

# TECHNICAL NOTE

## USING EXTENSION DOCUMENTS

JANUARY 2009

Device extensions allow you to repair an event with missing or incorrect information. You can also use device extensions to parse an event when the associated DSM fails to produce a result. Any new events produced by the device extension are associated to the device that failed to parse the original payload. This action prevents these events from appearing as un-parsed in the STRM interface.

Before you define a device extension, you must build an extension document. This document provides information on defining an extension document including:

- [About Extension Documents](#)
- [Understanding Extension Document Elements](#)
- [Creating Extension Documents](#)
- [Device Type IDs](#)



*This document assumes an advanced knowledge of XML coding.*

---

### About Extension Documents

An extension document is specified in Extensible Markup Language (XML) format and can be created or edited using any common word processing application. You can create multiple extension documents and associate an extension document to various device types.

Using XML format requires that all regular expressions be contained in character data (CDATA) sections to prevent the special characters required by regular expressions from interfering with the markup format. For example:

```
<pattern id="Protocol" case-insensitive="true" xmlns="">  
<![CDATA[(tcp|udp|icmp|gre)]]></pattern>
```

Where `(tcp|udp|icmp|gre)` is the actual regular expression pattern.

The configuration consists of two sections: patterns and match groups. For more information, see [Understanding Extension Document Elements](#).

## Understanding Extension Document Elements

This section explains the two main divisions of the extension document:

- [Patterns](#)
- [Match Groups](#)

**Patterns** Rather than associating a regular expression directly with a particular field name, patterns (**patterns**) are declared separately at the top of the extension document and can be subsequently referenced multiple times within the file.

**Table 1** Pattern Parameters

Parameter	Description
<b>id</b> (Required)	Specify a regular string that is unique within the extension document.
<b>case-insensitive</b> (Optional)	Specify if you want the pattern to ignore character case when doing a match, for example <code>abc</code> is the same as <code>ABC</code> .  If not specified, this parameter defaults to false.
<b>trim-whitespace</b> (Optional)	Specify if you want the pattern to ignore white space and carriage returns. If the CDATA sections are split onto different lines, this parameter ensures that any extra spaces and carriage returns are not interpreted as part of the pattern.  If not specified, this parameter defaults to false.

## Match Groups

A match group (**match-group**) is a set of patterns used for parsing or modifying one or more types of events. A matcher is an entity within a match group that is parsed (for example, `EventName`) and is paired with the appropriate pattern and group for parsing. Any number of match groups may appear in the extension document.

**Table 2** Match Group Parameters

Parameter	Description
<b>order</b> (Required)	Specify an integer greater than zero to specify the order in which the match groups are executed. It must be unique within the extension document.
<b>description</b> (Optional)	Specify a description for the match group, which can be any string. This information may appear in the logs.  If not specified, this parameter defaults to empty.
<b>device-type-id-override</b> (Optional)	Specify a different device's QID. Allows the particular match group to search in the specified device for the event type. It must be a valid device type ID, represented as an integer. A list of device type IDs is presented in <a href="#">Table 6</a> .  If not specified, this parameter defaults to the device type of the device to which the extension is attached.

Match groups can have up to three different types of entities:

- [Matcher](#) (`matcher`)
- [Single-Event Modifier](#) (`event-match-single`)
- [Multi-Event Modifier](#) (`event-match-multiple`)

### Matcher (`matcher`)

A matcher entity is a field that is parsed (for example, `EventName`) and is paired with the appropriate pattern and group for parsing. Matchers have an associated order, so if multiple matchers are specified for the same field name, the matchers are executed in the order presented until a successful parse is found or a failure occurs.

**Table 3** Matcher Entity Parameters

Parameter	Description
<code>field</code> (Required)	Specify the field to which you want the pattern to apply, for example, <code>EventName</code> , or <code>SourceIp</code> . See <a href="#">Table 4</a> for a list of valid field names.
<code>pattern-id</code> (Required)	Specify the pattern you want to use when parsing the field out of the payload. This value must match (including case) the ID parameter of the pattern previously defined in a pattern ID parameter ( <a href="#">Table 1</a> ).
<code>order</code> (Required)	Specify the order that you want this pattern to attempt among matchers assigned to the same field. If there are two matchers assigned to the <code>EventName</code> field, the one with the lowest order is attempted first.
<code>capture-group</code> (Optional)	<p>Specify capture group(s), as denoted in the regular expression inside parenthesis ( ). These captures are indexed starting at one and processed from left to right in the pattern. The <code>capture-group</code> field must be a positive integer less than or equal to the number of capture groups contained in the pattern. The default value is zero, which will be the entire match.</p> <p>For example, you can define a single pattern for a source IP address and port; where the <code>SourceIp</code> matcher may use a capture group of 1, and the <code>SourcePort</code> matcher may use a capture group of 2, but only one pattern needs to be defined.</p> <p>This field has a dual purpose when combined with the <code>enable-substitutions</code> parameter.</p>

**Table 3** Matcher Entity Parameters (continued)

Parameter	Description
<b>enable-substitutions</b> (Optional)	<p>Specify this Boolean parameter as <b>true</b> when a field cannot be adequately represented with a straight group capture.</p> <p>You can combine multiple groups together with extra text to form a value. This parameter enables that behavior.</p> <p>This parameter changes the meaning of the <b>capture-group</b> parameter. The <b>capture-group</b> parameter creates the new value, and group substitutions are specified using <b>\x</b> where <b>x</b> is a group number from 1 to 9. You may use groups multiple times, and any free-form text can also be inserted into the value. For example, if you need to form a value out of group 1, followed by an underscore, followed by group 2, an @, and then group 1 again, the appropriate capture-group syntax is:</p> <pre>capture-group="\1_\2@\1"</pre> <p>In another example, a MAC address is separated by colons, but STRM assumes that MAC addresses are hyphen separated. The syntax to parse and capture the individual portions is:</p> <pre>capture-group="\1:\2:\3:\4:\5:\6"</pre> <p>If no groups are specified in the capture-group when substitutions are enabled, a direct text replacement occurs.</p> <p>Default is false.</p>

[Table 4](#) provides a list of valid field names for use in the matcher field parameter (see [Table 3](#)).

**Table 4** Matcher Field Names

Field Name	Description
EventName (Required)	Specify the event name to be retrieved from the QID to identify the event.
EventCategory	<p>Specify an event category for any event with a category not handled by an <b>event-match-single</b> entity or an <b>event-match-multiple</b> entity.</p> <p>Combined with EventName, EventCategory is used to search for the event in the QID.</p>
SourceIp	Specify the source IP address for the message.
SourcePort	Specify the source port for the message.
SourceIpPreNAT	Specify the source IP address for the message before Network Address Translation (NAT) occurs.

**Table 4** Matcher Field Names (continued)

Field Name	Description
SourceIpPostNAT	Specify the source IP address for the message after NAT occurs.
SourceMAC	Specify the source MAC address for the message.
SourcePortPreNAT	Specify the source port for the message before NAT occurs.
SourcePortPostNAT	Specify the source port for the message after NAT occurs.
DestinationIp	Specify the destination IP address for the message.
DestinationPort	Specify the destination port for the message.
DestinationIpPreNAT	Specify the destination IP address for the message before NAT occurs.
DestinationIpPostNAT	Specify the destination IP address for the message after NAT occurs.
DestinationPortPreNAT	Specify the destination port for the message before NAT occurs.
DestinationPortPostNAT	Specify the destination port for the message after NAT occurs.
DestinationMAC	Specify the destination MAC address for the message.
DeviceTime	<p>Specify the time that the event was sent, according to the device (this is NOT the time that the event arrived).</p> <p>STRM detects timestamps in the following formats:</p> <ul style="list-style-type: none"> <li>Valid syslog timestamp in the form of mm dd hh:mm:ss, for example: <b>Jan 13 12:33:10</b></li> <li>Current locale timestamp</li> </ul> <p>Any other formats will not properly convert.</p>
Protocol	<p>Specify the protocol associated with the event; for example, TCP, UDP, or ICMP.</p> <p>If a protocol is not properly parsed out of a message, ports that were parsed may not appear in STRM (it only displays ports for port-based protocols).</p>
UserName	Specify the user name associated with the event.
HostName	Specify the host name associated with the event. Typically, this field is associated with identity events.
GroupName	Specify the group name associated with the event. This field is usually only associated with identity events.
NetBIOSName	Specify the NetBIOS name associated with the event. Typically, this field is associated with identity events.
ExtraIdentityData	Specify any user-specific data associated with the event. Typically, this field is associated with identity events.
SourceIpV6	Specify the IPv6 source IP address for the message.

**Table 4** Matcher Field Names (continued)

Field Name	Description
DestinationIpv6	Specify the IPv6 destination IP address for the message.

**Single-Event Modifier (event-match-single)**

Single-event modifier (**event-match-single**) matches (and subsequently modifies) exactly one type of event, as specified by the required, case-sensitive EventName parameter. This entity allows mutation of successful events by changing the device event category, severity, or the method for sending identity events.

When events matching this event name are parsed, the device category, severity, and identity properties are imposed upon the resulting event. An event-match-single entity consists of three optional properties:

**Table 5** Single-Event Modifier Parameters

Parameter	Description
<b>device-event-category</b>	Specify a new category for searching in the QID for the event. This is an optimizing parameter, since some devices have the same category for all events.
<b>severity</b>	Specify the severity of the event. This parameter must be an integer value between 1 and 10.  If a severity of less than 1 or greater than 10 is specified, the system defaults to 5.  If not specified, the default is whatever is found in the QID.
<b>send-identity</b>	Specifies the sending of identity change information from the event. Choose one of the following options: <ul style="list-style-type: none"> <li>• <b>UseDSMResults</b> – If the DSM returns an identity event, the event is passed on. If the DSM does not return an identity event, the DSM does not create or modify the identity information.  This is the default value if no value is specified.</li> <li>• <b>SendIfAbsent</b> – If the DSM creates identity information, the identity event is passed through unaffected. If no identity event is produced by the DSM, but there is enough information in the event to create an identity event, an event is generated with all the relevant fields set.</li> <li>• <b>OverrideAndAlwaysSend</b> – Ignores any identity event returned by the DSM and creates a new identity event, if there is enough information.</li> <li>• <b>OverrideAndNeverSend</b> – Suppress any identity information returned by the DSM.</li> </ul>

**Multi-Event Modifier (event-match-multiple)**

The multi-event modifier (`event-match-multiple`) matches a range of event types (and subsequently modifies) as specified by the `pattern-id` parameter and the `capture-group-index` parameter.



**Note:** *This match is not run against the payload, but is run against the results of the EventName matcher previously parsed out of the payload.*

This entity allows mutation of successful events by changing the device event category, severity, or the method the event uses to send identity events. The `capture-group-index` must be an integer value (substitutions are not supported) and pattern-ID must reference an existing pattern entity. All other properties are identical to their counterparts in the single-event modifier

---

**Creating Extension Documents**

This section provides you with information on creating extension documents including:

- [Writing a Complete Extension Document](#)
- [Uploading Extension Documents](#)
- [Solving Specific Parsing Issues](#)

## Writing a Complete Extension Document

The example of an extension document included in this section provides information on how to parse one particular type of Cisco FWSM so that events are not sent with an incorrect event name. For example, if you want to resolve the word `session`, which is embedded in the middle of the event name:

```
Nov 17 09:28:26 129.15.126.6 %FWSM-session-0-302015: Built UDP
connection for faddr 38.116.157.195/80 gaddr
129.15.127.254/31696 laddr 10.194.2.196/2157 duration 0:00:00
bytes 57498 (TCP FINs)
```

This condition causes the DSM to not recognize any events and all the events are un-parsed and associated with the generic logger.

Although only a portion of the text string (302015) is used for the QID search, the entire text string (%FWSM-session-0-302015) identifies the event as coming from a Cisco FWSM. Since the entire text string is invalid, the DSM assumes that the event is invalid.

A FWSM device has a large number of event types, many with unique formats. The following extension document example, indicates how to parse one event type.



*The pattern IDs do not have to match the field names that they are parsing. Even though the following example duplicates the pattern, the `SourceIp` field and the `SourceIpPreNAT` field could use the exact same pattern in this case (this may not be true in all FWSM events).*

```
<?xml version="1.0" encoding="UTF-8"?>
<device-extension xmlns="event_parsing/device_extension">

<pattern id="EventNameFWSM" xmlns=""><![CDATA[%FWSM[a-zA-Z\-\]*\d-(\d{1,6})]]></pattern>
<pattern id="SourceIp" xmlns=""><![CDATA[gaddr
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/([\d]{1,5})]]></pattern>
<pattern id="SourceIpPreNAT" xmlns=""><![CDATA[gaddr
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/([\d]{1,5})]]></pattern>
<pattern id="SourceIpPostNAT" xmlns=""><![CDATA[laddr
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/([\d]{1,5})]]></pattern>
<pattern id="DestinationIp" xmlns=""><![CDATA[faddr
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})/([\d]{1,5})]]></pattern>
<pattern id="Protocol" case-insensitive="true"
xmlns=""><![CDATA[(tcp|udp|icmp|gre)]]></pattern>
<pattern id="Protocol_6" case-insensitive="true" xmlns=""><![CDATA[ protocol=6]]></pattern>
<pattern id="EventNameId" xmlns=""><![CDATA[(\d{1,6})]]></pattern>

<match-group order="1" description="FWSM Test" device-type-id-override="6" xmlns="">
  <matcher field="EventName" order="1" pattern-id="EventNameFWSM" capture-group="1"/>
  <matcher field="SourceIp" order="1" pattern-id="SourceIp" capture-group="1" />
  <matcher field="SourcePort" order="1" pattern-id="SourceIp" capture-group="2" />
  <matcher field="SourceIpPreNAT" order="1" pattern-id="SourceIpPreNAT" capture-group="1" />
  <matcher field="SourceIpPostNAT" order="1" pattern-id="SourceIpPostNAT" capture-group="1" />
  <matcher field="SourcePortPreNAT" order="1" pattern-id="SourceIpPreNAT" capture-group="2" />
  <matcher field="SourcePortPostNAT" order="1" pattern-id="SourceIpPostNAT" capture-group="2" />
  <matcher field="DestinationIp" order="1" pattern-id="DestinationIp" capture-group="1" />
</match-group>
```

```

<matcher field="DestinationPort" order="1" pattern-id="DestinationIp" capture-group="2" />
<matcher field="Protocol" order="1" pattern-id="Protocol" capture-group="1" />
<matcher field="Protocol" order="2" pattern-id="Protocol_6" capture-group="TCP"
enable-substitutions="true"/>

<event-match-multiple pattern-id="EventNameId" capture-group-index="1"
device-event-category="Cisco Firewall"/>
</match-group>

</device-extension>

```

The above extension document example demonstrates some of the basic aspects of parsing:

- IP addresses
- Ports
- Protocol
- Multiple fields using the same pattern with different groups

This example parses all FWSM events that follow the specified pattern, although the fields that are parsed may not be present in those events (if the events include different content).

The information that was necessary to create this configuration that was not available from the event:

- The event name is only the last six digits (302015) of the `%FWSM-session-0-302015` portion of the event.
- The FWSM has a hard-coded device type category of `Cisco Firewall`.
- The FWSM uses the Cisco Pix QID and therefore includes the `device-type-id-override="6"` parameter in the match group (the Pix firewall's device type ID is 6, see [Table 6](#)).



*If the QID information is not specified or is unavailable, you can modify the event mapping using the Event Viewer. For more information, see the Modifying Event Mapping section in the STRM Users Guide.*

An event name and a device event category is required when looking for the event in the QID. This device event category is a grouping parameter within the database that helps define like events within a device. The `event-match-multiple` at the end of the match group includes hard-coding of the category. The `event-match-multiple` uses the `EventNameId` pattern on the parsed event name to match up to six digits. This pattern is not run against the full payload, just that portion parsed as the `EventName` field.

The `EventName` pattern references the `%FWSM` portion of the events; all Cisco FWSM events contain the `%FWSM` portion. The pattern in the example matches `%FWSM` followed by any number (zero or more) of letters and dashes. This pattern match resolves the word `session` that is embedded in the middle of the event name that needs to be removed. The event severity (according to Cisco), followed by

a dash and then the true event name as expected by STRM. The only string with a capture group (that is, bounded by parenthesis) is this pattern of digits (`(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})`).

The IP addresses and ports for the event all follow the same basic pattern: an IP address followed by a slash followed by the numeric port number. This pattern parses two pieces of data (the IP address and the port), and specifies different capture groups in the matcher section.

```
<matcher field="SourceIp" order="1" pattern-id="SourceIp" capture-group="1" />
<matcher field="SourcePort" order="1" pattern-id="SourceIp" capture-group="2" />
```

Thus, the IP address/port patterns are four sets of one to three digits, separated by periods followed by a slash and the port number. The IP address section is in a group, as is the port number (but not the slash). As was previously mentioned, the matcher sections for these fields reference the same pattern name, but a different capture group (the IP address is group 1 and the port is group 2).

The protocol is a common pattern that searches the payload for the first instance of TCP, UDP, ICMP, or GRE (the pattern is marked with the case-insensitive parameter so that any occurrence matches).



**Note:** You must search for the protocol when writing extension documents, as STRM may not display port numbers if the event is not based on a port-based protocol. See [Converting a Protocol](#) for an example of how to search for a protocol.

Although a second protocol pattern does not occur in the event being used as an example, there is a second protocol pattern defined with an order of two. If the lowest-ordered protocol pattern does not match, the next one is attempted (and so on). The second protocol pattern also demonstrates the concept of a direct substitution; there are no match groups in the pattern, but with the `enable-substitutions` parameter enabled, the text TCP can be used in place of `protocol=6`.

## Uploading Extension Documents

Multiple extension documents can be created, uploaded, and associated to various device types. Extension documents can be stored anywhere prior to uploading to STRM. When you select an extension document for uploading, STRM validates the document against the internal XSD. STRM also verifies the validity of the document before uploading to the system. For more information about device extensions, see the *Managing Sensor Devices Guide*.

## Solving Specific Parsing Issues

This section provides you with XML examples that can be used when resolving specific parsing issues that may arise:

- [Converting a Protocol](#)
- [Making a Single Substitution](#)
- [Generating Colon-Separated MAC Address](#)
- [Combining IP Address and Port](#)
- [Modifying an Event Category](#)
- [Modifying Multiple Event Categories](#)
- [Suppressing Identity Change Events](#)
- [Encoding Logs](#)

### Converting a Protocol

The following example shows a typical protocol conversion that searches for TCP, UDP, ICMP, or GRE anywhere in the payload, surrounded by any word boundary (for example, tab, space, end-of-line). Also, character case is ignored:

```
<pattern id="Protocol" case-insensitive="true"
xmlns=""><![CDATA[\b(tcp|udp|icmp|gre)\b]]> </pattern>
<matcher field="Protocol" order="1" pattern-id="Protocol" capture-group="1" />
```

### Making a Single Substitution

The following is an example of a straight substitution that parses the source IP address, and then overrides the result and sets the IP address to 10.100.100.100, ignoring the IP address in the payload. This example assumes that the source IP address matches something similar to SrcAddress=10.3.111.33 followed by a comma:

```
<pattern id="SourceIp_AuthenOK" xmlns="">
<![CDATA[SrcAddress=(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}),]]></pattern>

<matcher field="SourceIp" order="1" pattern-id="SourceIp_AuthenOK"
capture-group="100.100.100.100" enable-substitutions="true"/>
```

### Generating Colon-Separated MAC Address

STRM detects MAC addresses in a colon-separated form. Since all devices do not use this form, the following example shows how to correct that situation:

```
<pattern id="SourceMACWithDashes"
xmlns=""><![CDATA[SourceMAC=([0-9a-fA-F]{2})-([0-9a-fA-F]{2})-([0-9a-fA-F]{2})-([0-9a-fA-F]{2})-([0-9a-fA-F]{2})-([0-9a-fA-F]{2})]]></pattern>

<matcher field="SourceMAC" order="1" pattern-id=" SourceMACWithDashes"
capture-group="\1:\2:\3:\4:\5:\6" />
```

In the above example `sourceMAC=12-34-56-78-90-AB` is converted to a MAC address of `12:34:56:78:90:AB`.

If the dashes are removed from the pattern, the pattern converts a MAC address with no separators. If spaces are inserted, the pattern converts a space-separated MAC address, and so on.

### Combining IP Address and Port

Typically an IP address and port are combined in one field, separated by a colon or a slash. The following example uses multiple capture groups with one pattern:

```
pattern id="SourceIPColonPort" xmlns=""><![CDATA[Source=(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):([\d]{1,5})]]></pattern>
<matcher field="SourceIp" order="1" pattern-id="SourceIPColonPort" capture-group="1" />
<matcher field="SourcePort" order="1" pattern-id="SourceIPColonPort" capture-group="2" />
```

### Modifying an Event Category

A device event category may be hard-coded, or the severity needs to be adjusted. The following example adjusts the severity for a single event type:

```
<event-match-single event-name="TheEvent" device-event-category="Actual Category" severity="6" send-identity="UseDSMResults" />
```

### Modifying Multiple Event Categories

The following example is similar to the above single event example, except that this example matches all event codes starting with 7 and followed by one to five digits:

```
<pattern id="EventNameId" xmlns=""><![CDATA[(7\d{1,5})]]></pattern>
<event-match-multiple pattern-id="EventNameId" capture-group-index="1" device-event-category="Actual Category" severity="6" send-identity="UseDSMResults"/>
```

### Suppressing Identity Change Events

A DSM may unnecessarily send identity change events. The following is two examples; one is a method of how to suppress identity change events from being sent from a single event type. The other is a method of how to suppress identity change events from being sent from a group of events.

```
// Never send identity for the event with an EventName of "Authen OK"
<event-match-single event-name="Authen OK" device-event-category="ACS" severity="6" send-identity="OverrideAndNeverSend" />

// Never send any identity for an event with an event name starting with 7, followed by one to five other digits:
<pattern id="EventNameId" xmlns=""><![CDATA[(7\d{1,5})]]></pattern>
<event-match-multiple pattern-id="EventNameId" capture-group-index="1" device-event-category="Cisco Firewall" severity="7" send-identity="OverrideAndNeverSend"/>
```

## Encoding Logs

The following encoding formats are supported:

- US-ASCII
- UTF-8

Logs may be forwarded to the system in an encoding that does not match US-ASCII or UTF-8 formats. You can configure an advanced flag to ensure input can be re-encoded to UTF-8 for parsing and storage purposes.

For example, if you want to ensure that the source logs arrive in SHIFT-JIS (ANSI/OEM Japanese) encoding, enter the following:

```
<device-extension source-encoding="SHIFT-JIS"
xmlns="event_parsing/device_extension">
```

The logs will be enclosed in UTF-8 format.

## Device Type IDs

[Table 6](#) lists the Device Type IDs that can be used in a `match-group` statement:

**Table 6** Device Type ID Numbers

ID	Device	Description
2	Snort	Snort Open Source IDS
3	CheckPoint	CheckPoint Firewall-1 Devices using syslog
4	GenericFirewall	Configurable Firewall Filter
5	NetScreenFirewall	NetScreen Firewall Appliance
6	Pix	Cisco PIX Firewall
7	GenericAuthServer	Configurable Authentication message filter
8	SOAP	Webservice-based messages
9	Dragon	Enterasys Dragon IDS
10	Apache	Open Source Apache Webserver
11	LinuxServer	Linux login messages
12	WindowsAuthServer	Windows Security Event Log
13	IIS	Windows IIS Webserver logs
14	Iptables	Linux iptables Firewall
15	Proventia	ISS Proventia Device
16	Classify	Q1Labs Classify Engine
17	NetScreenIDP	NetScreen IDP
18	EventCRE	Event CRE
19	UnityOne	Tipping Point UnityOne
20	IOS	Cisco IOS

**Table 6** Device Type ID Numbers (continued)

<b>ID</b>	<b>Device</b>	<b>Description</b>
21	Contivity	Nortel Contivity VPN Switch
22	ARN	Nortel BayRS 15.x
23	VpnConcentrator	VPN 3000 Concentrator
24	Solaris	Solaris Login Messages
25	IntruShield	McAfee IntruShield Network IPS Appliance
26	CSA	Cisco CSA
28	Enterasys	Enterasys Matrix E1 Switch
29	Sendmail	Solaris Sendmail
30	IDS	Cisco IDS
31	Cisco FWSM	Cisco FWSM
33	SiteProtector	ISS SiteProtector
35	Cyberguard	Cyberguard
36	JuniperSA	Juniper Secure Access
37	Contivityv2	Nortel Contivity VPN Switch
38	TopLayerIPS	TopLayer IPS
39	Generic	Generic DSM
40	Tripwire	Tripwire
41	ASA	Cisco ASA
42	Niksun	Niksun
43	Sidewinder	Secure Computing - Sidewinder
45	NetscreenNSM	Netscreen NSM
46	WebProxy	Squid WebProxy
47	IpAngel	Lucid IpAngel
48	OracleDbAudit	Oracle Database Audit Records
49	BigIP	F5 Networks BigIP
50	SolarisDhcpd	Solaris DHCP Transaction Logs
55	ArrayVPN	Array VPN
56	CiscoCatOS	Cisco CatOS
57	ProFTPD	ProFTPD server
58	LinuxDHCPsServer	Linux DHCP Server
59	InfranetController	Juniper Infranet Controller
60	PDSN	Sprint PoC PDSN
61	RNC	Sprint PoC RNC
62	BTS	Sprint PoC BTS
63	ACS	Cisco ACS
64	JuniperRouter	Juniper Router

**Table 6** Device Type ID Numbers (continued)

<b>ID</b>	<b>Device</b>	<b>Description</b>
65	Sprint	Sprint PoC
66	CallManager	Cisco Call Manager
67	GenericLog	Generic Log DSM+C46
68	Nseries	Enterasys N Series Router/Switch
70	ExtremeWare	ExtremeWare 7.7
71	Sidewinder	Secure Computing - Sidewinder
73	FortiGuard	Fortinet FortiGuard
75	SymantecAV	Symantec Anti Virus
78	SonicWall	SonicWall Firewall
79	Vericept	Vericept Content 360
80	WindowsAgent	Q1 Windows Agent Syslog
82	SymantecFirewall	Symantec Firewall
83	JuniperSBR	Juniper Steel-Belted Radius
85	IBM AIX	IBM AIX 5L Operating System
86	metainfo Meta IP	metainfo Meta IP
87	SymantecSystemCenter	Symantec System Center JDBC/WinAgent
90	Cisco ACS	Cisco ACS
91	Sentinel	Enterasys Sentinel
92	CounterACT	Forescount CounterACT
93	McAfee ePO	McAfee ePolicy Orchestrator
94	Cisco CSA-syslog	Cisco CSA using syslog
95	Cisco NAC	Cisco NAC Appliance
96	TippingPointx505	3com Tippingpoint x5 series
97	MicrosoftDHCP	Microsoft DHCP server
98	MicrosoftIAS	Microsoft IAS server
99	MicrosoftExchange	Microsoft Exchange server
100	TrendISVW	Trend Interscan Virus wall
101	MsSQL	Microsoft SQL Server
102	AppleOSX	Apple OSX system/fw events
103	Bluecoat	Bluecoat SG Appliance
104	Firewall6000	Nortel Switched Firewall 6000
105	Juniper Networks STRM	Juniper Networks STRM
106	3Com 8800 Series Switch	3Com 8800 Series Switch
107	Nortel VPN Gateway	Nortel VPN Gateway
108	NortelTPS	Nortel Threat Protection Intrusion Sensor
109	Vis Host Events	Vis New Host, Port, Vuln Notifications

**Table 6** Device Type ID Numbers (continued)

<b>ID</b>	<b>Device</b>	<b>Description</b>
110	NortelNAS	Nortel Application Switch
111	JuniperDX	Juniper DX accelerator
112	SNAREReflector	SNARE Reflector
113	Cisco 12000 Series	Cisco 12000 Series
114	Cisco 6500 Series	Cisco 6500 Series
115	Cisco 7600 Series	Cisco 7600 Series
116	Cisco CRS Series	Cisco CRS Series
117	Cisco ISR Series	Cisco ISR Series
118	Juniper M Series	Juniper M Series
119	Third Brigade	Third Brigade Deep Security Manager
120	Firewall5100	Nortel Switched Firewall 5100
121	JunOS	JunOS
122	Juniper MX Series	Juniper MX Series
123	Juniper T Series	Juniper T Series

**Copyright Notice**

Copyright © 2008 Juniper Networks, Inc. All rights reserved. Juniper Networks and the Juniper Networks logo are registered trademarks of Juniper Networks Inc. in the United States and other countries. All other trademarks, service marks, registered trademarks, or registered service marks in this document are the property of Juniper Networks or their respective owners. All specifications are subject to change without notice. Juniper Networks assumes no responsibility for any inaccuracies in this document or for any obligation to update information in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Part Number 530-028627-01