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 A-37 lists the parameters specific to the Traffic ICMP engine.

Table A-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: • Inspects for original LOKI traffic. • Inspects for modified LOKI traffic.	is-loki is-mod-loki
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.

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