



Working With Signature Engines

This appendix describes signature engines and their parameters. See [Configuring Signatures, page 3-1](#), for information on tuning and creating signatures.

This appendix contains the following sections:

- [General Information, page A-2](#)
- [ATOMIC Engines, page A-17](#)
- [FLOOD Engines, page A-22](#)
- [HTTP Deobfuscation, page A-26](#)
- [OTHER Engine, page A-30](#)
- [SERVICE Engines, page A-32](#)
- [STATE.STRING Engines, page A-50](#)
- [STRING Engines, page A-56](#)
- [SWEEP Engines, page A-59](#)
- [SYSLOG Engine, page A-64](#)
- [TROJAN and TRAFFIC Engines, page A-65](#)

General Information

The following sections contain general information about signature engines:

- [Signature Engines, page A-2](#)
- [Alarm Handling, page A-4](#)
- [Configuration Parsing, page A-5](#)
- [MASTER Engine Configuration Restrictions, page A-5](#)
- [Engine Summarizer, page A-6](#)
- [MASTER Engine Parameters, page A-9](#)
- [Regular Expression Syntax, page A-13](#)
- [Error Messages, page A-15](#)

Signature Engines

A signature engine is a component of the sensor and supports many signatures in a certain category. An engine is composed of a parser and an inspector. Each engine has a set of legal parameters that have allowable ranges or sets of values.

[Table A-1](#) lists the signature engines and their descriptions.

Table A-1 Signature Engines

Engine	Description
ATOMIC.ARP	ARP simple and cross-packet signatures.
ATOMIC.ICMP	Simple ICMP alarms based on the following parameters: Type, Code, Sequence, and ID.
ATOMIC.IPOPTIONS	Simple alarms based on the decoding of layer 3 options.
ATOMIC.L3.IP	Simple layer 3 IP alarms.

Table A-1 Signature Engines (continued)

Engine	Description
ATOMIC.TCP	Simple TCP packet alarms based on the following parameters: Port, Destination, Flags, and single packet Regex. Use SummaryKey to define the address view for MinHits and Summarize counting. For best performance, use a StorageKey of xxxx.
ATOMIC.UDP	Simple UDP packet alarms based on the following parameters: Port, Direction, and DataLength.
FLOOD.HOST.ICMP	ICMP floods directed at a single host.
FLOOD.HOST.UDP	UDP floods directed at a single host.
FLOOD.NET	Multi-protocol floods directed at a network segment. IP addresses are wildcarded for this inspection.
OTHER	This engine is used to group generic signatures so common parameters can be changed. It defines an interface into common signature parameters.
SERVICE.DNS	Analyzes the DNS service.
SERVICE.FTP	FTP service special decode alarms.
SERVICE.GENERIC	Custom service/payload decode. For expert use only.
SERVICE.HTTP	HTTP protocol decode-based string engine. Includes anti-evasive URL deobfuscation.
SERVICE.IDENT	IDENT service (client and server) alarms.
SERVICE.MSSQL	Microsoft SQL service inspection engine.
SERVICE.NTP	Network Time Protocol-based signature engine.
SERVICE.RPC	Analyzes the RPC service.
SERVICE.SMB	SMB SuperInspector signatures.
SERVICE.SMTP	Inspects SMTP protocol.

Table A-1 Signature Engines (continued)

Engine	Description
SERVICE.SNMP	Inspects SNMP traffic.
SERVICE.SSH	SSH header decode signatures.
SERVICE.SYSLOG	Processes syslogs.
STATE.STRING.CISCOLOGIN	Telnet-based Cisco Login inspection engine.
STATE.STRING.LPRFORMAT STRING	Inspects LPR protocol.
STRING.ICMP	Generic ICMP-based string search engine.
STRING.TCP	Generic TCP-based string search engine.
STRING.UDP	Generic UDP-based string search engine.
SWEEP.HOST.ICMP	A single host sweeping a range of nodes using ICMP.
SWEEP.HOST.TCP	Detects host and service sweeps over TCP.
SWEEP.MULTI	Conducts cross-protocol sweeps.
SWEEP.OTHER.TCP	Conducts fingerprint scans.
SWEEP.PORT.TCP	Detects port sweeps between two nodes.
SWEEP.PORT.UDP	Detects UDP connections to multiple destination ports between two nodes.

Alarm Handling

The alarm output unit sends alarms to the EventStore. There are many attributes of the events in the EventStore. Some attributes are always present, such as eventID, severity, the originator block, and the time block. Other attributes are optional, such as the participant block, which can contain multiple addresses or ports for a single alarm.

The alertDetails field is optional and is used extensively for extra string data associated with the alarm.

The context block is used for relaying the packet(s) that caused the alarm. The STRING, STATE, and HTTP engines typically only use the context block.

The alarm does not normally have multiple addresses or ports in the participant block. In special cases, such as in SWEEPS, multiple addresses or ports are included.

Configuration Parsing

A setConfig control transaction (CT) is received along with the new configuration in XML data format. The engines parse and interpret the data, checking for violations of constraints, such as missing required parameters, parameter values out of range, or mutually exclusive parameters. If an error is encountered, the setConfig CT is rejected (with an error code) and an error string is returned to the caller with details about the faulty parameters.

MASTER Engine Configuration Restrictions

The following configuration restrictions apply to the MASTER engine parameters:

- Not all MASTER parameters are interpreted by all subengines.



Note The parameters most commonly not used by all subengines are WantFrag, MaxInspectLength, and ResetAfterIdle.

- If you use AlarmInterval, you must set the following parameters:
 - MinHits must be less than 1
 - AlarmThrottle to FireAll
 - ChokeThreshold to ANY
 - Configure a valid ThrottleInterval
- You cannot use MinHits with AlarmThrottle FireOnce.
- You cannot use ChokeThreshold with AlarmThrottle FireOnce.
- A ChokeThreshold does not make sense when the AlarmThrottle is GlobalSummary.

Some signature engines do not support custom signatures. These engines are specific to the signatures, not general to the protocol decode like the other engines.

You cannot create custom signatures based on these engines:

- SERVICE.SMB
- SERVICE.SSH
- TRAFFIC.ICMP
- TROJAN.BO2K
- TROJAN.TFN2K
- TROJAN.UDP

If you try to create custom signatures for these engines, you receive the following error message: `Error: Array contains max entries, could not add new entry.`

Engine Summarizer

The Summarizer decreases the volume of alarms sent out from the sensor and provides basic aggregation of events into a single alarm. Special parameters are specified for each signature and they influence the handling of the alarms. Each signature has a default set of settings that demonstrate preferred normal behavior. You can modify each signature to change the default behavior within the constraints for each engine type. The summary feature operates in Fixed (no auto-upgrades of summary mode) or Variable mode (auto-upgrades and downgrades are possible).

The Variable mode counters anti-IDS tools, such as “stick,” which are designed to send bogus traffic so that the IDS produces thousands of alarms in a very short time. This can be an IDS console denial-of-service (DoS), because the alarm display can be cluttered with so many alarms the operator is forced to delete the alarms, some of which may be legitimate ones in the cluttered mess of bogus alarms.

The Variable mode watches for spikes in alarm volume (on a per-signature basis) and suppresses alarms after a certain point in the interval. At the end of the interval, a summary alarm is sent and normal operation is resumed. Upgrades and

Downgrades of mode refer to the automatic changing of the handling of the alarm suppression and summarization for that signature on the address set in the time period (interval).

Fixed mode operation does not attempt to suppress alarms any further than is specified in the parameters for the signature.

The basic aggregation features include two concepts. The simpler mode is when a signature has a threshold number of hits that must be seen before the alarm is sent. A more advanced mode is the timed-interval counting. This watches for X hits in the past Y seconds and only sends an alarm when that condition is met. Here, a “hit” is a term used to describe an event, which is basically an alarm, but it will not be sent out of the sensor as an alarm until the threshold number of hits has been exceeded.

The following parameters influence the behavior of signature alarms:

- **ThrottleInterval** specifies the time (in seconds) interval used for the counting algorithms.
- **MinHits** specifies the alarm limiter based on count. **MinHits** can be used with or without **AlarmInterval** to get different behavior. If the **AlarmInterval** is not specified, it is a raw count across the life of the inspector and does not reset after a time out. Use this feature when you want to count **MinHits** but do not care about the elapsed time.
- **AlarmInterval** is used to enable Timed counting. **AlarmInterval** continuously counts the alarms based on the activity seen in the last Y seconds and only sends an alarm when there have been X hits in the past Y seconds. Here, the X is **MinHits** and the Y is **ThrottleInterval**. Use this feature when you want to limit the alarms with a time period. A typical use of the **AlarmInterval** is for the login failure signature. After 3 failed logins in the last 60 seconds, an alarm is sent. If you do not use the elapsed time discriminator, the signature eventually fires, maybe an hour later. However, that is not indicative of a serious login failure because login failures are a common event, but are more serious (such as brute force password attacks) when there are rapid failures.
- **AlarmThrottle** defines which summarization mode to use.
 - **FireAll** does not limit the alarms being sent unless the automatic upgrade kicks in from a **ChokeThreshold** setting.
 - **FireOnce** fires only one alarm across the life of the inspector (on the address set).

- Summarize fires the first alarm in the interval and then begins counting. At the end of the interval, if there was more than 1 event, a summary alarm is sent. The summary alarm matches the original one that was sent with one change—the alertDetails field has a string like the following: “Interval Summary: X alarms this interval.” Summarize alarms are counted on the address set specified by SummaryKey (or its default value).
- GlobalSummarize is similar to Summarize except that the alertDetails field has a string like the following: “Global Summary: X alarms this interval,” and the address set used for counting is the Global (xxxx) key. This means that all the same SIGID alarms on the sensor are counted in one spot so the number reported in the Global Summary string is a sensor-wide count.
- ChokeThreshold enables or disables the automatic upgrade/downgrade feature. This feature is turned off (disabled) by leaving ChokeThreshold blank if ChokeThreshold does not have a default value or by setting ChokeThreshold to a large value such as 100,000. A numeric value for ChokeThreshold denotes the threshold number of alarms in the interval to trigger an auto-upgrade. The upgrade sequence is FireAll to Summarize to GlobalSummarize.

It goes straight from FireAll to GlobalSummarize if 2 times the ChokeThreshold value is exceeded. Downgrades occur at the start of the next interval. It reverts to its normal behavior starting at the next interval (right after the summary alarm is sent).

Example: SIG 2000 AlarmThrottle FireAll ChokeThreshold 100
ThrottleInterval 60

Traffic1: 90 alarms in 60 seconds. Result: You get 90 regular alarms.

Traffic2: 120 alarms in 60 seconds. Result: You get 100 regular alarms and 1 IntervalSummary alarm with count 120.

Traffic3: 240 alarms in 60 seconds. Result: You get 100 regular alarms and 1 GlobalSummary alarm with count 240.



Note This example assumes that all alarms are on the same address set, for example, from 10.1.1.1 to 10.2.2.2.

- SummaryKey specifies which address view to use for the counting algorithm. If a SummaryKey is not specified, the algorithm tries to use the same key as the one set in StorageKey. A StorageKey of xxxx is not valid for SummaryKey, so you must specify one for these signatures. The SummaryKey is the address set where the counters for the algorithms live, so you receive different results (depending on traffic variance) for values of Axxx (source address only), xxBx (destination address only), and AxBx (the pair of source and destination addresses).

Not all the parameters work together. There are several constraints on the modes:

- You cannot use AlarmThrottle FireOnce with ChokeThreshold X (where X is not ANY.)
- You cannot use AlarmThrottle FireOnce with signatures that use StorageKey xxxx.
- If AlarmInterval is specified, MinHits must be greater than 1, AlarmThrottle must be FireAll, ChokeThreshold must be ANY.
- You cannot set a SummaryKey with ports (AaBb, Axxb) where the protocol of the inspector does not have ports.
- SummaryKey tries to default to the same as StorageKey except when StorageKey is xxxx. In this case, you must either specify a SummaryKey or the summarization features will not work (you will get all alarms).

MASTER Engine Parameters

The MASTER engine supplies parameters to each subengine. It is the central point for input (parsing the configuration) and output (sending the alarms). Each subengine provides the specific protocol that is needed to decode and inspect the input. The MASTER engine structures and methods are inherited by the subengines.

Table A-2 lists and describes the MASTER engine parameters. A parameter is a name-value pair. The name is defined by each engine. The value has limiters that are defined by the engine so that only values falling in a particular range or out of a set of choices are valid. The names are constant across all signatures for an engine. The values are variable entities that house the differences between signatures in an engine group.

**Note**

The Data Type can have a limiter or valid value.

If a parameter is protected, you cannot change it for the default signatures. You can modify it for custom signatures.

If a parameter is required, you must define it for all signatures—both default signatures and custom signatures.

Table A-2 MASTER Engine Parameters

Parameter Name	Data Type	Description
AlarmDelayTimer	NUMBER (1–3600)	The number of seconds to delay further signature inspection after an alarm.
AlarmInterval	NUMBER (2–1000)	Special handling for time events. Use AlarmInterval <i>Y</i> with MinHits <i>X</i> for <i>X</i> alarms in a <i>Y</i> -second interval.
AlarmSeverity	ENUM (high, medium, low, informational)	The severity of this alert reported in the alarm.
AlarmThrottle	ENUM (FireAll, FireOnce, GlobalSummarize, Summarize)	Technique used to limit alarm firings with the following options: <ul style="list-style-type: none"> • FireAll—Sends all alarms. • FireOnce—Sends the first alarm and then deletes the inspector. • Summarize—Sends an IntervalSummary alarm. • GlobalSummarize—Sends a GlobalSummary alarm.
AlarmTraits	NUMBER (0–65535)	User-defined traits that further describe this signature.

Table A-2 MASTER Engine Parameters (continued)

Parameter Name	Data Type	Description
ChokeThreshold	NUMBER	Threshold value of alarms per interval to autoswitch AlarmThrottle modes. If ChokeThreshold is defined, the sensor switches AlarmThrottle modes when a large volume of alarms is seen in the ThrottleInterval.
Enabled	BOOLEAN (true, false)	True to enable the signature; false to disable the signature.
EventAction	BITSET (log, reset, shunHost, shunConnection, ZERO)	The action to perform when the alarm is fired.
FlipAddr	BOOLEAN (true, false)	True if address (and ports) Source and Destination are swapped in the alarm message; false if they are not swapped (normal)
MaxInspectLength	NUMBER	The maximum number of bytes to inspect.
MaxTTL	NUMBER (0–1000)	The maximum number of seconds to inspect a logical stream. The inspector is deleted after X seconds of being active.
MinHits	NUMBER	The minimum number of signature hits before the alarm message is sent. This is a limiter for firing the alarm only after X times of seeing the signature on the address key.
Protocol	BITSET (FRAG, IP, TCP, UDP, ICMP, ARP, CROSS, ZERO CUSTOM)	Protocol of interest for this inspector.

Table A-2 MASTER Engine Parameters (continued)

Parameter Name	Data Type	Description
ResetAfterIdle	NUMBER (2–1000)	Number of seconds to wait to reset signature counters after the host(s) were idle.
ServicePorts	NUMBER	A comma-separated list of ports or port ranges where the target service may reside.
SigComment	STRING	USER NOTES—Miscellaneous information about this signature.
SIGID	NUMBER (993–50000)	Signature identifier. The range 993–19999 is valid for default signatures. The range 20000–50000 is valid for custom signatures.
SigName	STRING	Official name of the signature.
SigStringInfo	STRING	Extra information included in the alarm message.
StorageKey	BITSET (xxxx Axxx xxBx AxBx AaBb Axxb STREAM DOUBLE ZERO)	Type of Address Key used to store persistent data.
SubSig	NUMBER	Subsignature ID. Denotes a specific variant of a signature.
SummaryKey	ENUM (AaBb AxBx Axxb Axxx xxBx)	The Storage Type on which to summarize this signature.
ThrottleInterval	NUMBER (0–1000)	Number of seconds defining an AlarmThrottle interval. This is used with the AlarmThrottle parameter to tune special alarm limiters.
WantFrag	ENUM (ANY, false, true)	True if a fragment is desired; false if a fragment is not desired; ANY for either.

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 IDS version 4.0 Regex syntax.

Table A-3 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.
{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).

Table A-3 Regular Expression Syntax (continued)

Metacharacter	Name	Description
\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.

[Table A-4](#) 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
foo and bar on the same line with anything except a new line between them	foo.*bar
Either foo or bar	foolbar
Either moon or soon	(m s)oon

Error Messages

You may come across the following error messages when working with signature engines.

The following errors may occur in the service submode commands. They are located in source

```
/vob/csids_2/dev/srcPool/cli/cidCliStateTransition.cpp.
```

- Error: Attempt to remove an entry that does not exist. Operation failed.

This error occurs if you try to delete an entry that does not exist. For example, in virtual sensor configuration, **ATOMIC.ARP: no signatures sigid 7106**

- Error: Attempt to delete permanent entry. Operation failed.

This error occurs if you try to delete a protected entry. For example, in **ATOMIC.ARP: no signatures sigid 7105**

- Error: Array contains min entries, could not delete requested entry.

This error occurs when you try to remove an entry from an array that needs a minimum number of entries.

- Error: Cannot create a custom signature with SIGID < 20000.
- Error: Array contains max entries, could not add new entry as stated in original email thread.
- Error: <XML Config Name Str> failed validation. Would you like to exit anyway and discard your changes?

Where <XML Config Name Str> is replaced with a colon-separated string indicating the invalid configuration. For example, in the service host configuration:

```
sensor(config-Host)# timeParams  
sensor(config-Host-tim)# summerTimeParams  
sensor(config-Host-tim-sum)# active-selection recurringParams  
sensor(config-Host-tim-sum)# exit
```

```
Error: 'summerTimeParams': 'recurringParams': 'summerTimeZoneName'  
- there is neither a value nor a 'default' attribute for a required  
failed validation. Would you like to exit anyway?[no]
```

The following set of errors comes from the IDAPI interface and are located in `/vob/csids_2/dev/srcPool/lib/idapi/cidCtlTrans.cpp`. You may see these errors in the CLI under the following circumstances:

- When a control transaction is attempted.
- In service submode entry and service submode exit if changes were made.
- In tune-alarm-channel mode.
- In tune-virtual-sensor mode.

The commands executed while in the service submode are applied to a local copy of the `.xml` file. The only non-service CLI commands that do not execute a CT are “terminal length,” “display-serial,” “show history,” “show events,” and “clear events.”

- Error: `<appName>` not responding, please check system processes.

Where `<appName>` is replaced by an application name: `authentication`, `cidcli`, `cidwebserver`, `logApp`, `mainApp`, or `sensorApp`. This is a fatal error. It means you have lost communication with one of the applications. You must reboot the sensor to recover.

- Error: Control transaction has unregistered.

This should not occur in the normal run-time environment. It means the recipient of the CT has cleanly shut down while you were in the middle of attempting to complete a CT.

- Error: Unknown control transaction name.

This should not occur in the normal run-time environment. It is similar to the previous error except the CT was not registered at the beginning of attempting the CT. In non-CLI applications it may mean an incorrect CT name was used.

- Error: Control transaction cannot be completed at this time.

This occurs if an application is too busy to respond to the CT. Try again at a later time. This only occurs after changes have been made to the `sensorApp`, for commands such as, **tune-virtual-sensor**, **tune-alarm-channel**, **reset-signatures**, **sensing-interface**, and **shutdown**.

ATOMIC Engines

The following sections describe the ATOMIC engines:

- [About ATOMIC Engines, page A-17](#)
- [ATOMIC.ARP Engine Parameters, page A-18](#)
- [ATOMIC.ICMP Engine Parameters, page A-19](#)
- [ATOMIC.IPOPTIONS Engine Parameters, page A-20](#)
- [ATOMIC.L3.IP Engine Parameters, page A-20](#)
- [ATOMIC.TCP Engine Parameters, page A-21](#)
- [ATOMIC.UDP Engine Parameters, page A-22](#)
- [ATOMIC Engines Configuration Restrictions, page A-22](#)

About ATOMIC Engines

ATOMIC engines do not store persistent data across packets. Instead they can fire an alarm from the analysis of a single packet. Therefore, the basic features of these engines do not require attachment to a non-Global StorageKey. They use the StorageKey xxxx. Because ATOMIC engines have no storage, they are essentially 1 to 1 signatures. Each ATOMIC engine has discriminators specialized to its protocol.

ATOMIC.L3.IP is a general-purpose Layer 3 inspector. It can handle DataLength and Protocol Number comparisons. It also has some hooks for fragment and partial ICMP comparisons. None of the parameters are required, so a simple signature meaning “any IP packet” can be written.

ATOMIC.ICMP is specialized to the L4 protocol. It has many parameters, some with both a single value, and a Min/Max for the boundaries. You can use all the single parameters together in a signature, but it is not very practical. It does not have any required parameters, so you can write a simple “any ICMP packet” signature.

ATOMIC.UDP is specialized to the L4 protocol. It is a simple engine with basic capabilities to interrogate ports and packet lengths. It does not have required parameters, so you can write an “any UDP packet” signature or you can augment that signature with a port (or range), for example, “any UDP packet on port 53.”

ATOMIC.IPOPTIONS is a simple engine that decodes L3 options. There is one required Pupation parameter used for configuring signatures.

ATOMIC.TCP specializes in L4 TCP packets and has basic TcpFlags/Mask comparisons along with port filters and the SinglePacketRegex. The TcpFlags/Mask shares a common technique with the SWEEP*.TCP engines in the way they compare packets against the configured parameters to determine packets of interest. TcpFlags and Mask are required, but you can use a TcpFlags and Mask of ACK to register interest for non-connect, non-close packets. The SinglePacketRegex provides a simple Regex match capability so you can combine ports, flags, and regex match in a single signature.

ATOMIC.ARP is for basic Layer2 ARP signatures and also for more advanced detection of the ARP spoof tools dsniff and ettercap.

ATOMIC.ARP Engine Parameters

Table A-5 lists the ATOMIC.ARP engine parameters.

Table A-5 ATOMIC.ARP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
ArpOperation	NUMBER (0–255)	No	No	The ARP operation code this signature is interested in.
MacFlip	NUMBER (0–65535)	No	No	Fire an alarm when the MAC address changes more than this many times for the IP address.
RequestInbalance	NUMBER (0–65535)	No	No	Fire an alarm when there are this many more requests than replies on the IP address.
wantDstBroadcast	BOOLEAN (True, False)	No	No	Fire an alarm for this signature when it sees an ARP destination address of 255.255.255.255.
wantSrcBroadcast	BOOLEAN (True, False)	No	No	Fire an alarm for this signature when it sees an ARP source address of 255.255.255.255.

ATOMIC.ICMP Engine Parameters

Table A-6 lists the ATOMIC.ICMP engine parameters.

Table A-6 ATOMIC.ICMP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
IcmpCode	NUMBER (0–255)	No	No	The ICMP header CODE value.
IcmpId	NUMBER (0–65535)	No	No	The ICMP header IDENTIFIER value.
IcmpMaxCode	NUMBER (0–255)	No	No	The ICMP CODE values above this cause alarms.
IcmpMaxSeq	NUMBER (0–65535)	No	No	The ICMP SEQ values above this causes alarm.
IcmpMinCode	NUMBER (0–255)	No	No	The ICMP CODE values below this cause alarms.
IcmpMinSeq	NUMBER (0–65535)	No	No	The ICMP SEQ values below this cause alarms.
IcmpSeq	NUMBER (0–65535)	No	No	The ICMP header SEQUENCE value.
IcmpType	NUMBER (0–255)	No	No	The ICMP header TYPE value
IpTOS	NUMBER (0–255)	No	No	The IP header TYPE OF SERVICE value.

ATOMIC.IPOPTIONS Engine Parameters

Table A-7 lists the ATOMIC.IPOPTIONS engine parameters.

Table A-7 ATOMIC.IPOPTIONS Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
HasBadOption	BOOLEAN (True, False)	No	No	The list of options is malformed.
IpOption	NUMBER (0–255)	No	No	The IP option code to match.

ATOMIC.L3.IP Engine Parameters

Table A-8 lists the ATOMIC.L3.IP engine parameters.

Table A-8 ATOMIC.L3.IP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
MaxDataLen	NUMBER (0–65535)	No	No	An IP data length greater than this causes an alarm.
MaxProto	NUMBER (0–255)	No	No	An IP protocol number greater than this causes an alarm.
MinDataLen	NUMBER (0–65535)	No	No	An IP data length less than this causes an alarm.
MinProto	NUMBER (0–255)	No	No	An IP protocol number greater than this causes an alarm.
ProtoNum	NUMBER (0–255)	No	No	A single IP protocol number to alarm on.
isIcmp	BOOLEAN (True, False)	No	No	Examine ICMP packets only.
isImpossiblePacket	BOOLEAN (True, False)	No	No	Source IP equals destination IP.
isLocalhost	BOOLEAN (True, False)	No	No	Localhost (127.0.0.1) is seen in IP packet.

Table A-8 *ATOMIC.L3.IP Engine Parameters (continued)*

Parameter Name	Data Type	Protected	Required	Description
isOverrun	BOOLEAN (True, False)	No	No	A fragment overrun.
isRFC1918	BOOLEAN (True, False)	No	No	A reserved (RFC1918) IP address.

ATOMIC.TCP Engine Parameters

Table A-9 lists the ATOMIC.TCP engine parameters.

Table A-9 *ATOMIC.TCP Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
DstPort	NUMBER (0–65535)	No	No	A single Destination Port to match.
Mask	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The mask used in TcpFlags comparison.
PortRange	NUMBER (0–2)	No	No	The destination port: Only Low Ports (1), Only High Ports (2), or All. (0)
PortRangeSource	NUMBER (0–2)	No	No	The source port: Only Low Ports (1), Only High Ports (2), or All (0).
SinglePacketRegex	STRING	No	No	A regular expression to search for in a single TCP packet.
SrcPort	NUMBER (0–65535)	No	No	A single Source Port to match.
TcpFlags	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The TCP Flags to match when masked by Mask.

ATOMIC.UDP Engine Parameters

Table A-10 lists the ATOMIC.UDP engine parameters.

Table A-10 ATOMIC.UDP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
DstPort	NUMBER (0–65535)	No	No	A single destination port to match.
MinUDPLength	NUMBER (0–65535)	No	No	Fire an alarm when the packet UDP length is less than this.
ShortUDPLength	BOOLEAN (True, False)	No	No	Fire an alarm when the IP data length is less than the UDP header length.
SrcPort	NUMBER (0–65535)	No	No	A single source port to match.

ATOMIC Engines Configuration Restrictions

The following configuration restrictions apply to the ATOMIC engines:

- The master parameters `ResetAfterIdle` and `MaxInspectLength` are ignored by these engines.
- ATOMIC.UDP requires you to specify at least one of its parameters.
- ATOMIC.TCP has required `TcpFlags` and `Mask`.
- ATOMIC.L3.IP, ICMP, IPOPTIONS, or ARP do not have any required parameters.

FLOOD Engines

The following sections describe FLOOD engines:

- [About FLOOD Engines, page A-23](#)
- [FLOOD.HOST.ICMP Engine Parameters, page A-24](#)
- [FLOOD.HOST.UDP Engine, page A-24](#)

- [FLOOD.NET Engine Parameters, page A-25](#)
- [FLOOD Engines Configuration Restrictions, page A-26](#)

About FLOOD Engines

FLOOD engines analyze traffic directed at one host from many (FLOOD.HOST.*) or the aggregate traffic on the whole segment of the sensor (FLOOD.NET).

Host floods are an n to 1 signature that attaches a packets-per-second (PPS) rate counter to the destination address. The sampling is done on a per-second basis.

Net floods are an n to n signature that counts rate on a virtual sensor basis and does not use the addresses for counting. The virtual sensor means that if the physical sensor is monitoring more than one interface, the interfaces are configured into one or more interface groups, and a virtual sensor is attached to one and only one interface group. Sampling is also done on a per-second basis.

Each FLOOD.HOST engine has other discriminators specialized to its protocol. FLOOD.HOST.ICMP uses the parameter `IcmpType` to determine what kind of ICMP packets will be counted.

FLOOD.HOST.UDP provides three each of excluded source and destination ports so that counting is not performed when packets to and/or from these ports are seen.

FLOOD.NET uses the parameters `Gap`, `Peaks`, and `Rate` to compare current and recent past samples with the settings that will trigger the alarm. Further, Protocol can be changed to have different signatures for TCP, UDP, ICMP. `IcmpType` is the only L4 specific parameter supported. You can configure these signatures to send an alarm under the following condition: when there are more than “Peaks” number of seconds where PPS is greater than “Rate,” within the time period “AlarmThrottle,” with less than “Gap” seconds where the Rate is not exceeded.

FLOOD.HOST.ICMP Engine Parameters

Table A-11 lists the FLOOD.HOST.ICMP engine parameters.

Table A-11 FLOOD.HOST.ICMP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
IcmpType	NUMBER (0–256)	No	No	The ICMP header TYPE value.
Rate	NUMBER	No	Yes	The maximum allowed PPS.

FLOOD.HOST.UDP Engine

Table A-12 lists the FLOOD.HOST.UDP engine parameters.

Table A-12 FLOOD.HOST.UDP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
ExcludeDst1	NUMBER (0–65536)	No	No	The destination port to exclude from flood counting.
ExcludeDst2	NUMBER (0–65536)	No	No	The destination port to exclude from flood counting.
ExcludeDst3	NUMBER (0–65536)	No	No	The destination port to exclude from flood counting.
ExcludeSrc1	NUMBER (0–65536)	No	No	The source port to exclude from flood counting.
ExcludeSrc2	NUMBER (0–65536)	No	No	The source port to exclude from flood counting.
ExcludeSrc3	NUMBER (0–65536)	No	No	The source port to exclude from flood counting.
Rate	NUMBER	No	Yes	The threshold number of PPS. An alarm fires when a second occurs where Rate is greater than PPS.

FLOOD.NET Engine Parameters

Table A-13 lists the FLOOD.NET engine parameters.

Table A-13 FLOOD.NET Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Gap	NUMBER	No	No	The number of seconds allowed within the ThrottleInterval where PPS is less than Rate. If you get more than Gap seconds that are not suspects, the alarm does not fire and counting is reset.
IcmpType	NUMBER (0–256)	No	No	The ICMP type value from the header. This parameter is valid only when Protocol is set to ICMP.
Peaks	NUMBER	No	No	The threshold of suspect seconds. An alarm fires when there are more Peaks suspect seconds in a ThrottleInterval.
Rate	NUMBER	No	No	The threshold for PPS. A suspect second is defined as the case when the PPS exceeds the Rate. Set the Rate value to 0 for diagnostics/feedback mode.

The Protocol master parameter is used to set what protocol the FLOOD.NET signature is interested in. If it is ICMP, the IcmpType field is valid; otherwise, the IcmpType field is invalid.

The ThrottleInterval master parameter is used to specify the interval for sampling and comparison against the Peaks and Gaps.

FLOOD Engines Configuration Restrictions

You must specify a Rate parameter for all signatures in the FLOOD group.

FLOOD engines ignore the following master parameters:

- WantFrag
- MaxInspectLength
- ResetAfterIdle

FLOOD.NET Learning Mode—A special configuration element for the FLOOD.NET engine is feedback mode. Feedback mode replaces normal inspection with a regular diagnostic alarm. The alarm contains the Maximum PPS it saw in the interval. Feedback mode is specified when the Rate parameter is set to 0. You can use the diagnostic information to determine a baseline of network traffic and tune the signatures accordingly. The feedback is a plain string that shows “MaxPPS=xyz” in the alertDetails field of the alarm record. This plain text is similar to other alertDetails values and is displayed by the alarm viewer.

HTTP Deobfuscation

The following sections describe HTTP deobfuscation in the signature engines:

- [About HTTP Deobfuscation, page A-26](#)
- [Characters Not Converted, page A-27](#)
- [Decode Variations, page A-27](#)
- [Supported Decodings, page A-29](#)
- [Error Conditions, page A-29](#)
- [Deobfuscation Alarms, page A-30](#)

About HTTP Deobfuscation

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 initial IDS 4.0 release has default deobfuscation behavior for the Microsoft IIS web server.

Characters Not Converted

Any character that could potentially be a protocol-delimiting character is *not* converted to an ASCII equivalent. These special characters include the following:

- hex values 0x00 (Null)
- 0x09 (Tab)
- 0x0A (NewLine)
- 0x0D (CarriageReturn)
- 0x20 (Space)

When a special character is encountered in an encoded format, it is converted to UTF-8 single octet representation. If the deobfuscator encounters a Null in any encoded format, such as %e0%80%80 for example, it has an output of %00. This provides a consistent base representation to use for pattern-matching purposes in case a special character must be matched in a deobfuscated data stream.

Decode Variations

A number of implementations have introduced decoding variations. Other variations exist for the sole purpose of evading intrusion detection systems. The Cisco IDS deobfuscation feature supports decoding the following variations:

- Double Encoding
The code point passes through two levels of encoding. The base encoding can be either a UTF-8 single octet or Unicode %U encoding without variation. The second encoding can encode each octet of the base encoding with any encoding method and variation.

- Unencoded octets mixed with encoded octets in a UTF8 sequence.
Any octet except the first octet in a UTF8 sequence can be an unencoded value. For example the value 0x123 represented in UTF8 as %E0%84%A3 can be represented as %E0%*o*%A3 where *o* has an ASCII value of 0x84.
- Ambiguous bits
Some decoder implementations ignore certain bits in the encoding. For example, IIS 4.0 treats %C0 and %D0 identically discarding the fifth bit in a UTF8 two-octet encoding.
- Microsoft base-36
Another example of decoder implementation error, old versions of Microsoft's UTF8 decoder accept 36 characters (A-Z and 0-9) as valid hexadecimal characters in the UTF8 encoding instead of the normal 16 characters (A-F and 0-9).
- Alternate code pages
Most Windows-based personal computers have extended Latin code pages loaded. When an extended character is processed, it gets normalized to an ASCII equivalent. For example, the Unicode code point U+212C has an ASCII normalized equivalent of "b."
- Self referencing directories
The directory name foo/. / . /bar refers to the same path as foo/bar.
- Multiple directory delimiters
Some operating systems treat "/" and "\" equivalently as directory delimiters. In addition, repeated directory delimiters are ignored

Double encoding represents the worst case scenario of how many unique ways a single character can be encoded. Here is a rough calculation of how many unique ways the character "/" can be represented using the sequence %uhhhh, where each "h" digit is a hex character in a unicode sequence:

- "%" can be represented at least 140 ways
- "u" can be represented at least 3260 ways
- "h" can be represented at least 1000 ways

There are at least $1000^4 * 3260 * 140 = 4.56e+17$, or 45 quadrillion ways to encode a single character that Microsoft IIS 4.0 will understand using double encoding.

Supported Decodings

The Cisco IDS deobfuscation feature supports two base Unicode encoding techniques:

- UTF-8 encoding—Unicode (or Universal Character Set) Transformation Format, 8-bit encoding form. UTF-8 is the UTF that serializes a Unicode scalar value (code point) as a sequence of one to four bytes.

Characters are encoded using sequences of 1 to 3 octets. In a single octet “sequence,” the higher-order bit is set to 0, the remaining 7 bits are used to encode the character value. In a sequence of n octets, $n > 1$, the initial octet has the n higher-order bits set to 1, followed by a bit set to 0. The remaining bit(s) of that octet contain bits from the payload. The following octet(s) all have the higher-order bit set to 1 and the next bit set to 0 leaving 6 bits to be encoded in each octet with payload. See Table.

- Microsoft’s custom Unicode %Uxxxx encoding

Microsoft introduced an encoding method that prefixes the 16-bit Unicode number with “%U or %u” followed by four hexadecimal digits. For example, [Table A-14](#) shows that the code point U+1234 would be encoded as %U1234.

Table A-14 Unicode %Uxxxx Encoding Example

U-00000000 - U-0000007F:

U-00000080 - U-000007FF:

U-00000800 - U-0000FFFF:

Error Conditions

When a character sequence cannot be converted, the deobfuscator unwinds the error as best it can. For example, if %E0%84<09> is encountered where values in <> are raw hex values, the sequence <E0><84><09> is the output. For performance reasons, the deobfuscator does not keep the original values used to decode the sequence, and is only able to output the intermediate values used in the decoding process up to the point of the error condition. Likewise if %C0%A# is encountered, the sequence <C0>%A# is the output.

Deobfuscation Alarms

The deobfuscator sets flags in its state buffer to indicate when certain abnormal events have occurred. The inspector of the calling engine sends the actual alarm. [Table A-15](#) lists these alarms.

Table A-15 Deobfuscation Alarms

Alarm	Description	Trigger
5249.0	IDS evasive encoding	When NULL, CR, LF, “.”, “/”, or “\” are decoded in the URI section.
5250.0	IDS evasive double encoding	When NULL, CR, LF, “.”, “/”, or “\” are double decoded in the URI section.

OTHER Engine

The following sections describe the OTHER engine:

- [About the OTHER Engine, page A-30](#)
- [OTHER Engine Signature List, page A-31](#)
- [OTHER Engine Parameters, page A-32](#)

About the OTHER Engine

The OTHER engine handles signatures that do not fit into the other engine protocol decodes. The OTHER engine allows you to configure parameters for built-in signatures using the same engine infrastructure as the other engines. These signatures are not implemented in an OTHER engine, but are used by the specialized processors in the rest of the system, for example, StreamProcessor, FragProcessor, and so forth.



Caution

You cannot define custom signatures for the OTHER engine.

OTHER Engine Signature List

Table A-16 lists signatures that are part of the OTHER engine.



Note

You cannot add any custom signatures to this engine, because all the OTHER engine signatures are already supported by custom code.

Table A-16 OTHER Engine Signature List

Signature ID:Subsignature	Functional Location
994:1	Packet capture
994:2	Packet capture
95:1	Packet capture
995:2	Packet capture
1200	Fragment processing
1201	Fragment processing
1202	Fragment processing
1203	Fragment processing
1204	Fragment processing
1205	Fragment processing
1206	Fragment processing
1207	Fragment processing
1208	Fragment processing
1220	Fragment processing
3050	TCP stream processing
3250	TCP stream processing
3251	TCP stream processing
5249	HTTP deobfuscation
5250	HTTP deobfuscation

OTHER Engine Parameters

Table A-17 lists the OTHER engine parameters.

Table A-17 Other Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
HijackMaxOldAck	NUMBER	No	No	Maximum number of old dataless client-to-server acknowledgements allowed before triggering a Hijack.
HijackReset	BOOLEAN (True, False)	No	No	Hijack signature requires a reset to be present.
ServicePorts	RANGE	No	No	A comma-separated list of ports or port ranges where the target service may reside.
SynFloodMaxEmbryonic	NUMBER	No	No	The maximum number of allowed simultaneous embryonic connections to any service.
TrafficFlowTimeout	NUMBER	No	No	The number of seconds that must go by with no traffic to fire the alarm.

SERVICE Engines

The following sections describes the SERVICE engines:

- [About SERVICE Engines, page A-33](#)
- [SERVICE Engines Configuration Restrictions, page A-34](#)
- [SERVICE.DNS Engine Parameters, page A-34](#)
- [SERVICE.FTP Engine Parameters, page A-35](#)
- [SERVICE.GENERIC Engine Parameters, page A-36](#)
- [About SERVICE.HTTP Engine, page A-37](#)
- [SERVICE.IDENT Engine Parameters, page A-42](#)
- [SERVICE.MSSQL Engine Parameters, page A-43](#)

- [SERVICE.NTP Engine Parameters, page A-44](#)
- [SERVICE.RPC Engine Parameters, page A-44](#)
- [SERVICE.SMB Engine Parameters, page A-46](#)
- [About the SERVICE.SNMP Engine, page A-47](#)
- [SERVICE.SSH Engine Parameters, page A-49](#)

About SERVICE Engines

SERVICE engines analyze L5+ traffic between two hosts. These are one-to-one signatures that track persistent data on the STREAM (AaBb) for TCP or on the DUAL (AxBx) or QUAD (AaBb) for UDP. The engines decode and interpret the L5+ payload in a manner similar to the live service. A full service-like decode may not be necessary if the partial decode provides adequate information to inspect the signatures. The engines decode enough of the data to make the signature determinations, but do not decode more than is needed so that CPU and memory load are minimized.

SERVICE engines have common characteristics, such as using the output from the STREAM processor, but each engine has specific knowledge of the service that it is inspecting. SERVICE engines supplement the capabilities of the generic string engine specializing in algorithms where using the string engine is inadequate or undesirable.

The purpose of the SERVICE decode is to mimic the live server's interpretation of the L5+ payload. Results of the interpretation are the decoded fields of the protocol. These are used primarily in the determination of signatures as the decoded fields are compared to the signature's parameters.

As the engine is decoding, errors with bad payloads can occur. These error conditions are linked to a different kind of signature known as Protocol Violations or Error Traps. These occur when the engine is decoding the payload and an error occurs because of a malformation in the payload that violates the rules of the service's protocol. An error trap handles this in the analysis code. Specifying the trap conditions that map to signatures is done by using the normal parameters, such as the SERVICE.FTP with BadPort. In some cases, these trap conditions can be combined to form a signature that results when multiple trap conditions are encountered. However, in most cases, the trap conditions have a one-to-one mapping to the trap signatures.

SERVICE Engines Configuration Restrictions

SERVICE engines do not use the following parameters:

- ResetAfterIdle
- WantFrag
- MaxInspectLength

The following configuration restrictions apply to SERVICE.RPC:

- The master parameter Protocol needs to specify either TCP or UDP.
- You must specify a value for PortMapProgram if isPortMapper is True.
- You must specify a value for RpcProgram if isPortmapper is False.
- You must specify a value for Unique if isSweep is True.
- You cannot specify a value for Unique if isSweep is false.

SERVICE.DNS does not have any required parameters, so the interface is open for custom signature extension. You need to add UDP and TCP signatures to have complete coverage. You must specify either TCP or UDP for the master parameter, Protocol.

For SERVICE.FTP you can combine the Boolean BadPortCmdAddress and BadPortCmdPort in one or many signatures. You can only use the Boolean BadPortCmdShort by itself. You can only use the Boolean hasPasvSpooft by itself.

For SERVICE.IDENT you can combine MaxBytes, hasNewline, and HasBadPort, but the practical application of this is limited.

You cannot add any new signatures to the SERVICE.SSH engine. Only two signatures are supported by this engine.

You cannot add any new signatures to SERVICE.GENERIC. Signature updates add new signatures for this engine.

SERVICE.DNS Engine Parameters

SERVICE.DNS specializes in advanced DNS decodes, which includes anti-evasive techniques, such as following multiple jumps. This engine has many parameters and operates on both TCP and UDP port 53; thus, it is a bi-protocol Inspector. It uses the STREAM for TCP and the QUAD for UDP.

Table A-18 lists the SERVICE.DNS engine parameters.

Table A-18 SERVICE.DNS Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
QueryChaosString	STRING	No	No	DNS query class chaos string.
QueryClass	NUMBER	No	No	DNS query class 2-byte value.
QueryInvalidDomainName	BOOLEAN (True, False)	No	No	DNS query length >255.
QueryJumpCountExceeded	BOOLEAN (True, False)	No	No	DNS compression counter.
QueryOpcode	NUMBER	No	No	DNS query opcode 1-byte value.
QueryRecordDataInvalid	BOOLEAN (True, False)	No	No	DNS record data incomplete.
QueryRecordDataLen	NUMBER	No	No	DNS response record data length.
QuerySrcPort53	BOOLEAN (True, False)	No	No	DNS packet source port 53.
QueryStreamLen	NUMBER	No	No	DNS packet length.
QueryType	NUMBER	No	No	DNS query type 2-byte value.
QueryValue	BOOLEAN (True, False)	No	No	Query 0 Response 1.

SERVICE.FTP Engine Parameters

SERVICE.FTP fills in the gap where the STRING engine is not appropriate for the detection. It is mainly centered around the FTP port command decode and traps invalid port command and the PASV spoofer. The parameters are BOOLEANs that provide a mapping to the various error trap conditions in the port command decode. FTP runs on TCP port 20 and 21. The sensor inspects port 21 for the CTs, but not port 20, which has only the data.

Table A-19 lists the SERVICE.FTP engine parameters.

Table A-19 SERVICE.FTP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
BadPortCmdAddress	BOOLEAN (True, False)	No	No	An invalid address was specified in the port command.
BadPortCmdPort	BOOLEAN (True, False)	No	No	An invalid port was specified in the port command.
BadPortCmdShort	BOOLEAN (True, False)	No	No	The port command was malformed (too short).
Direction	ENUM (ANY, FromService, ToService)	No	No	ToService or FromService indicates whether the sensor is watching traffic destined to or coming from the service port.
ServicePorts	RANGE	No	No	A comma-separated list of ports or port ranges where the target service may reside.
isPASV	BOOLEAN (True, False)	No	No	A PASV port spoof was detected.

SERVICE.GENERIC Engine Parameters

SERVICE.GENERIC is an unusual engine. It is not intended to be used to write signatures. It is a facility to allow programmatic signatures to be issued in a configuration-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 knows how to process the instructions and pull the important pieces of information out of the packet and run them through the comparisons and operations specified in the machine code.

It is intended as an emergency response engine that supplements the STRING and STATE engines for rapid signature response.




Caution

You cannot make custom signatures with this engine.

Table A-20 lists the SERVICE.GENERIC engine parameters.

Table A-20 SERVICE.GENERIC Engine Parameters

Parameter Name	Date Type	Protected	Required	Description
DstPort	NUMBER (0–65535)	No	No	The destination port of interest for this signature.
IntermediateInstructions	STRING	Yes	No	Assembly or machine code. Machine-ready (regname, oper, opnd1, opnd2) in string form.  Caution For expert use only.
PayloadSource	ENUM (FullTcpStream ICMPdata L2header L3header L4header TCPdata UDPdata)	No	No	Where to start payload offset calculations.
SrcPort	NUMBER (0–65535)	No	No	The source port of interest for this signature.

About SERVICE.HTTP Engine

SERVICE.HTTP provides regular expression-based pattern inspection and alarm functionality that is specifically designed for the Hyper-Text Transport protocol (HTTP). It is a string-based pattern-matching inspection engine.

Regular expressions (Regex) are a powerful and flexible notational language that allow you to describe text. Regular expressions are compiled into a data structure called a pattern matcher that is used to match patterns in data. In the context of pattern matching, regular expressions allow a succinct description of any arbitrary pattern.

SERVICE.HTTP uses an updated Regex library that can combine multiple patterns into a single pattern matching table that allows for a single search through the data. The new Regex also contains the alternation “|” operator also known as the OR operator.

SERVICE.HTTP only searches traffic directed to web services, or HTTP requests. Return traffic cannot be inspected using this engine. Each signature defined on this engine may independently specify separate web ports of interest.

This section contains the following topics:

- [HTTP Field Sections, page A-38](#)
- [String Match Length, page A-39](#)
- [SERVICE.HTTP Engine Limitations and Restrictions, page A-40](#)
- [SERVICE.HTTP Engine Parameters, page A-41](#)

HTTP Field Sections

The primary goal of the SERVICE.HTTP engine is to subdivide the HTTP request into specific sections after the HTTP request method (GET, POST, or HEAD, for example). Subdividing the requests makes the regular expressions more simple and the size of the pattern matcher is not as large.

Example shows an example undivided HTTP request.

```
POST /eng/Tech/projectB/foobar.html?name=john&last=doe HTTP/1.0
<CRLF>
Accept: text/html
User-Agent: Mozilla
Host: 10.1.20.55
Context-Length: 45
<CRLF>
<CRLF>
Argument1=td&Argument2=foobar&middlename=levi
<CRLF>
```

The HTTP request is divided into three sections:

- URI

Uniform Resource Identifier. A name to identify the access method and resource name of an entity.

The URI is the section that follows immediately after the HTTP method up to and including the first LF or argument delimiter (?&).

```
/eng/Tech/projectB/foobar.html
```

- Argument

If the URI contains an argument delimiter “&” or “?” the arguments start immediately following the delimiter up to the first LF. If the POST method is used, the arguments also include all data in the entity-body as defined by the Content-Length header field. The Arguments section may be empty if neither of these cases is true. In the case of a POST, the entity body may be inspected before the arguments in the URI request line.

```
name=john&last=doe
```

```
Argument1=td&Argument2=foobar&middlename=levi
```

- Header

The Header comes immediately after the first LF up to the double <CRLF><CRLF>.

```
Accept: text/html
```

```
User-Agent: Mozilla
```

```
Host: 10.1.20.55
```

```
Content-Length: 45
```

String Match Length

The MinMatchLength parameter supports the length of string match. The length is calculated starting with the character preceding the first repetition operator (* or +) and ends with the last character in the pattern. For example, if the pattern is foobar.*abc, the data being searched is “foobarxxxxxxxxxxabc.” The length recorded by the Regex search is 13, which is the distance from the “r” in foobar to the “c” in abc. If a MinMatchLength is defined in a signature description parameter but the RegexString does not have a repetition operator (including each substring when using alternation “|” operator), an error is logged and the signature

description is *not* processed. Regular expressions without repetition are fixed length and always have the same match length unless alternation is used, in which case the length is one of the fixed alternation lengths.

SERVICE.HTTP Engine Limitations and Restrictions

Table A-21 lists the limitations for the SERVICE.HTTP engine.

Table A-21 SERVICE.HTTP Engine Limitations

Limitation	Limit	Error Handling
The maximum number of states in a single pattern matcher	65500	New pattern matcher generated and added to inspection list.
The maximum number of pattern matchers	25	Error reported to log.
The maximum number of patterns in a pattern matcher with MinMatchLength defined.	255	New pattern matcher generated and added to inspection list.



Note

Because HTTP processing requires a lot of time, if a valid HTTP method (GET, HEAD, POST) is not detected in the first 20 bytes of the request, inspection is stopped for the entire data stream.

The following restrictions apply to the SERVICE.HTTP engine:

- MinRequestMatchLength is only applicable when a RequestRegex has been specified and RequestRegex contains an iterator (* or +).
- An Error is generated if MaxRequestFieldLength is less than MinRequestMatchLength.

SERVICE.HTTP Engine Parameters

Table A-22 lists the SERVICE.HTTP engine parameters.

Table A-22 SERVICE.HTTP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
UriRegex ¹	STRING	Yes	No	The regular expression of the pattern to match in the URI section of the HTTP request. The URI field is defined to be after the valid HTTP method and before the first CRLF or argument delimiter (?&).
ArgNameRegex ²	STRING	Yes	No	The regular expression of the pattern to match in the Arguments section.
ArgValueRegex ³	STRING	Yes	No	The regular expression of the pattern to match in the Arguments section after ArgNameRegex is matched. An ordered match is performed. ArgNameRegex and ArgValueRegex together are useful to match argument key-value pairs.
HeaderRegex ⁴	STRING	Yes	No	The regular expression of the pattern to match in the Header section.
RequestRegex ⁵	STRING	Yes	No	The regular expression of the pattern to match over the entire request.
MaxUriFieldLength	NUMBER (0-4294967295)	No	No	The maximum length of the URI field.
MaxArgFieldLength	NUMBER (0-4294967295)	No	No	The maximum length of the Arguments field.

Table A-22 SERVICE.HTTP Engine Parameters (continued)

Parameter Name	Data Type	Protected	Required	Description
MaxHeaderFieldLength	NUMBER (0-4294967295)	No	No	The maximum length of the Header field.
MaxRequestFieldLength	NUMBER (0-4294967295)	No	No	The maximum length of the Header field.
Deobfuscate	BOOLEAN (True, False)	No	No	Apply anti-evasive HTTP deobfuscation before searching.
ServicePorts	SET	No	Yes	A comma-separated list of ports or port ranges where the target service may reside.
MinRequestMatchLength ⁶	NUMBER (0-4294967295)	No	No	The minimum number of bytes the UriRegex must match.

1. The string must be a valid regular expression.
2. The string must be a valid regular expression. Match state is reset when an argument delimiter is reached.
3. The string must be a valid regular expression. ArgNameRegex must also be defined. Match state is reset when an argument delimiter is reached.
4. The string must be a valid regular expression.
5. The string must be a valid regular expression.
6. This parameter requires the RequestRegex to have a repetition operator such as * or +, otherwise it will be rejected.

SERVICE.IDENT Engine Parameters

The SERVICE.IDENT engine is for TCP port 113 traffic. It has basic decode and provides parameters to specify length overflows and looks for the decode error traps. It does not require any cross-packet storage other than what is handled externally in the Stream Reassembly engine.

Table A-23 lists the SERVICE.IDENT engine parameters.

Table A-23 SERVICE.IDENT Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Direction	ENUM (FromService, ToService)	No	No	ToService or FromService indicates whether the sensor is watching traffic destined to or coming from the service port.
MaxBytes	NUMBER (0–65535)	No	No	Fires alarm if the payload length is longer than this.
ServicePorts	RANGE	No	No	A comma-separated list of ports or port ranges where the target service may reside.
hasBadPort	BOOLEAN (True, False)	No	No	Fires alarm if IDENT payload contains a bad port specifier.
hasNewline	BOOLEAN (True, False)	No	No	Fires alarm if payload contains a nonterminating newline character.

SERVICE.MSSQL Engine Parameters

The SERVICE.MSSQL engine inspects the protocol used by Microsoft’s SQL server (MSSQL). You can add custom signatures based on MSSQL protocol values, such as login username and whether a password was used.

Table A-24 lists the SERVICE.MSSQL engine parameters.

Table A-24 SERVICE.MSSQL Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
sqlUsername	STRING	No	No	The username (exact match) of the user logging into the MSSQL service.
passwordPresent	BOOLEAN	No	No	Whether or not a password was used in an MSSQL login.

SERVICE.NTP Engine Parameters

The SERVICE.NTP engine inspects the Network Time Protocol (NTP).

Table A-25 lists the SERVICE.NTP engine parameters.

Table A-25 SERVICE.NTP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Mode	NUMBER (0x0-0xffff)	No	No	The mode of operation of NTP packets per RFC 1305.
MaxSizeOfControlData	NUMBER (0x0-0xffff)	No	No	Maximum allowed amount of data sent in a control packet.
ControlOpCode	NUMBER (0x0-0xffff)	No	No	The opcode number of an NTP control packet according to RFC1305, Appendix B.
isInvalidDataPacket ¹	BOOLEAN (True, False)	No	No	Checks the structure of the NTP data packet to ensure it has the correct size.
isNonNtpTraffic ²	BOOLEAN (True, False)	No	No	The parameter to check for non (or strange) traffic on an NTP port.

1. This token cannot be used with any other engine specific token.

2. This token cannot be used with any other engine specific token.

SERVICE.RPC Engine Parameters

The SERVICE.RPC engine decoder has full decode as an anti-evasive strategy. It can handle fragmented messages (one message in several packets) or batch messages (several messages in a single packet). The RPC port mapper 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. They also segregate on each RPC program type for sweep unique counting.

Table A-26 lists the SERVICE.RPC engine parameters.

Table A-26 SERVICE.RPC Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Direction	ENUM (FromService, ToService)	No	No	ToService or FromService indicates whether the sensor is watching traffic destined to or coming from the service port.
PortMapProgram	NUMBER (0–99999)	No	No	The program number sent to the port mapper of interest for this signature.
RpcMaxLength	NUMBER (0–99999)	No	No	The maximum allowed length of the whole RPC message. Lengths longer than this cause an alarm.
RpcProcedure	NUMBER (0–65535)	No	No	The RPC procedure number of interest for this signature.
RpcProgram	NUMBER (0–99999)	No	No	The RPC program number of interest for this signature.
ServicePorts	RANGE	No	No	A comma-separated list of ports or port ranges where the target service may reside.
Unique	NUMBER (2–40)	No	No	This parameter is for sweeps. The threshold number of the unique port connections allowed until the alarm fires. You must have parameter isSweep set to True for this to be valid.
isPortMapper	BOOLEAN (True, False)	No	No	True if the signature is interested in port 111 traffic; false if it is not.
isSpoolSrc	BOOLEAN (True, False)	No	No	Fire the alarm when the SrcAddress is set to 127.0.0.1.
isSweep	BOOLEAN (True, False)	No	No	True if this signature is an RPC sweep; false if it is not. You must set the value for Unique for this to be valid.

SERVICE.SMB Engine Parameters

The SERVICE.SMB engine decodes the SMB protocol. The following built-in signatures are included in the SERVICE.SMB engine:

- 3303—Login successful with guest privileges.
- 3304—NULL login attempt.
- 3305—Windows 95/98 password file access.
- 3306—Remote Registry access attempt.
- 3307—RedButton reconnaissance.
- 3308—Remote isarpc service access attempt.
- 3309—Remote srsvsvc service access attempt.
- 6255—SMB login failure.



Note

The list of signatures changes with each signature update. This list applies to the 4.0 release.



Caution

You cannot add custom signatures to the SERVICE.SMB engine. If you try to add custom signatures to the SERVICE.SMB engine, you receive the following error message: `Error: Array contains max entries, could not add new entry.`

Table A-27 lists the SERVICE.SMB engine parameters.

Table A-27 SERVICE.SMB Engine Parameters

Parameter Name	Data Type	Protected	Required	Description Example
ScanInterval	NUMBER 1–131071	No	No	The time interval in seconds that is used to determine alarm rates (for signature 6255 only).
HitCount	NUMBER 1–65535	No	No	The number of occurrences in the ScanInterval that causes the alarm to be triggered (for signature 6255 only).

Table A-27 SERVICE.SMB Engine Parameters (continued)

Parameter Name	Data Type	Protected	Required	Description Example
PipeName	STRING 260 bytes	No	No	Name of the pipe to watch for in NT_CREATE_ANDX (not implemented for user signatures).
AccountName	STRING 260 bytes	No	No	The account name to send alarms on the attempts to log in (not implemented for user signatures).
FileName	STRING 260 bytes	No	No	Filename to alarm on attempts to open (not implemented for user signatures.)

**Note**

You can only tune the ScanInterval and HitCount parameters with signature 6255 (SMB Login failure).

About the SERVICE.SNMP Engine

The SERVICE.SNMP engine inspects the SNMP protocol. The SERVICE.SNMP engine filters on all UDP packets destined for port 161.

This section contains these topics:

- [SERVICE.SNMP Engine Limitations, page A-48](#)
- [SERVICE.SNMP Engine Parameters, page A-48](#)

SERVICE.SNMP Engine Limitations

Table A-28 lists the current limitations for the SERVICE.SNMP engine.

Table A-28 SERVICE.SNMP Engine Limitations

Limitation	Limit	Error Handling
Maximum length of decoded community name	256 Chars	Fire a protocol violation alarm.
Maximum length of decoded object identifier	256 Chars	Fire a protocol violation alarm.

The SERVICE.SNMP engine inspects only SNMP version 1.



Caution

There is a small possibility that two different community name strings may produce the same integer hash. This can result in false positives for community name matches. The hash of a community name is constructed by shifting up to the first four bytes on the community name into a 32-bit integer. Any remaining bytes are added to the integer sum.

SERVICE.SNMP Engine Parameters

The CommunityName strings are converted into an integer-sized hash, which speeds up the protocol decode and reduces storage space. The CommunityName decoded from the packet is also converted to an integer hash. Each CommunityName string should produce a near unique integer hash. These hashes are used to determine if the CommunityNames match. The hashes are also stored and compared to determine if a brute force attempt occurred.

ObjectId strings are converted into an array of integers, which speeds up the protocol decode and reduces storage space. The ObjectIds decoded from the packet are also converted into an integer array. These arrays are compared to determine if the ObjectIds match.

Table A-29 lists the SERVICE.SNMP engine parameters.

Table A-29 SERVICE.SNMP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
CommunityName ¹	STRING	No	No	The SNMP password that is being searched for.
ObjectId ²	STRING	No	No	The object identifier being searched for.
BruteForceCount ³	NUMBER	No	No	The number of unique community names seen on a dual to constitute a brute force attempt.
IsBruteForce ⁴	BOOLEAN (True, False)	Yes	No	The token that signifies the definition of the BruteForceCount.
IsValidPacket ⁵	BOOLEAN (True, False)	Yes	Yes	The token that signifies an SNMP protocol violation.
IsNonSnmpTraffic ⁶	BOOLEAN (True, False)	Yes	No	The token that signifies non-SNMP traffic destined for UDP port 161.

1. A character string >1 and <256 in length.
2. A character string >1 and <256 in length. The string must be of the form: .1.3.6.4.1 The minimum string is: .<number>.
3. An integer >1 and <32 in value.
4. This token can only be used with the BruteForceCount token.
5. This token cannot be used with any other engine specific token.
6. This token cannot be used with any other engine specific token.

SERVICE.SSH Engine Parameters

The SERVICE.SSH engine is specialized for port 22 SSH traffic. Because everything 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 existing signatures, but you cannot add new SSH signatures.



Caution

You cannot add custom signatures to the SERVICE.SSH engine. If you try to add custom signatures to the SERVICE.SSH engine, you receive the following error message: `Error: Array contains max entries, could not add new entry.`

Table A-30 lists the SERVICE.SSH engine parameters.

Table A-30 SERVICE.SSH Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Direction	ENUM (ANY, FromService, ToService)	No	No	ToService or FromService indicates whether the sensor is watching traffic destined to or coming from the service port.
KeyLength	NUMBER (0–65535)	No	No	Keys larger than this fire an RSAREF overflow.
PacketDepth	NUMBER (0–65535)	No	No	The number of packets to watch before determining that a session key was missed.
ServicePorts	RANGE	No	No	A comma-separated list of ports or port ranges where the target service may reside.
UserLength	NUMBER (0–65535)	No	No	Username fields larger than this fire the USERNAME overflow.

STATE.STRING Engines

This section contains the following topics:

- [About the STATE.STRING Engines, page A-50](#)
- [STATE.STRING Engine Limitations, page A-52](#)
- [STATE.STRING Engine Parameters, page A-52](#)
- [Predefined State Machines, page A-53](#)

About the STATE.STRING Engines

The STATE.STRING engine provides state-based, regular expression-based, pattern inspection and alarming functionality for TCP streams.

The STATE.STRING engine allows you to define arbitrary string-based state machines, which allow a signature to be confined to the context of the state machine. A *state machine* is a device that stores the state of something and at a given time can operate on input to move 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. This is a very powerful concept that can be applied to protocol decoding and arbitrary network transactions. The STATE.STRING engine reduces the number of false positives due to higher fidelity signatures.

**Caution**

The initial release of the STATE.STRING engine does not have state machines that are configurable or upgradeable during a signature update.

Each StateMachine that is defined resides in an independent engine. The name of each engine is STATE.STRING.<StateMachine name>, where <StateMachine name> is the name of the StateMachine that has been defined. The StateMachine definitions reside in a separate .xml file that you can eventually update with signature updates. You cannot configure the StateMachines in the initial release (version 4.0). Each StateName defined in the StateMachine configuration is visible through the StateName enumeration. You can use the predefined StateNames to define custom signatures. You should understand what each State represents so you can use it effectively.

STATE.STRING Engine Limitations

Table A-31 lists the STATE.STRING engine limitations.

Table A-31 STATE.STRING Engine Limitations

Limitation	Limit	Error Handling
The maximum number of states in a single pattern matcher.	65500	A new pattern matcher is generated and added to the inspection list.
The maximum number of pattern matchers.	25	An error message is generated.
The maximum number of patterns in a pattern matcher with MinMatchLength defined.	255	A new pattern matcher is generated and added to the inspection list.

STATE.STRING Engine Parameters

Table A-32 lists the STATE.STRING engine parameters.

Table A-32 STATE.STRING Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Direction	ENUM (ToService or FromService)	Yes	Yes	Indicates whether to inspect traffic destined to or coming from the service ports.
MaxInspectLength	NUMBER (1-4294967295)	Yes	No	The maximum depth into the data stream to inspect for this signature. The RegexString must match at or before this point.
EndMatchOffset	NUMBER (1-4294967295)	Yes	No	The exact stream offset in bytes in which the RegexString must report the match.

Table A-32 STATE.STRING Engine Parameters (continued)

Parameter Name	Data Type	Protected	Required	Description
MinMatchLength ¹	NUMBER (1–4294967295)	Yes	No	The minimum number of bytes the RegexString must match from the start of the match to end of the match.
RegexString ²	STRING	Yes	Yes	The regular expression pattern.
ServicePorts	SET	Yes	Yes	A comma-separated list of ports or port ranges where the target service may reside.
StateName ³	ENUM	Yes	Yes	The name of the state in this StateMachine to restrict the match of the RegexString.

1. This parameter requires the regular expression to have a repetition operator such as *, or +, otherwise it is rejected. RegexStrings without repetition are fixed length and always have the same match length.
2. The RegexString needs to be a string in the form of a regular expression.
3. This enumeration is defined by the StateMachine definition.

Predefined State Machines

A state machine consists of a starting StateName and a list of transitions. The starting StateName is the name of a state that is defined in a transition that serves as the initial state of the state machine.

This section contains these topics:

- [State Machine Engine Transition Parameters, page A-54](#)
- [SERVICE.SMTP Engine Transitions, page A-54](#)
- [STATE.STRING.CISCOLOGIN Engine Transitions, page A-55](#)
- [STATE.STRING.LPRFORMAT Engine Transitions, page A-56](#)

State Machine Engine Transition Parameters

Table A-33 lists parameters that define a transition for a state machine engine.

Table A-33 STATE Machine Engine Transition Parameters

Parameter Name	Data Type	Description
RegexString	STRING	The regular expression string that triggers a state transition.
MaxInspectLength	NUMBER (1–4294967295)	The maximum stream offset in bytes that can be inspected for the RegexString.
EndOffset	NUMBER (1–4294967295)	The exact stream offset in bytes the RegexString must match.
RequiredState	STRING	The current state of the state machine required for the transition to occur. Each unique string entered defines the ENUM for RequiredState parameter.
NextState	STRING	When the transition occurs, NextState is the new state. Each unique string entered defines the enumeration for RequiredState parameter.
Direction	ENUM (ToService or FromService)	The direction of the data stream to inspect for the transition.

SERVICE.SMTP Engine Transitions

Table A-34 shows SERVICE.SMTP engine transitions for the initial state START.

Table A-34 SERVICE.SMTP Engine Transitions

Regex String	Required State	Next State	Direction
[\\r\\n]250[]	START	SmtPCommands	FromService
220[]([\\r\\n[\\x7f-\\xff])*SNMP	START	SmtPCommands	FromService
(HE EH)LO	START	SmtPCommands	ToService

Table A-34 SERVICE.SMTP Engine Transitions (continued)

Regex String	Required State	Next State	Direction
<code>[\r\n](235 220.*TLS)</code>	START	ABORT	FromService
<code>[\r\n](235 220.*TLS)</code>	SmtptCommands	ABORT	FromService
<code>[Dd][Aa][Tt][Aa][Bb][Dd][Aa][Tt]</code>	SmtptCommands	MailHeader	ToService
<code>[\r\n]354</code>	SmtptCommands	MailHeader	FromService
<code>[\r\n][.][\r\n]</code>	MailHeader	SmtptCommands	ToService
<code>[\r\n][2][0-9][0-9][]</code>	MailHeader	SmtptCommands	FromService
<code>([\r][\n][\n][\r]){2}</code>	MailHeader	MailBody	ToService
<code>[\r\n][.][\r\n]</code>	MailBody	SmtptCommands	ToService
<code>[\r\n][2][0-9][0-9][]</code>	MailBody	SmtptCommands	FromService

STATE.STRING.CISCOLOGIN Engine Transitions

Table A-35 shows STATE.STRING.CISCOLOGIN engine transitions for the initial state START.

Table A-35 STATE.STRING.CISCOLOGIN Engine Transitions

Regex String	Required State	Next State	Direction
<code>User[]Access[]Verification</code>	START	CiscoDevice	FromService
<code>Cisco[]Systems[]Console</code>	START	CiscoDevice	FromService
<code>assword[:]</code>	CiscoDevice	PassPrompt	FromService
<code>\x03</code>	PassPrompt	ControlC	ToService
<code>(enable)</code>	ControlC	EnableBypass	FromService
<code>\x03[\x00-\xFF]</code>	ControlC	PassPrompt	ToService

STATE.STRING.LPRFORMAT Engine Transitions

Table A-36 shows STATE.STRING.LPRFORMAT engine transitions for the initial state START.

Table A-36 STATE.STRING.LPRFORMAT Engine Transitions

Regex String	End Offset	Required State	Next State	Direction
[1-9]	1	START	ABORT	ToService
%		START	FormatChar	ToService
[\x0a\x0d]		FormatChar	ABORT	ToService

STRING Engines

This section contains these topics:

- [About STRING Engines, page A-56](#)
- [String Match Length, page A-57](#)
- [STRING Engine Parameters, page A-57](#)
- [String Engine Limitations, page A-58](#)

About STRING Engines

The STRING engine provides regular expression-based pattern inspection and alarm functionality for multiple transport protocols including TCP, UDP and ICMP.

Regular expressions 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. Regular expressions are compiled into a data structure called a pattern matcher, which is then used to match patterns in data.

The STRING engine is a generic string-based pattern matching inspection engine for TCP, UDP, and ICMP protocols. This STRING engine uses a new Regex engine that can combine multiple patterns into a single pattern-matching table

allowing for a single search through the data. The new regex has the alternation “|” operator also known as the OR operator. There are three STRING engines: STRING.TCP, STRING.UDP, and STRING.ICMP.

String Match Length

The MinMatchLength parameter supports length of string match. The length is calculated starting with the character preceding the first repetition operator (* or +) and ends with the last character in the pattern. For example, if the pattern is foobar.*abc, the data being searched is “foobarxxxxxxxxxxxabc.” The length recorded by the Regex search is 13, which is the distance from the “r” in foobar to the “c” in abc. If a MinMatchLength is defined in a signature description parameter but the RegexString does not have a repetition operator, an error is logged and the signature description is *not* processed. It is also an error if the OR operator is used and not all the operands have repetition. For example “catldog+ylmouse.*trap” logs an error. Regular expressions without repetition are fixed length and always have the same match length unless alternation is used, in which case the length is one of the fixed alternation lengths.

STRING Engine Parameters

Table A-37 lists the STRING engine parameters.

Table A-37 STRING Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
Direction ¹	BOOLEAN	Yes	Yes	Indicates whether to inspect traffic destined to or coming from the service ports.
EndMatchOffset	NUMBER	Yes	No	The exact stream offset in bytes that the RegexString must use to report the match.
MinMatchLength ²	NUMBER	Yes	No	The minimum number of bytes the RegexString must match.
RegexString ³	STRING	Yes	Yes	The Regular Expression pattern.

Table A-37 *STRING Engine Parameters (continued)*

Parameter Name	Data Type	Protected	Required	Description
ServicePorts	SET	No	Yes	A comma-separated list of ports or port ranges where the target service may reside.
StripTelnetOptions	BOOLEAN	Yes	No	Strips the Telnet option control characters from the data stream before the pattern is searched. Primarily used as an IDS anti-evasion tool.

1. ToService or FromService.
2. This parameter requires the regular expression to have a repetition operator such as * or +, otherwise it is rejected. RegexStrings without repetition are fixed length and always have the same match length.
3. The RegexString needs to be a string in the form of a regular expression.

String Engine Limitations

[Table A-38](#) lists limitations that apply to the STRING engine.

Table A-38 *STRING Engine Limitations*

Limitation	Limit	Error Handling
The maximum number of states in a single pattern matcher.	65500	New pattern matcher generated and added to inspection list.
The maximum number of pattern matchers.	25	Error reported to log.
The maximum number of patterns in a pattern matcher with MinMatchLength defined.	255	New pattern matcher generated and added to inspection list.

SWEEP Engines

This section contains these topics:

- [About SWEEP Engines, page A-59](#)
- [SWEEP Engine Configuration Restrictions, page A-60](#)
- [SWEEP.HOST.ICMP Engine Parameters, page A-60](#)
- [SWEEP.HOST.TCP Engine Parameters, page A-61](#)
- [SWEEP.MULTI Engine Parameters, page A-61](#)
- [SWEEP.OTHER.TCP Engine Parameters, page A-62](#)
- [SWEEP.PORT.TCP Engine Parameters, page A-63](#)
- [SWEEP.PORT.UDP Engine Parameters, page A-64](#)

About SWEEP Engines

SWEEP engines analyze traffic either between two hosts (SWEEP.PORT.*) or from a host to many hosts (SWEEP.HOST.*). Host Sweeps are one-to-*n* signatures that count unique host connections from a single attacker host and send alarms when the Unique count of hosts are greater than its threshold. Port Sweeps are one-to-one signatures that count unique port connections between the two hosts and send alarms when the Unique count is greater than its threshold.

SWEEP.MULTI is responsible for cross-protocol sweeps, such as the old SATAN signatures which trigger when both UDP and TCP ports are swept.

SWEEP.OTHER.TCP is responsible for non-normal sweeps, such as the Queso, or NMAP sweeps that send funny TcpFlags combinations and try to fingerprint the operating system of the target machine. This engine does not do the regular type of Unique counting as the other SWEEP engines. Instead, it is looking for different sets of TcpFlags sent to the victim.

Each of the SWEEPS' alarm conditions ultimately depend on the count of Unique. Unique is the threshold parameter that triggers firing the alarm when more than Unique number of ports/hosts is seen on the address set within the time period. The processing of Unique port/host tracking is called counting. Getting into the counting section is controlled by the other parameters such as IcmpType, Mask/TcpFlags, WantFrag boolean, or the UDP ports. These parameters are the main discriminators that bypass counting when the packet conditions are not met.

SWEEP Engine Configuration Restrictions

SWEEP engines use the master parameter, `ResetAfterIdle`. `ResetAfterIdle` is used to clear the current state of Unique counting when traffic has been absent on the inspector for X seconds. This means that the hosts being tracked on the inspector (the address set) did not have any traffic in the last X seconds. These engines may also use the `WantFrag` master parameter. They do not use the `MaxInspectLength` parameter.

SWEEP.HOST.ICMP Engine Parameters

[Table A-39](#) lists the SWEEP.HOST.ICMP engine parameters.

Table A-39 SWEEP.HOST.ICMP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
IcmpType	NUMBER (0–255)	No	No	The IcmpType of interest for this sweep signature.
Unique	NUMBER (2–40)	No	Yes	The threshold number of unique host connections. The alarm fires when the unique number of host connections is exceeded during the interval.

SWEEP.HOST.TCP Engine Parameters

Table A-40 lists the SWEEP.HOST.TCP engine parameters.

Table A-40 *SWEEP.HOST.TCP Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
Mask	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The mask used in TcpFlags comparison.
TcpFlags	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The TCP Flags to match when masked by Mask.
Unique	NUMBER (2–40)	No	Yes	The threshold number of unique connections allowed. When exceeded, the alarm fires.

SWEEP.MULTI Engine Parameters

Table A-41 lists the SWEEP.MULTI engine parameters.

Table A-41 *SWEEP.MULTI Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
TcpInterest	NUMBER (1–2)	No	No	The predefined TCP ports of interest: 1 = satan normal, 2 = satan heavy.
UdpInterest	NUMBER (1–2)	No	No	The predefined TCP ports of interest: 1 = satan normal, 2 = satan heavy.
UniqueTCPports	NUMBER (2–40)	No	No	The threshold number of unique TCP port connections allowed until the alarm fires.
UniqueUDPports	NUMBER (2–40)	No	No	The threshold number of unique UDP port connections allowed until the alarm fires.

SWEEP.OTHER.TCP Engine Parameters

Table A-42 lists the SWEEP.OTHER.TCP engine parameters.

Table A-42 *SWEEP.OTHER.TCP Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
PortRange	NUMBER (0–2)	No	No	Which Destination port range to use: 1 = Low Ports, 2 = High Ports, 0 = All.
TcpFlags1	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The TCP flags for equality comparison (1 of 4).
TcpFlags2	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	No	The TCP flags for equality comparison (2 of 4).
TcpFlags3	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	No	The TCP flags for equality comparison (3 of 4).
TcpFlags4	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	No	The TCP flags for equality comparison (4 of 4).

SWEEP.PORT.TCP Engine Parameters

Table A-43 lists the SWEEP.PORT.TCP engine parameters.

Table A-43 *SWEEP.PORT.TCP Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
InvertedSweep	BOOLEAN (True, False)	No	No	Use SRCPORT instead of DSTPORT for Unique counting.
Mask	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The mask used in the TcpFlags comparison.
PortRange	NUMBER (0–2)	No	Yes	The port range to examine: 1 = Low Ports, 2 = High Ports, 0 = All.
SupressReverse	BOOLEAN (True False)	No	No	Do not fire an alarm when a sweep has fired in the reverse direction on this address set.
TcpFlags	BITSET (FIN SYN RST PSH ACK URG ZERO)	No	Yes	The TCP Flags to match when masked by Mask.
Unique	NUMBER (2–40)	No	Yes	The threshold number of unique connections allowed until the alarm fires.

SWEEP.PORT.UDP Engine Parameters

Table A-44 lists the SWEEP.PORT.UDP engine parameters.

Table A-44 *SWEEP.PORT.UDP Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
PortsInclude	STRING	No	No	A comma-separated list of ports or port ranges to inspect for sweeps.
Unique	NUMBER (2–40)	No	Yes	The threshold number of unique port connections between the two hosts.

SYSLOG Engine

The SYSLOG engine analyzes traffic directed at the syslog port (514 UDP). It provides specialized handling of the contents of the syslog data. If the contents of the syslog match the predetermined format for a CISCO ACL policy violation message, the contents of the syslog are used to generate an alert. Any syslog that does not match the ACL format is ignored.

Table A-45 lists the SYSLOG engine parameters.

Table A-45 *SYSLOG Engine Parameters*

Parameter Name	Data Type	Protected	Required	Description
AcldataSource	STRING	No	No	A comma-separated list of IP addresses that are valid sources of ACL violations.
AcldataFilterName	STRING	No	No	The name of the ACL filter.
Facility	NUMBER	No	No	The syslog facility level.
Priority	NUMBER	No	No	The syslog priority level.

TROJAN and TRAFFIC Engines

There are three engines in the TROJAN group:

- TROJAN.BO2K
- TROJAN.TFN2K
- TROJAN.UDP

There is one engine in the TRAFFIC group—TRAFFIC.ICMP.



Caution

You cannot add custom signatures to the TROJAN and TRAFFIC engines. If you try to add custom signatures to the TROJAN and TRAFFIC engines, you receive the following error message: `Error: Array contains max entries, could not add new entry.`

The TRAFFIC and TROJAN engines target nonstandard protocols such as BackOrifice 2000 (BO2K), Tribe Flood Net 2000 (TFN2K), LOKI, and Distributed Denial Of Service (DDOS).

BackOrifice (BO) is the original Windows back door Trojan that runs over UDP. BO2K soon superseded it. BO2K supports UDP and TCP with basic XOR encryption. The TROJAN.UDP engine handles the UDP modes of BO and BO2K. The TROJAN.BO2K engine handles the TCP modes.

TFN2K is the newer version of the Tribe Flood Net (TFN). It is a DDOS agent that is used to control coordinated attacks by infected machines (zombies) to target a single machine (or domain) with bogus traffic floods from hundreds or thousands of unknown attacking hosts. TFN2K is particularly nasty in the randomization of packet header information it sends, but it has two discriminators that can be used to determine the signature. One is that the L3 checksum is incorrect and the other is the remnants of the Base64 encoding it uses and char 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 them.

LOKI is another type of back door Trojan. Once the machine is infected, the malicious code creates an ICMP tunnel that can be used to send small payload in ICMP replies, which can go straight through a firewall if not configured to block ICMP. The signature looks for an imbalance of ICMP echo requests to replies and simple `IcmpCode` and payload discriminators.

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

There are no specific parameters for the TROJAN engines. You can tune the engines in this group by using the master parameters.

[Table A-46](#) lists parameters that are specifically supported by the TRAFFIC.ICMP engine.

Table A-46 TRAFFIC.ICMP Engine Parameters

Parameter Name	Data Type	Protected	Required	Description
ReplyRatio	NUMBER	No	No	The unbalance of replies to requests. It fires the alarm when there are this many more replies than requests.
WantRequest	ENUM (ANY, False, True)	No	No	Requires an ECHO REQUEST to be seen before the alarm is fired. True = must have a request to fire the alarm. False = cannot have a request to fire. ANY = does not care.
isLoki	BOOLEAN (True, False)	No	No	This signature is watching for the original LOKI.
isModLoki	BOOLEAN (True, False)	No	No	This signature is watching for a modified LOKI.