



System Architecture

This appendix describes the IPS system architecture, and contains the following sections:

- [IPS System Design, page A-1](#)
- [System Applications, page A-3](#)
- [Security Features, page A-5](#)
- [MainApp, page A-6](#)
- [SensorApp, page A-22](#)
- [CollaborationApp, page A-27](#)
- [SwitchApp, page A-29](#)
- [CLI, page A-30](#)
- [Communications, page A-31](#)
- [Cisco IPS File Structure, page A-34](#)
- [Summary of Cisco IPS Applications, page A-35](#)

Understanding the IPS System Architecture

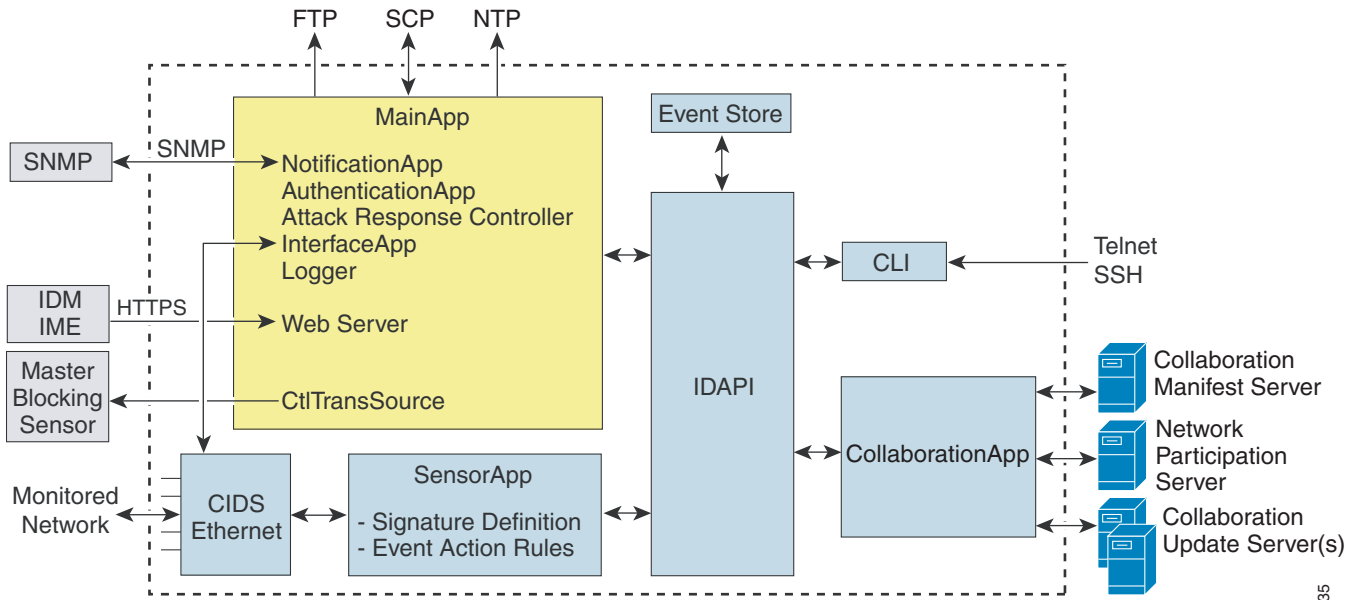
The purpose of the Cisco IPS is to detect and prevent malicious network activity. You can install the Cisco IPS software on two platforms: appliances and the modules. The Cisco IPS contains a management application and a monitoring application. The IDM is a network management JAVA application that you can use to manage and monitor the IPS. The IME is an IPS network monitoring JAVA application that you can use to view IPS events. The IME also contains the IDM configuration component. The IDM and the IME communicate with the IPS using HTTP or HTTPS and are hosted on your computer.

IPS System Design

The Cisco IPS software runs on the Linux operating system. We have hardened the Linux OS by removing unnecessary packages from the OS, disabling unused services, restricting network access, and removing access to the shell.

Figure A-1 illustrates the system design for IPS software.

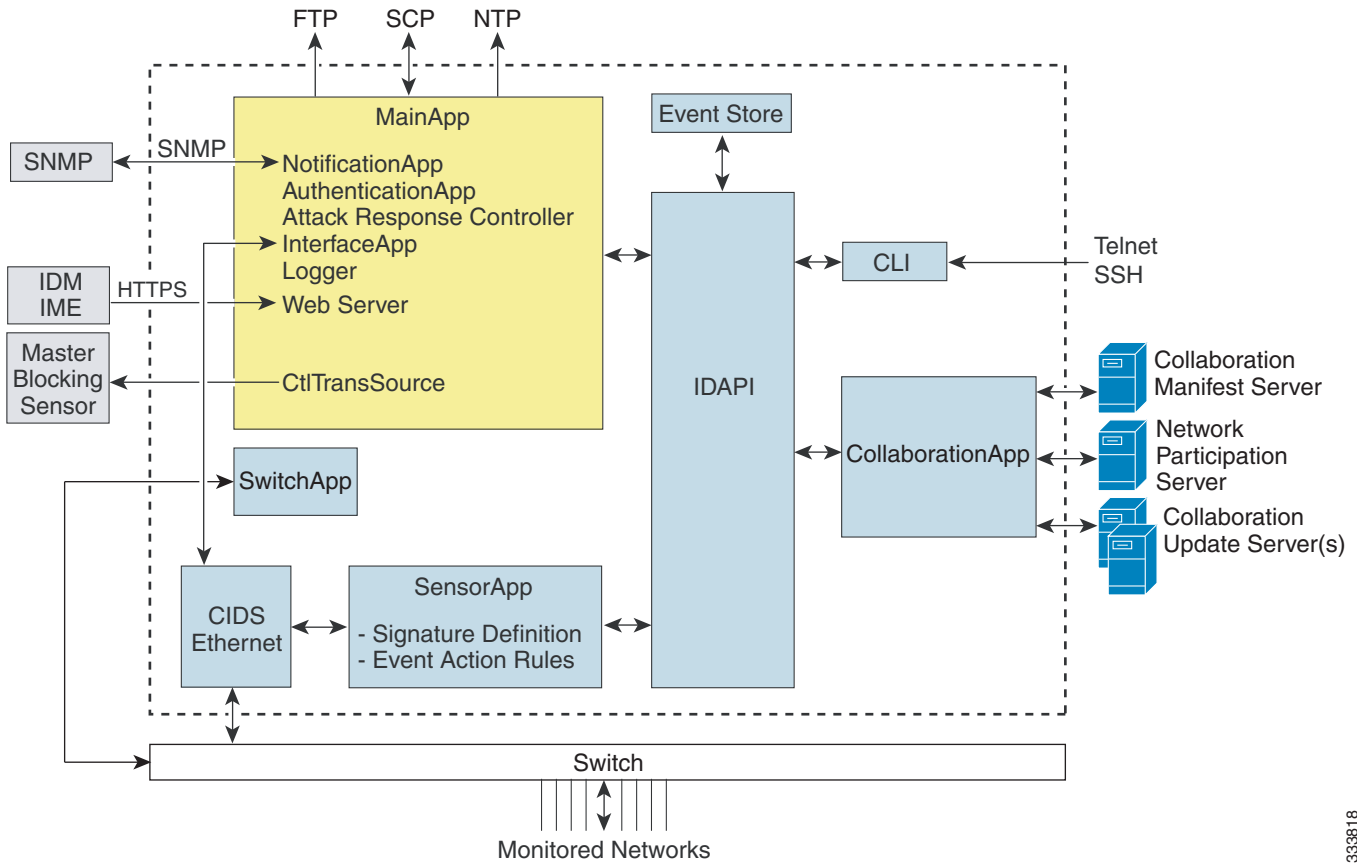
Figure A-1 System Design for the IPS



251235

Figure A-2 illustrates the system design for IPS software for the IPS 4500 series sensors.

Figure A-2 System Design for IPS 4500 Series Sensors



333818

For More Information

- For detailed information about the MainApp, see [MainApp, page A-6](#).
- For detailed information about the SensorApp, see [SensorApp, page A-22](#).
- For detailed information about the CollaborationApp, see [CollaborationApp, page A-27](#).
- For detailed information about the CLI, see [CLI, page A-30](#).
- For detailed information about the SwitchApp, see [SwitchApp, page A-29](#).

System Applications



Note

Each application has its own configuration file in XML format.

The Cisco IPS software includes the following applications:

- **MainApp**—Initializes the system, starts and stops the other applications, configures the OS, and performs upgrades. It contains the following components:
 - **ctlTransSource** (Control Transaction server)—Allows sensors to send control transactions. This is used to enable the master blocking sensor capability of Attack Response Controller (formerly known as Network Access Controller).
 - **Event Store**—An indexed store used to store IPS events (error, status, and alert system messages) that is accessible through the CLI, IDM, IME, ASDM, or SDEE.



Note The Event Store has a fixed size of 30 MB for all platforms.

- **InterfaceApp**—Handles bypass and physical settings and defines paired interfaces. Physical settings are speed, duplex, and administrative state.
- **Logger**—Writes all the log messages of the application to the log file and the error messages of the application to the Event Store.
- **Attack Response Controller** (formerly known as Network Access Controller) —Manages remote network devices (firewalls, routers, and switches) to provide blocking capabilities when an alert event has occurred. The ARC creates and applies ACLs on the controlled network device or uses the **shun** command (firewalls).
- **NotificationApp**—Sends SNMP traps when triggered by alert, status, and error events. The NotificationApp uses the public domain SNMP agent. SNMP GETs provide information about the general health of the sensor.
- **Web server** (HTTP SDEE server)—Provides a web interface and communication with the other IPS devices through the SDEE protocol using several servlets to provide the IPS services.
- **AuthenticationApp**—Verifies that users are authorized to perform CLI, IDM, IME, ASDM, or SDEE actions.
- **SensorApp** (Analysis Engine)—Performs packet capture and analysis.
- **CollaborationApp**—Interfaces with the MainApp and the SensorApp using various interprocess communication technologies including IDAPI control transactions, semaphores, shared memory, and file exchange.
- **CLI**—The interface that is run when you successfully log in to the sensor through Telnet or SSH. All accounts created through the CLI will use the CLI as their shell (except the service account—only one service account is allowed). Allowed CLI commands depend on the privilege of the user.

All Cisco IPS applications communicate with each other through a common API called the IDAPI. Remote applications (other sensors, management applications, and third-party software) communicate with sensors through the SDEE protocol.

The sensor has the following partitions:

- **Application partition**—A full IPS system image.
- **Recovery partition**—A special purpose image used for recovery of the sensor. Booting into the recovery partition enables you to completely reimage the application partition. Network settings are preserved, but all other configuration is lost.

You interact with the Cisco IPS in the following ways:

- Configure device parameters

You generate the initial configuration for the system and its features. This is an infrequent task, usually done only once. The system has reasonable default values to minimize the number of modifications you must make. You can configure Cisco IPS through the CLI, IDM, IME, CSM, ASDM, or through another application using SDEE.

- Tune

You make minor modifications to the configuration, primarily to Analysis Engine, which is the portion of the application that monitors network traffic. You can tune the system frequently after initially installing it on the network until it is operating efficiently and only producing information you find useful. You can create custom signatures, enable features, or apply a service pack or signature update. You can tune Cisco IPS through the CLI, IDM, IME, CSM, ASDM, or through another application using SDEE.

- Update

You can schedule automatic updates or apply updates immediately to the applications and signature data files. You can update Cisco IPS through the CLI, IDM, IME, CSM, ASDM, or through another application using SDEE.

- Retrieve information

You can retrieve data (status messages, errors, and alerts) from the system through the CLI, IDM, IME, CSM, ASDM, CS MARS or another application using SDEE.

For More Information

For detailed information about SDEE, see [SDEE, page A-33](#).

Security Features

Cisco IPS has the following security features:

- Network access is restricted to hosts who are specifically allowed access.
- All remote hosts who attempt to connect through the web server, SSH and SCP or Telnet will be authenticated.
- By default Telnet access is disabled. You can choose to enable Telnet.
- By default SSH access is enabled.
- An FTP server does not run on the sensor. You can use SCP to remotely copy files.
- By default the web server uses TLS or SSL. You can choose to disable TLS and SSL.
- Unnecessary services are disabled.
- Only the SNMP set required by the Cisco MIB Police is allowed within the CISCO-CIDS-MIB. OIDs implemented by the public domain SNMP agent will be writeable when specified by the MIB.

For More Information

For detailed information about SNMP and Cisco MIBs, see [Chapter 15, “Configuring SNMP”](#)

MainApp

This section describes the MainApp, and contains the following topics:

- [Understanding the MainApp, page A-6](#)
- [MainApp Responsibilities, page A-6](#)
- [Event Store, page A-7](#)
- [NotificationApp, page A-9](#)
- [CtlTransSource, page A-11](#)
- [Attack Response Controller, page A-12](#)
- [Logger, page A-19](#)
- [AuthenticationApp, page A-20](#)
- [Web Server, page A-22](#)

Understanding the MainApp

The MainApp includes all IPS components except SensorApp and the CLI. It is loaded by the operating system at startup and loads SensorApp. The MainApp then brings the following subsystem components up:

- Authentication
- Logger
- ARC
- Web Server
- Notification (SNMP)
- External Product Interface
- Interface manager
- Event Store
- Health and security monitoring

MainApp Responsibilities

The MainApp has the following responsibilities:

- Validate the Cisco-supported hardware platform
- Report software version and PEP information
- Start, stop, and report the version of the IPS components
- Configure the host system settings
- Manage the system clock
- Manage the Event Store
- Install and uninstall software upgrades



Note In the Cisco IPS, the MainApp can automatically download signature and signature engine updates from Cisco.com.

- Shut down or reboot the operating system

The MainApp responds to the **show version** command by displaying the following information:

- Sensor build version
- MainApp version
- Version of each running application
- Version and timestamp of each installed upgrade
- Next downgrade version of each installed upgrade
- Platform version
- Version of sensor build on the other partition

The MainApp also gathers the host statistics and reports the health and security monitoring status.

Event Store

This section describes the Event Store, and contains the following topics:

- [Understanding the Event Store, page A-7](#)
- [Event Data Structures, page A-8](#)
- [IPS Events, page A-9](#)

Understanding the Event Store



Note The Event Store has a fixed size of 30 MB for all platforms.

Each IPS event is stored in the Event Store with a time stamp and a unique, monotonic, ascending ID. This time stamp is the primary key used to index the event into the fixed-size, indexed Event Store. When the circular Event Store has reached its configured size, the oldest event or events are overwritten by the new event being stored. The SensorApp is the only application that writes alert events into the Event Store. All applications write log, status, and error events into the Event Store.

The fixed-sized, indexed Event Store allows simple event queries based on the time, type, priority, and a limited number of user-defined attributes. If each event is assigned a priority of low, medium, or high, a single event query can specify a list of desired event types, intrusion event priorities, and a time range.

Table A-1 shows some examples:

Table A-1 IPS Event Examples

IPS Event Type	Intrusion Event Priority	Start Time Stamp Value	Stop Time Stamp Value	Meaning
status	—	0	Maximum value	Get all status events that are stored.
error status	—	0	65743	Get all error and status events that were stored before time 65743.
status	—	65743	Maximum value	Get status events that were stored at or after time 65743.
intrusion attack response	low	0	Maximum value	Get all intrusion and attack response events with low priority that are stored.
attack response error status intrusion	medium high	4123000000	4123987256	Get attack response, error, status, and intrusion events with medium or high priority that were stored between time 4123000000 and 4123987256.

The size of the Event Store allows sufficient buffering of the IPS events when the sensor is not connected to an IPS event consumer. Sufficient buffering depends on your requirements and the capabilities of the nodes in use. The oldest events in the circular buffer are replaced by the newest events.

Event Data Structures

The various functional units communicate the following seven types of data:

- Intrusion events—Produced by the SensorApp. The sensor detects intrusion events.
- Error events—Caused by hardware or software malfunctions.
- Status events—Reports of a change in the status of the application, for example, that its configuration has been updated.
- Control transaction log events—The sensor logs the result of a control transaction.
- Attack response events—Actions for the ARC, for example, a block request.
- Debug events—Highly detailed reports of a change in the status of the application used for debugging.
- Control transaction data—Data associated with control transactions, for example, diagnostic data from an application, session logs, and configuration data to or from an application.

All seven types of data are referred to collectively as *IPS data*. The six event types—intrusion, error, status, control transaction log, network access, and debug—have similar characteristics and are referred to collectively as *IPS events*. IPS events are produced by the several different applications that make up the IPS and are subscribed to by other IPS applications. IPS events have the following characteristics:

- They are spontaneously generated by the application instances configured to do so. There is no request from another application instance to generate a particular event.
- They have no specific destination. They are stored and then retrieved by one or more application instances.

Control transactions involve the following types of requests:

- Request to update the configuration data of an application instance
- Request for the diagnostic data of an application instance
- Request to reset the diagnostic data of an application instance
- Request to restart an application instance
- Request for ARC, such as a block request

Control transactions have the following characteristics:

- They always consist of a request followed by a response.

The request and response may have an arbitrary amount of data associated with them. The response always includes at least a positive or negative acknowledgment.

- They are point-to-point transactions.

Control transactions are sent by one application instance (the initiator) to another application instance (the responder).

IPS data is represented in XML format as an XML document. The system stores user-configurable parameters in several XML files.

IPS Events

IPS applications generate IPS events to report the occurrence of some stimulus. The events are the data, such as the alerts generated by SensorApp or errors generated by any application. Events are stored in a local database known as the Event Store.

There are five types of events:

- evAlert—Alert event messages that report when a signature is triggered by network activity.
- evStatus—Status event messages that report the status and actions of the IPS applications.
- evError—Error event messages that report errors that occurred while attempting response actions.
- evLogTransaction—Log transaction messages that report the control transactions processed by each sensor application.
- evShunRqst—Block request messages that report when ARC issues a block request.

You can view the status and error messages using the CLI, IME, and ASDM. The SensorApp and ARC log response actions (TCP resets, IP logging start and stop, blocking start and stop, trigger packet) as status messages.

NotificationApp

The NotificationApp allows the sensor to send alerts and system error messages as SNMP traps. It subscribes to events in the Event Store and translates them into SNMP MIBs and sends them to destinations through a public-domain SNMP agent. The NotificationApp supports sending sets and gets. The SNMP GETs provide information about basic sensor health.

The NotificationApp has been modified in IPS 7.2(2)E4 and later to support SNMPv3. SNMPv3 configuration, such as access control (rwuser and rouser) security level (authPriv, noAuthNoPriv, authNoPriv), authentication protocol (SHA or none), and privacy protocol (AES or none), has been added. The administrator can also associate SNMPv3 users with trap destinations.

The NotificationApp sends the following information from the evAlert event in sparse mode:

- Originator information
- Event ID
- Event severity
- Time (UTC and local time)
- Signature name
- Signature ID
- Subsignature ID
- Participant information
- Alarm traits

The NotificationApp sends the following information from the evAlert event in detail mode:

- Originator information
- Event ID
- Event severity
- Time (UTC and local time)
- Signature name
- Signature ID
- Subsignature ID
- Version
- Summary
- Interface group
- VLAN
- Participant information
- Actions
- Alarm traits
- Signature
- IP log IDs

The NotificationApp determines which evError events to send as a trap according to the filter that you define. You can filter based on error severity (error, fatal, and warning). The NotificationApp sends the following information from the evError event:

- Originator information
- Event ID
- Event severity
- Time (UTC and local time)
- Error message

The NotificationApp supports GETs for the following general health and system information from the sensor:

- Packet loss
- Packet denies
- Alarms generated
- Fragments in FRP
- Datagrams in FRP
- TCP streams in embryonic state
- TCP streams in established state
- TCP streams in closing state
- TCP streams in system
- TCP packets queued for reassembly
- Total nodes active
- TCP nodes keyed on both IP addresses and both ports
- UDP nodes keyed on both IP addresses and both ports
- IP nodes keyed on both IP addresses
- Sensor memory critical stage
- Interface status
- Command and control packet statistics
- Fail-over state
- System uptime
- CPU usage
- Memory usage for the system
- PEP



Note Not all IPS platforms support PEP.

The NotificationApp provides the following statistics:

- Number of error traps
- Number of event action traps
- Number of SNMP GET requests
- Number of SNMP SET requests

CtlTransSource

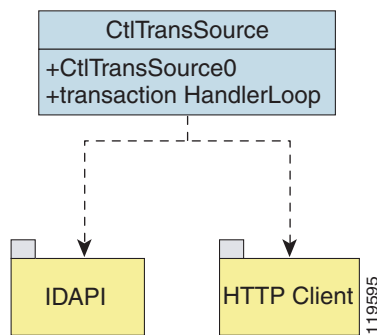
The CtlTransSource is an application that forwards locally initiated remote control transactions to their remote destinations using HTTP protocol. The CtlTransSource initiates either TLS or non-TLS connections and communicates remote control transactions to HTTP servers over these connections.

The CtlTransSource must establish sufficient credentials on the remote HTTP server to execute a remote control transaction. It establishes its credentials by presenting an identity to the HTTP server on the remote node in the form of a username and password (basic authentication). When the authentication is successful, the requestor is assigned a cookie containing a user authentication that must be presented with each request on that connection.

The transactionHandlerLoop method in the CtlTransSource serves as a proxy for remote control transaction. When a local application initiates a remote control transaction, IDAPI initially directs the transaction to the CtlTransSource. The transactionHandlerLoop method is a loop that waits on remote control transactions that are directed to the CtlTransSource.

Figure A-3 shows the transactionHandlerLoop method in the CtlTransSource.

Figure A-3 CtlTransSource



When the transactionHandlerLoop receives a remotely addressed transaction, it tries to forward the remote control transaction to its remote destination. The transactionHandlerLoop formats the transaction into a control transaction message. The transactionHandlerLoop uses the HttpClient classes to issue the control transaction request to the HTTP server on the remote node. The remote HTTP server handles the remote control transaction and returns the appropriate response message in an HTTP response. If the remote HTTP server is an IPS web server, the web server uses the CtlTransSource servlet to process the remote control transactions.

The transactionHandlerLoop returns either the response or a failure response as the response of the control transaction to the initiator of the remote control transaction. If the HTTP server returns an unauthorized status response (indicating the HTTP client has insufficient credentials on the HTTP server), the transactionHandlerLoop reissues the transaction request using the designated username and password of the CtlTransSource to authenticate the identity of the requestor. The transactionHandlerLoop continues to loop until it receives a control transaction that directs it to exit or until its exit event is signaled.

Attack Response Controller

This section describes the ARC, which is the IPS application that starts and stops blocking on routers, switches, and firewalls, and rate limits traffic on routers running Cisco IOS 12.3. A *block* is an entry in the configuration or ACL of a device to block incoming and outgoing traffic for a specific host IP address or network address. This section contains the following topics:

- [Understanding the ARC, page A-13](#)
- [ARC Features, page A-14](#)
- [Supported Blocking Devices, page A-15](#)

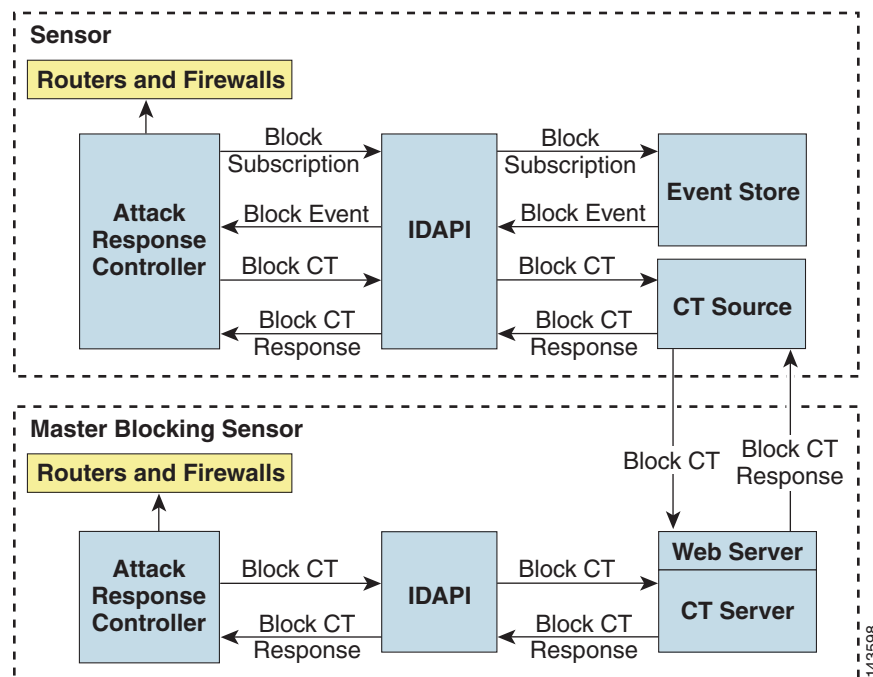
- [ACLs and VACLs, page A-16](#)
- [Maintaining State Across Restarts, page A-16](#)
- [Connection-Based and Unconditional Blocking, page A-17](#)
- [Blocking with Cisco Firewalls, page A-18](#)
- [Blocking with Catalyst Switches, page A-19](#)

Understanding the ARC

The main responsibility of the ARC is to block events. When it responds to a block, it either interacts with the devices it is managing directly to enable the block or it sends a block request through the Control Transaction Server to a master blocking sensor. The web server on the master blocking sensor receives the control transaction and passes it to the Control Transaction Server, which passes it to the ARC. The ARC on the master blocking sensor then interacts with the devices it is managing to enable the block.

Figure A-4 illustrates the ARC.

Figure A-4 ARC



Note

An ARC instance can control 0, 1, or many network devices. The ARC does not share control of any network device with other ARC applications, IPS management software, other network management software, or system administrators. Only one ARC instance is allowed to run on a given sensor.

The ARC initiates a block in response to one of the following:

- An alert event generated from a signature that is configured with a block action
- A block configured manually through the CLI, IDM, IME, or ASDM
- A block configured permanently against a host or network address

When you configure the ARC to block a device, it initiates either a Telnet or SSH connection with the device. The ARC maintains the connection with each device. After the block is initiated, the ARC pushes a new set of configurations or ACLs (one for each interface direction) to each controlled device. When a block is completed, all configurations or ACLs are updated to remove the block.

ARC Features

The ARC has the following features:

- Communication through Telnet and SSH 1.5 with 3DES (the default) or DES encryption

Only the protocol specified in the ARC configuration for that device is attempted. If the connection fails for any reason, the ARC attempts to reestablish it.

- Preexisting ACLs on routers and VACLs on switches

If a preexisting ACL exists on a router interface or direction that is controlled by the ARC, you can specify that this ACL be merged into the ARC-generated configuration, either before any blocks by specifying a preblock ACL or after any blocks by specifying a postblock ACL. The Catalyst 6000 VACL device types can have a preblock and postblock VACL specified for each interface that the ARC controls. The firewall device types use a different API to perform blocks and the ARC does not have any effect on preexisting ACLs on the firewalls.



Note Catalyst 5000 RSM and Catalyst 6000 MSFC2 network devices are supported in the same way as Cisco routers.

- Forwarding blocks to a list of remote sensors

The ARC can forward blocks to a list of remote sensors, so that multiple sensors can in effect collectively control a single network device. Such remote sensors are referred to as master blocking sensors.

- Specifying blocking interfaces on a network device

You can specify the interface and direction where blocking is performed in the ARC configuration for routers. You can specify the interface where blocking is performed in the VACL configuration. The ARC can simultaneously control up to 250 interfaces.



Note Cisco firewalls do not block based on interface or direction, so this configuration is never specified for them.

- Blocking hosts or networks for a specified time

The ARC can block a host or network for a specified number of minutes or indefinitely. The ARC determines when a block has expired and unblocks the host or network at that time.

- Logging important events

The ARC writes a confirmation event when block or unblock actions are completed successfully or if any errors occur. The ARC also logs important events such as loss and recovery of a network device communication session, configuration errors, and errors reported by the network device.

- Maintaining the blocking state across ARC restarts

The ARC reapplies blocks that have not expired when a shutdown or restart occurs. The ARC removes blocks that have expired while it was shut down.



Note The ARC can only maintain the blocking state successfully if no one changes the system time while the application is shut down.

- Maintaining blocking state across network device restarts

The ARC reapplies blocks and removes expired blocks as needed whenever a network device is shut down and restarted. The ARC is not affected by simultaneous or overlapping shutdowns and restarts of the ARC.

- Authentication and authorization

The ARC can establish a communications session with a network device that uses AAA authentication and authorization including the use of remote TACACS+ servers.

- Two types of blocking

The ARC supports host blocks and network blocks. Host blocks are connection based or unconditional. Network blocks are always unconditional.

- NAT addressing

The ARC can control network devices that use a NAT address for the sensor. If you specify a NAT address when you configure a network device, that address is used instead of the local IP address when the sensor address is filtered from blocks on that device.

- Single point of control

The ARC does not share control of network devices with administrators or other software. If you must update a configuration, shut down ARC until the change is complete. You can enable or disable the ARC through the CLI or any Cisco IPS manager. When the ARC is reenabled, it completely reinitializes itself, including rereading the current configuration for each controlled network device.



Note We recommend that you disable the ARC from blocking when you are configuring any network device, including firewalls.

- Maintains up to 250 active blocks at any given time

The ARC can maintain up to 250 active blocks at a time. Although the ARC can support up to 65535 blocks, we recommend that you allow no more than 250 at a time.



Note The number of blocks is not the same as the number of interface and directions.

Supported Blocking Devices

The ARC can control the following devices:

- Cisco routers running Cisco IOS 11.2 or later



Note To perform rate limiting, the routers must be running Cisco IOS 12.3 or later.

- Catalyst 5000 series switches with Supervisor Engine software 5.3(1) or later running on the supervisor engine, and IOS 11.2(9)P or later running on the RSM.



Note You must have the RSM because blocking is performed on the RSM.

- Catalyst 6000 series switches with PFC installed running Catalyst software 5.3 or later
- Catalyst 6000 MSFC2 with Catalyst software 5.4(3) or later and Cisco IOS 12.1(2)E or later on the MSFC2
- Cisco ASA 5500 series models: ASA 5510, ASA 5520, and ASA 5540
- FWSM



Note The FWSM cannot block in multi-mode admin context.

ACLs and VACLs

If you want to filter packets on an interface or direction that the ARC controls, you can configure the ARC to apply an ACL before any blocks (preblock ACL) and to apply an ACL after any blocks (postblock ACL). These ACLs are configured on the network device as inactive ACLs. You can define preblock and postblock ACLs for each interface and direction. The ARC retrieves and caches the lists and merges them with the blocking ACEs whenever it updates the active ACL on the network device. In most cases, you will want to specify a preexisting ACL as the postblock ACL so that it does not prevent any blocks from taking effect. ACLs work by matching a packet to the first ACE found. If this first ACE permits the packet, a subsequent deny statement will not be found.

You can specify different preblock and postblock ACLs for each interface and direction, or you can reuse the same ACLs for multiple interfaces and directions. If you do not want to maintain a preblock list, you can use the never block option and always block hosts and networks by using existing configuration statements. A forever block is a normal block with a timeout value of -1.

The ARC only modifies ACLs that it owns. It does not modify ACLs that you have defined. The ACLs maintained by ARC have a specific format that should not be used for user-defined ACLs. The naming convention is **IPS_<interface_name>_[in | out]_[0 | 1]**. <interface_name> corresponds to the name of the blocking interface as given in the ARC configuration.

For Catalyst switches, it is a blocking interface VLAN number. Do not use these names for preblock and postblock ACLs. For Catalyst 6000 VACLs, you can specify a preblock and postblock VACL and only the interface is specified (direction is not used in VLANs). For firewalls, you cannot use preblock or postblock ACLs because the firewall uses a different API for blocking. Instead you must create ACLs directly on the firewalls.

Maintaining State Across Restarts

When the sensor shuts down, the ARC writes all blocks and rate limits (with starting timestamps) to a local file (nac.shun.txt) that is maintained by the ARC. When the ARC starts, this file is used to determine if any block updates should occur at the controlled network devices. Any unexpired blocks found in the file are applied to the network devices at startup. When the ARC shuts down, no special actions on the ACLs are taken even if outstanding blocks are in effect. The nac.shun.txt file is accurate only if the system time is not changed while the ARC is not running.



Caution

Do not make manual changes to the nac.shun.txt file.

The following scenarios demonstrate how the ARC maintains state across restarts.

Scenario 1

There are two blocks in effect when the ARC stops and one of them expires before the ARC restarts. When the ARC restarts, it first reads the `nac.shun.txt` file. It then reads the preblock and postblock ACLs or VACLs. The active ACL or VACL is built in the following order:

1. The **allow** *sensor_ip_address* command (unless the **allow sensor shun** command has been configured)
2. Preblock ACL
3. The **always block** command entries from the configuration
4. Unexpired blocks from `nac.shun.txt`
5. Postblock ACL

When a host is specified as never block in the ARC configuration, it does not get translated into permit statements in the ACL. Instead, it is cached by the ARC and used to filter incoming `addShunEvent` events and `addShunEntry` control transactions.

Scenario 2

There are no preblock or postblock ACLs specified, but there is an existing active ACL. The new ACL is built in the following order:

1. The **allow** *sensor_ip_address* command (unless the **allow sensor shun** command has been configured)
2. The **always block** command entries from the configuration
3. Unexpired blocks from `nac.shun.txt`
4. The **permit IP any any** command

Connection-Based and Unconditional Blocking

The ARC supports two types of blocking for hosts and one type of blocking for networks. Host blocks are connection-based or unconditional. Network blocks are always unconditional.

When a host block is received, the ARC checks for the `connectionShun` attribute on the host block. If `connectionShun` is set to true, the ARC performs connection blocking. Any host block can contain optional parameters, such as destination IP address, source port, destination port, and protocol. For a connection block to take place, at least the source and destination IP address must be present. If the source port is present on a connection block, it is ignored and not included in the block.

Under the following conditions, the ARC forces the block to be unconditional, converting the block from connection type if necessary:

- A block of any type is active for a specified source IP address
- A new block of any type is received for that source IP address
- The new block differs in any of its optional parameters (except the source port) from the old block

When a block is updated (for example, when a new block arrives while an existing block for that source IP address or network is already in effect), the remaining minutes of the existing block are determined. If the time for the new block is less than or equal to the remaining minutes, no action is taken. Otherwise, the new block timeout replaces the existing block timeout.

**Caution**

Cisco firewalls do not support connection blocking of hosts. When a connection block is applied, the firewall treats it like an unconditional block. Cisco firewalls also do not support network blocking. ARC never tries to apply a network block to a Cisco firewall.

Blocking with Cisco Firewalls

The ARC performs blocks on firewalls using the **shun** command. The **shun** command has the following formats:

- To block an IP address:
`shun srcip [destination_ip_address source_port destination_port [port]]`
- To unblock an IP address:
`no shun ip`
- To clear all blocks:
`clear shun`
- To show active blocks or to show the global address that was actually blocked:
`show shun [ip_address]`

The ARC uses the response to the **show shun** command to determine whether the block was performed. The **shun** command does not replace existing ACLs, conduits, or outbound commands, so there is no need to cache the existing firewall configuration, nor to merge blocks into the firewall configuration.

**Caution**

Do not perform manual blocks or modify the existing firewall configuration while ARC is running.

If the **block** command specifies only the source IP address, existing active TCP connections are not broken, but all incoming packets from the blocked host are dropped.

When the ARC first starts up, the active blocks in the firewall are compared to an internal blocking list. Any blocks that do not have a corresponding internal list entry are removed.

The ARC supports authentication on a firewall using local usernames or a TACACS+ server. If you configure the firewall to authenticate using AAA but without the TACACS+ server, the ARC uses the reserved username *pix* for communications with the firewall.

If the firewall uses a TACACS+ server for authentication, you use a TACACS+ username. In some firewall configurations that use AAA logins, you are presented with three password prompts: the initial firewall password, the AAA password, and the enable password. The ARC requires that the initial firewall password and the AAA password be the same.

When you configure a firewall to use NAT or PAT and the sensor is checking packets on the firewall outside network, if you detect a host attack that originates on the firewall inside network, the sensor tries to block the translated address provided by the firewall. If you are using dynamic NAT addressing, the block can be ineffective or cause innocent hosts to be blocked. If you are using PAT addressing, the firewall could block the entire inside network. To avoid these situations, position your sensor on the inside interface or do not configure the sensor to block.

Blocking with Catalyst Switches

Catalyst switches with a PFC filter packets using VACLs. VACLs filter all packets between VLANs and within a VLAN. MSFC router ACLs are supported when WAN cards are installed and you want the sensor to control the interfaces through the MSFC2.

**Note**

An MSFC2 card is not a required part of a Catalyst switch configuration for blocking with VACLs.

**Caution**

When you configure the ARC for the Catalyst switch, do not specify a direction with the controlled interface. The interface name is a VLAN number. Preblock and postblock lists should be VACLs.

The following commands apply to the Catalyst VACLs:

- To view an existing VACL:

```
show security acl info acl_name
```
- To block an address (*address_spec* is the same as used by router ACLs):

```
set security acl ip acl_name deny address_spec
```
- To activate VACLs after building the lists:

```
commit security acl all
```
- To clear a single VACL:

```
clear security acl map acl_name
```
- To clear all VACLs:

```
clear security acl map all
```
- To map a VACL to a VLAN:

```
set sec acl acl_name vlans
```

Logger

The sensor logs all events (alert, error, status, and debug messages) in a persistent, circular buffer. The sensor also generates IP logs. The messages and IP logs are accessible through the CLI, IDM, and ASDM.

The IPS applications use the Logger to log messages. The Logger sends log messages at any of five levels of severity: debug, timing, warning, error, and fatal. The Logger writes the log messages to `/usr/cids/idsRoot/log/main.log`, which is a circular text file. New messages overwrite older messages when the file reaches its maximum size; therefore the last message written may not appear at the end of the `main.log`. Search for the string “= END OF FILE =” to locate the last line written to the `main.log`.

The `main.log` is included in the **show tech-support** command output. If the message is logged at warning level or above (error or fatal), the Logger converts the message to an `evError` event (with the corresponding error severity) and inserts it in the Event Store.

The Logger receives all syslog messages, except cron messages, that are at the level of informational and above (`*.info;cron.none`), and inserts them in to the Event Store as `evErrors` with the error severity set to Warning. The Logger and application logging are controlled through the service logger commands.

The Logger can control what log messages are generated by each application by controlling the logging severity for different logging zones. You would only access the individual-zone-control of the logger service at the request and supervision of a TAC engineer or developer. For troubleshooting purposes, TAC might request that you turn on debug logging.

AuthenticationApp

This section describes the AuthenticationApp, and contains the following topics:

- [Understanding the AuthenticationApp, page A-20](#)
- [Authenticating Users, page A-20](#)
- [Configuring Authentication on the Sensor, page A-20](#)
- [Managing TLS and SSH Trust Relationships, page A-21](#)

Understanding the AuthenticationApp

The AuthenticationApp has the following responsibilities:

- To authenticate the identity of a user
- To administer the accounts, privileges, keys, and certificates of the user
- To configure which authentication methods are used by the AuthenticationApp and other access services on the sensor

Authenticating Users

You must configure authentication on the sensor to establish appropriate security for user access. When you install a sensor, an initial cisco account with an expired password is created. A user with administrative access to the sensor accesses the sensor through the CLI or an IPS manager, such as the IDM or the ASDM, by logging in to the sensor using the default administrative account (**cisco**). In the CLI, the administrator is prompted to change the password. IPS managers initiate a `setEnableAuthenticationTokenStatus` control transaction to change the password of an account.

Through the CLI or an IPS manager, the administrator configures which authentication method is used, such as username and password or an SSH authorized key. The application servicing the administrator initiates a `setAuthenticationConfig` control transaction to establish the authentication configuration.

The authentication configuration includes a login attempt limit value that is used to specify how account locking is handled. Account locking is invoked when the number of consecutive failed login attempts for a given account exceeds the login attempt limit value. After an account is locked, all further attempts to log in to that account are rejected. The account is unlocked by resetting the authentication token of the account using the `setEnableAuthenticationTokenStatus` control transaction. The account locking feature is disabled when the login attempt limit value is set to zero.

The administrator can add additional user accounts either through the CLI or an IPS manager.

Configuring Authentication on the Sensor

When a user tries to access the sensor through a service such as web server or the CLI, the identity of the user must be authenticated and the privileges of the user must be established. The service that is providing access to the user initiates an `execAuthenticateUser` control transaction request to the

AuthenticationApp to authenticate the identity of the user. The control transaction request typically includes the username and a password, or the identity of the user can be authenticated using an SSH authorized key.

The AuthenticationApp responds to the `execAuthenticateUser` control transaction request by attempting to authenticate the identity of the user. The AuthenticationApp returns a control transaction response that contains the authentication status and privileges of the user. If the identity of the user cannot be authenticated, the AuthenticationApp returns an unauthenticated status and anonymous user privileges in the control transaction response. The control transaction response also indicates if the account password has expired. User interface applications that authenticate users by initiating an `execAuthenticateUser` control transaction prompt the user to change the password.

The AuthenticationApp uses the underlying operating system to confirm the identity of a user. All the IPS applications send control transactions to the AuthenticationApp, which then uses the operating system to form its responses.

Remote shell services, Telnet and SSH, are not IPS applications. They call the operating system directly. If the user is authenticated, it launches the IPS CLI. In this case, the CLI sends a special form of the `execAuthenticateUser` control transaction to determine the privilege level of the logged-in user. The CLI then tailors the commands it makes available based on this privilege level.

Managing TLS and SSH Trust Relationships

Encrypted communications over IP networks provide data privacy by making it impossible for a passive attacker to discover from the packets exchanged alone the secret key needed to decrypt the data in the packets.

However, an equally dangerous attack vector is for an imposter to pretend to be the server end of the connection. All encryption protocols provide a means for clients to defend themselves from these attacks. IPS supports two encryption protocols, SSH and TLS, and the AuthenticationApp helps manage trust when the sensor plays either the client or server role in encrypted communications.

The IPS web server and SSH server are server endpoints of encrypted communications. They protect their identities with a private key and offer a public key to clients that connect to them. For TLS this public key is included inside an X.509 certificate, which includes other information. Remote systems that connect to the sensor should verify that the public key received during connection establishment is the key they expect.

Clients must maintain a list of trusted public keys to protect themselves from man-in-the-middle attacks. The exact procedure by which this trust is established varies depending on the protocol and client software. In general, the client displays a fingerprint of 16 or 20 bytes. The human operator who is configuring the client to establish trust should use an out-of-band method to learn the key fingerprints of the server before attempting to establish trust. If the fingerprints match, the trust relationship is established and henceforth the client can automatically connect with that server and be confident that the remote server is not an imposter.

You can use the **`show ssh server-key`** and **`show tls fingerprint`** to display the key fingerprints of the sensor. By recording the output of these commands when directly connected to the sensor console, you can reliably use this information to confirm the identity of the sensor over the network later when establishing trust relationships.

For example, when you initially connect to a sensor through the Microsoft Internet Explorer web browser, a security warning dialog box indicates that the certificate is not trusted. Using the user interface of Internet Explorer, you can inspect the certificate thumbprint, a value that should exactly match the SHA1 fingerprint displayed by the **`show tls fingerprint`** command. After verifying this, add this certificate to the list of trusted CAs of the browser to establish permanent trust.

Each TLS client has different procedures for establishing this trust. The sensor itself includes a TLS client that is used to send control transactions to other sensors and download upgrades and configuration files from other TLS web servers. Use the **tls trusted-host** command to establish trust of the TLS servers with which the sensor communicates.

Similarly, the sensor includes an SSH client that is used to communicate with managed network devices, download upgrades, and copy configurations and support files to remote hosts. Use the **ssh host-key** command to establish trust relationships with the SSH servers the sensor will contact.

You can manage the list of TLS trusted certificates and SSH known hosts through the commands **service trusted-certificates** and **service ssh-known-hosts**.

X.509 certificates include additional information that can increase the security of the trust relationship; however, these can lead to confusion. For example, an X.509 certificate includes a validity period during which the certificate can be trusted. Typically this period is a number of years starting at the moment the certificate is created. To ensure that an X.509 certificate is valid at the moment it is being used requires that the client system maintain an accurate clock.

X.509 certificates are also tied to a particular network address. Sensors fill this field with the IP address of the command and control interface of the sensor. Consequently, if you change the command and control IP address of the sensor, the X.509 certificate of the server is regenerated. You must reconfigure all clients on the network that trusted the old certificate to locate the sensor at its new IP address and trust the new certificate.

By using the SSH known hosts and TLS trusted certificates services in the AuthenticationApp, you can operate sensors at a high level of security.

Web Server

The web server provides SDEE support, which enables the sensor to report security events, receive IDIOM transactions, and serve IP logs. The web server supports HTTP 1.0 and 1.1. Communications with the web server often include sensitive information, such as passwords, that would severely compromise the security of the system if an attacker were able to eavesdrop. For this reason, sensors ship with TLS enabled. The TLS protocol is an encryption protocol that is compatible with SSL.



Note

We deprecated the RDEP event sever service in IPS 6.1, and deleted it from the IPS 7.0(1) system architecture. The web server now uses the SDEE event server.

SensorApp

This section describes the SensorApp, and contains the following topics:

- [Understanding the SensorApp, page A-23](#)
- [Inline, Normalization, and Event Risk Rating Features, page A-24](#)
- [SensorApp New Features, page A-25](#)
- [Packet Flow, page A-25](#)
- [Signature Event Action Processor, page A-26](#)

Understanding the SensorApp

The SensorApp performs packet capture and analysis. Policy violations are detected through signatures in the SensorApp and the information about the violations is forwarded to the Event Store in the form of an alert. Packets flow through a pipeline of processors fed by a producer designed to collect packets from the network interfaces on the sensor. Event actions can be associated with an event risk rating threshold that must be surpassed for the actions to take place. Some of the processors call inspectors to perform signature analysis. All inspectors can call the alarm channel to produce alerts as needed.

The SensorApp supports the following processors:

- Time Processor—This processor processes events stored in a time-slice calendar. Its primary task is to make stale database entries expire and to calculate time-dependent statistics.
- Deny Filters Processor—This processor handles the deny attacker functions. It maintains a list of denied source IP addresses. Each entry in the list expires based on the global deny timer, which you can configure in the virtual sensor configuration.
- Signature Event Action Processor—This processor processes event actions. Event actions can be associated with an event risk rating threshold that must be surpassed for the actions to take place. It supports the following event actions:
 - Reset TCP flow
 - IP log
 - Deny packets
 - Deny flow
 - Deny attacker
 - Alert
 - Block host
 - Block connection
 - Generate SNMP trap
 - Capture trigger packet
- Statistics Processor—This processor keeps track of system statistics such as packet counts and packet arrival rates.
- Layer 2 Processor—This processor processes layer 2-related events. It also identifies malformed packets and removes them from the processing path. You can configure actionable events for detecting malformed packets such as alert, capture packet, and deny packet. The layer 2 processor updates statistics about packets that have been denied because of the policy you have configured.
- Database Processor—This processor maintains the signature state and flow databases.
- Fragment Reassembly Processor—This processor reassembles fragmented IP datagrams. It is also responsible for normalization of IP fragments when the sensor is in inline mode.
- Stream Reassembly Processor—This processor reorders TCP streams to ensure the arrival order of the packets at the various stream-based inspectors. It is also responsible for normalization of the TCP stream. The normalizer engine lets you enable or disable alert and deny actions.

The TCP Stream Reassembly Processor normalizer has a hold-down timer, which lets the stream state rebuild after a reconfiguration event. You cannot configure the timer. During the hold-down interval, the system synchronizes stream state on the first packet in a stream that passes through the system. When the hold down has expired, sensorApp enforces your configured policy. If this policy calls for a denial of streams that have not been opened with a 3-way handshake, established streams

that were quiescent during the hold-down period will not be forwarded and will be allowed to timeout. Those streams that were synchronized during the hold-down period are allowed to continue.

- **Signature Analysis Processor**—This processor dispatches packets to the inspectors that are not stream-based and that are configured for interest in the packet in process.
- **Slave Dispatch Processor**—A process found only on dual CPU systems.

The SensorApp also supports the following units:

- **Analysis Engine**—The Analysis Engine handles sensor configuration. It maps the interfaces and also the signature and alarm channel policy to the configured interfaces.
- **Alarm Channel**—The Alarm Channel processes all signature events generated by the inspectors. Its primary function is to generate alerts for each event it is passed.

Inline, Normalization, and Event Risk Rating Features

The SensorApp contains the following inline, normalization, and event risk rating features:

- **Processing packets inline**

When the sensor is processing packets in the data path, all packets are forwarded without any modifications unless explicitly denied by policy configuration. Because of TCP normalization it is possible that some packets will be delayed to ensure proper coverage. When policy violations are encountered, the SensorApp allows for the configuration of actions. Additional actions are available in inline mode, such as deny packet, deny flow, and deny attacker.

All packets that are unknown or of no interest to the IPS are forwarded to the paired interface with no analysis. All bridging and routing protocols are forwarded with no participation other than a possible deny due to policy violations. There is no IP stack associated with any interface used for inline (or promiscuous) data processing. The current support for 802.1q packets in promiscuous mode is extended to inline mode.

- **IP normalization**

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

- **TCP normalization**

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

- Event risk rating

Event risk rating helps reduce false positives from the system and gives you more control over what causes an alarm. The event risk rating incorporates the following additional information beyond the detection of a potentially malicious action:

 - Severity of the attack if it were to succeed
 - Fidelity of the signature
 - Relevance of the potential attack with respect to the target host
 - Overall value of the target host

SensorApp New Features

The SensorApp contains the following new features:

- Policy table—Provides a list of risk category settings (high, medium, and low).
- Evasion protection—Lets an inline interface mode sensor switch from strict mode to asymmetric mode for the Normalizer.
- Sensor health meter—Provides sensor-wide health statistics.
- Top services—Provides the top ten instances of the TCP, UDP, ICMP, and IP protocols.
- Security meter—Profiles alerts into threat categories and reports this information in red, yellow, and green buckets. You can configure the transition points for these buckets.
- Clear Flow state—Lets you clear the database, which causes the sensor to start fresh just as in a restart.
- Restart status—Reports periodically the current start and restart stages of the sensor.

Packet Flow

Packets are received by the NIC and placed in the kernel user-mapped memory space by the IPS-shared driver. The packet is prepended by the IPS header. Each packet also has a field that indicates whether to pass or deny the packet when it reaches Signature Event Action Processor.

The producer pulls packets from the shared-kernel user-mapped packet buffer and calls the process function that implements the processor appropriate to the sensor model. The following orders occur:

- Single processor execution
Time Processor --> Layer 2 Processor --> Deny Filters Processor --> Fragment Reassembly Processor --> Statistics Processor --> Database Processor --> Signature Analysis Processor --> Stream Reassembly Processor --> Signature Event Action Processor
- Dual processor execution
Execution Thread 1 Time Processor --> Layer 2 Processor --> Deny Filters Processor --> Fragment Reassembly Processor --> Statistics Processor --> Database Processor --> Signature Analysis Processor --> Slave Dispatch Processor --> | Execution Thread 2 Database Processor --> Stream Reassembly Processor --> Signature Event Action Processor

Signature Event Action Processor

The Signature Event Action Processor coordinates the data flow from the signature event in the Alarm Channel to processing through the Signature Event Action Override, the Signature Event Action Filter, and the Signature Event Action Handler. It consists of the following components:

- Alarm Channel—The unit that represents the area to communicate signature events from the SensorApp inspection path to signature event handling.
- Signature Event Action Override—Adds actions based on the risk rating value. Signature Event Action Override applies to all signatures that fall in the range of the configured risk rating threshold. Each Signature Event Action Override is independent and has a separate configuration value for each action type.
- Signature Event Action Filter—Subtracts actions based on the signature ID, addresses, and risk rating of the signature event. The input to the Signature Event Action Filter is the signature event with actions possibly added by the Signature Event Action Override.



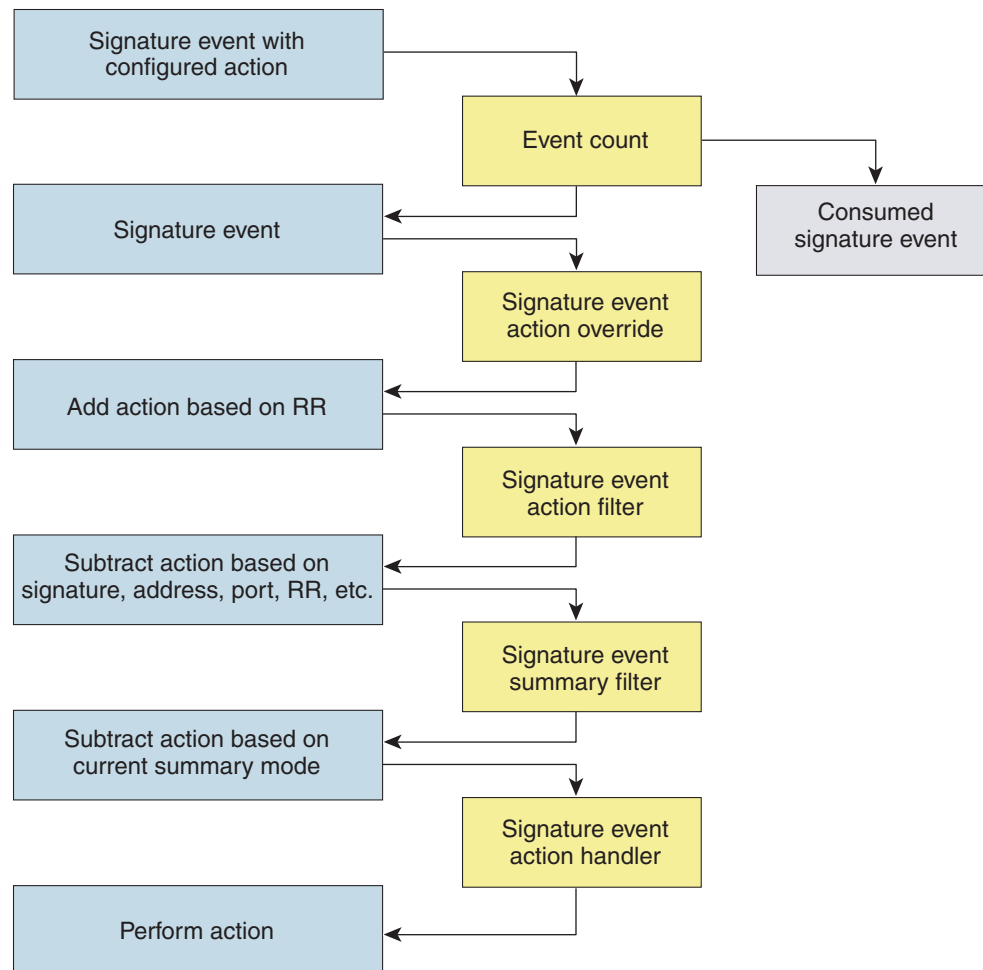
Note The Signature Event Action Filter can only subtract actions, it cannot add new actions.

The following parameters apply to the Signature Event Action Filter:

- Signature ID
 - Subsignature ID
 - Attacker address
 - Attacker port
 - Victim address
 - Victim port
 - Risk rating threshold range
 - Actions to subtract
 - Sequence identifier (optional)
 - Stop-or-continue bit
 - Enable action filter line bit
 - Victim OS relevance or OS relevance
- Signature Event Action Handler—Performs the requested actions. The output from the Signature Event Action Handler is the actions being performed and possibly an evIdsAlert written to the Event Store.

[Figure A-5 on page A-27](#) illustrates the logical flow of the signature event through the Signature Event Action Processor and the operations performed on the action for this event. It starts with the signature event with configured action received in the Alarm Channel and flows top to bottom as the signature event passes through the functional components of the Signature Event Action Processor.

Figure A-5 Signature Event Through Signature Event Action Processor



132188

CollaborationApp

This section describes the CollaborationApp, and contains the following sections:

- [Understanding the CollaborationApp, page A-27](#)
- [Update Components, page A-28](#)
- [Error Events, page A-29](#)

Understanding the CollaborationApp

The CollaborationApp is a peer of the MainApp and the SensorApp. It interfaces with them using various interprocess communication technologies, such as IDAPI control transactions, semaphores, shared memory, and file exchange.

Reputation updates are exchanged between the Global Correlation server and the CollaborationApp. The CollaborationApp communicates with the sensors using four update components:

- Set of rules score weight values
- Set of IP addresses and address ranges, which together with the rules and alerts provide the information needed to calculate reputation scores
- List of IP addresses and address ranges for which traffic should always be denied
- Network participation configuration, which allows the server to control the rate at which sensors send telemetry data to the server

The sensor sends collaboration information to the Network Participation server. The sensor queries the Global Correlation server for a list of what collaboration updates are available and from which Global Correlation server to download the update files.

**Note**

The SensorApp starts before the CollaborationApp, but they initialize asynchronously. Therefore, it is possible that the Reputation Update server may download and attempt to apply one or more global correlation updates before the SensorApp is ready to accept the update. The update server may download and partially process the update, but it must wait until the SensorApp is ready before it can commit the update.

For More Information

For detailed information on global correlation and how to configure it, see [Chapter 10, “Configuring Global Correlation.”](#)

Update Components

The Global Correlation Update client exchanges manifests with the Global Correlation Update server. It parses the server manifest to determine what new updates are available for download and where they reside, and then builds a list of updates to be installed. If all updates are applied successfully, then the Global Correlation Update client commits the applied updates for each component, notifies SensorApp that new updates are available, and updates the client manifest to reflect the latest committed updates for each component.

The client manifest contains the UDI of the sensor, which includes the serial number of the sensor, and an encrypted shared secret that the server uses to verify the sensor is an authentic Cisco IPS sensor. The server manifest contains a list of update files available for each component. For each update file in the list, the server manifest contains data, such as the update version, type, order, location, file transfer protocol, and so forth.

There are two types of updates files: a full update file that replaces any existing data in the database of the component, and an incremental update that modifies the existing reputation data by adding, deleting, or replacing information. When all update files have been applied for all components, the temporary databases are committed by replacing the working databases.

Authentication and authorization are achieved through the secret encryption mechanism and decryption key management. The Global Correlation Update server authenticates the sensor using the shared secret encryption mechanism contained in the client manifest. The Global Correlation Update client authorizes sensors using decryption key management. Sensors that have been authenticated by the Global Correlation Update server are sent valid keys in the server manifest so that they can decrypt the update files.

**Caution**

You receive a warning message if you have enabled global correlation, but you have not configured a DNS or HTTP proxy server. This warning is a reminder to either disable global correlation or add a DNS or HTTP proxy server.

For More Information

For the procedure for adding a DNS or proxy server to support global correlation, see [Changing Network Settings, page 4-2](#).

Error Events

Whenever a global correlation update fails, an evError event is generated. The error message is included in sensor statistics. The following conditions result in a status message with the severity of Error:

- The sensor is unlicensed
- No DNS or HTTP proxy server is configured
- The manifest exchange failed
- An update file download failed
- Applying or committing the update failed

An evError event is generated with the severity level of Warning if you edit and save either the host or global correlation configurations so that global correlation is enabled, but no DNS or HTTP proxy servers are configured.

For More Information

For the procedure for displaying sensor statistics, see [Displaying Statistics, page 17-30](#).

SwitchApp

The 4500 series sensors have a built in switch that provides the external monitoring interfaces of the sensor. The SwitchApp is part of the IPS 4500 series design that enables the InterfaceApp and sensor initialization scripts to communicate with and control the switch. Any application that needs to get or set information on the switch must communicate with the SwitchApp. Additionally the SwitchApp implements the following:

- Detects bypass—When the SensorApp is not monitoring, the SwitchApp places the switch in bypass mode and then back to inspection mode once the SensorApp is up and running normally.
- Collects port statistics—The SwitchApp monitors the switch and collects statistics on the external interfaces of the switch for reporting by InterfaceApp.
- Handles the external interface configuration—When you update the interface configuration, the configuration is sent to the InterfaceApp, which updates the interface configuration for SwitchApp, which then forwards that configuration on to the switch.

CLI

The CLI provides the sensor user interface for all direct node access such as Telnet, SSH, and serial interface. You configure the sensor applications with the CLI. Direct access to the underlying OS is allowed through the service role. This section describes the IPS CLI, and contains the following topics:

- [User Roles, page A-30](#)
- [Service Account, page A-31](#)

User Roles



Caution

You should carefully consider whether you want to create a service account. The service account provides shell access to the system, which makes the system vulnerable. However, you can use the service account to create a password if the administrator password is lost. Analyze your situation to decide if you want a service account existing on the system.

There are four user roles:

- **Viewer**—Can view configuration and events, but cannot modify any configuration data except their user passwords.
- **Operator**—Can view everything and can modify the following options:
 - Signature tuning (priority, disable or enable)
 - Virtual sensor definition
 - Managed routers
 - Their user passwords
- **Administrator**—Can view everything and can modify all options that operators can modify in addition to the following:
 - Sensor addressing configuration
 - List of hosts allowed to connect as configuration or viewing agents
 - Assignment of physical sensing interfaces
 - Enable or disable control of physical interfaces
 - Add and delete users and passwords
 - Generate new SSH host keys and server certificates
- **Service**—Only one user with service privileges can exist on a sensor. The service user cannot log in to the IDM or the IME. The service user logs in to a bash shell rather than the CLI.

The service role is a special role that allows you to bypass the CLI if needed. Only one service account is allowed. You should only create an account with the service role for troubleshooting purposes. Only a user with administrator privileges can edit the service account.

When you log in to the service account, you receive the following warning:

```
***** WARNING *****
UNAUTHORIZED ACCESS TO THIS NETWORK DEVICE IS PROHIBITED.
This account is intended to be used for support and troubleshooting purposes only.
Unauthorized modifications are not supported and will require this device to be
re-imaged to guarantee proper operation.
*****
```

**Note**

For IPS 5.0 and later, you can no longer remove the **cisco** account. You can disable it using the **no password cisco** command, but you cannot remove it. To use the **no password cisco** command, there must be another administrator account on the sensor. Removing the **cisco** account through the service account is not supported. If you remove the **cisco** account through the service account, the sensor most likely will not boot up, so to recover the sensor you must reinstall the sensor system image.

Service Account

The service account is a support and troubleshooting tool that enables TAC to log in to a native operating system shell rather than the CLI shell. It does not exist on the sensor by default. You must create it so that it is available for TAC to use for troubleshooting your sensor.

Only one service account is allowed per sensor and only one account is allowed a service role. When the password of the service account is set or reset, the password of the root account is set to the same password. This allows the service account user to su to root using the same password. When the service account is removed, the password of the root account is locked.

The service account is not intended to be used for configuration purposes. Only modifications made to the sensor through the service account under the direction of TAC are supported. Cisco Systems does not support the addition and/or running of an additional service to the operating system through the service account, because it affects proper performance and proper functioning of the other IPS services. TAC does not support a sensor on which additional services have been added.

You can track logins to the service account by checking the log file `/var/log/.tac`, which is updated with a record of service account logins.

**Note**

The Cisco IPS incorporates several troubleshooting features that are available through the CLI, IDM, or IME. The service account is not necessary for most troubleshooting situations. You may need to create the service account at the direction of TAC to troubleshoot a very unique problem. The service account lets you bypass the protections built into the CLI and allows root privilege access to the sensor, which is otherwise disabled. We recommend that you do not create a service account unless it is needed for a specific reason. You should remove the service account when it is no longer needed.

Communications

This section describes the communications protocols used by the Cisco IPS. It contains the following topics:

- [IDAPI, page A-32](#)
- [IDIOM, page A-32](#)
- [IDCONF, page A-33](#)
- [SDEE, page A-33](#)
- [CIDEE, page A-34](#)

IDAPI

IPS applications use an interprocess communication API called the IDAPI to handle internal communications. The IDAPI reads and writes event data and provides a mechanism for control transactions. The IDAPI is the interface through which all the applications communicate.

The SensorApp captures and analyzes the network traffic on its interfaces. When a signature is matched, the SensorApp generates an alert, which is stored in the Event Store. If the signature is configured to perform the blocking response action, the SensorApp generates a block event, which is also stored in the Event Store.

Figure A-6 illustrates the IDAPI interface.

Figure A-6 IDAPI



Each application registers to the IDAPI to send and receive events and control transactions. The IDAPI provides the following services:

- Control transactions
 - Initiates the control transaction.
 - Waits for the inbound control transaction.
 - Responds to the control transaction.
- IPS events
 - Subscribes to remote IPS events, which are stored in the Event Store when received.
 - Reads IPS events from the Event Store.
 - Writes IPS events to the Event Store.

The IDAPI provides the necessary synchronization mechanisms to guarantee atomic data accesses.

IDIOM

IDIOM is a data format standard that defines the event messages that are reported by the IPS as well as the operational messages that are used to configure and control intrusion detection systems. These messages consist of XML documents that conform to the IDIOM XML schema.

IDIOM supports two types of interactions: event and control transaction. Event interactions are used to exchange IPS events such as alerts. IDIOM uses two types of messages for event interactions: event and error messages. Control transactions provide a means for one host to initiate an action in, change the state of, or read the state of another host. Control transactions utilize four types of IDIOM messages: request, response, configuration, and error messages. Events and control transactions that are communicated between application instances within a host are known as local events or local control transactions, or collectively, local IDIOM messages. Events and control transactions that are communicated between different hosts are known as remote events and remote control transactions, or collectively, remote IDIOM messages.



Note

IDIOM for the most part has been superseded by IDCONF, SDEE, and CIDEE.

IDCONF

The Cisco IPS manages its configuration using XML documents. IDCONF specifies the XML schema including the Cisco IPS control transactions. The IDCONF schema does not specify the contents of the configuration documents, but rather the framework and building blocks from which the configuration documents are developed. It provides mechanisms that let the IPS managers and CLI ignore features that are not configurable by certain platforms or functions through the use of the feature-supported attribute.

IDCONF messages are wrapped inside IDIOM request and response messages.

The following is an IDCONF example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<request xmlns="http://www.cisco.com/cids/idiom" schemaVersion="2.00">
  <editConfigDelta xmlns="http://www.cisco.com/cids/idconf">
    <component name="userAccount">
      <config typedefsVersion="2004-03-01" xmlns="http://www.cisco.com/cids/idconf">
        <struct>
          <map name="user-accounts" editOp="merge">
            <mapEntry>
              <key>
                <var name="name">cisco</var>
              </key>
              <struct>
                <struct name="credentials">
                  <var name="role">administrator</var>
                </struct>
              </struct>
            </mapEntry>
          </map>
        </struct>
      </config>
    </component>
  </editDefaultConfig>
</request>
```

SDEE

The Cisco IPS produces various types of events including intrusion alerts and status events. The IPS communicates events to clients such as management applications using the proprietary IPS-industry leading protocol, SDEE, which is a product-independent standard for communicating security device events. SDEE adds extensibility features that are needed for communicating events generated by various types of security devices.

Systems that use SDEE to communicate events to clients are referred to as SDEE providers. SDEE specifies that events can be transported using the HTTP or HTTP over SSL and TLS protocols. When HTTP or HTTPS is used, SDEE providers act as HTTP servers, while SDEE clients are the initiators of HTTP requests.

The IPS includes the web server, which processes HTTP or HTTPS requests. The web server uses run-time loadable servlets to process the different types of HTTP requests. Each servlet handles HTTP requests that are directed to the URL associated with the servlet. The SDEE server is implemented as a web server servlet.

The SDEE server only processes authorized requests. A request is authorized if it originates from a web server to authenticate the identity of the client and determine the privilege level of the client.

CIDEE

CIDEE specifies the extensions to SDEE that are used by the Cisco IPS. The CIDEE standard specifies all possible extensions that are supported by the Cisco IPS. Specific systems may implement a subset of CIDEE extensions. However, any extension that is designated as being required **MUST** be supported by all systems. CIDEE specifies the Cisco IPS-specific security device events and the IPS extensions to the SDEE evIdsAlert element.

CIDEE supports the following events:

- evError—Error event
Generated by the CIDEE provider when the provider detects an error or warning condition. The evError event contains error code and textual description of the error.
- evStatus—Status message event
Generated by CIDEE providers to indicate that something of potential interest occurred on the host. Different types of status messages can be reported in the status event—one message per event. Each type of status message contains a set of data elements that are specific to the type of occurrence that the status message is describing. The information in many of the status messages are useful for audit purposes. Errors and warnings are not considered status information and are reported using evError rather than evStatus.
- evShunRqst—Block request event
Generated to indicate that a block action is to be initiated by the service that handles network blocking.

The following is a CIDEE extended event example:

```
<sd:events xmlns:cid="http://www.cisco.com/cids/2004/04/cidee"
xmlns:sd="http://example.org/2003/08/sdee">
  <sd:evIdsAlert eventId="1042648730045587005" vendor="Cisco" severity="medium">
    <sd:originator>
      <sd:hostId>Beta4Sensor1</sd:hostId>
      <cid:appName>sensorApp</cid:appName>
      <cid:appInstanceId>8971</cid:appInstanceId>
    </sd:originator>
    <sd:time offset="0" timeZone="UTC">1043238671706378000</sd:time>
    <sd:signature description="IOS Udp Bomb" id="4600" cid:version="S37">
      <cid:subsigId>0</cid:subsigId>
    </sd:signature> ...
  </sd:evIdsAlert>
</sd:events>
```

Cisco IPS File Structure

The Cisco IPS has the following directory structure:

- /usr/cids/idsRoot—Main installation directory.
- /usr/cids/idsRoot/shared—Stores files used during system recovery.
- /usr/cids/idsRoot/var—Stores files created dynamically while the sensor is running.
- /usr/cids/idsRoot/var/updates—Stores files and logs for update installations.
- /usr/cids/idsRoot/var/virtualSensor—Stores files used by SensorApp to analyze regular expressions.
- /usr/cids/idsRoot/var/eventStore—Contains the Event Store application.
- /usr/cids/idsRoot/var/core—Stores core files that are created during system crashes.
- /usr/cids/idsRoot/var/iplogs—Stores IP log file data.

- /usr/cids/idsRoot/bin—Contains the binary executables.
- /usr/cids/idsRoot/bin/authentication—Contains the authentication application.
- /usr/cids/idsRoot/bin/cidDump—Contains the script that gathers data for tech support.
- /usr/cids/idsRoot/bin/cidwebserver—Contains the web server application.
- /usr/cids/idsRoot/bin/cidcli—Contains the CLI application.
- /usr/cids/idsRoot/bin/nac—Contains the ARC application.
- /usr/cids/idsRoot/bin/logApp—Contains the logger application.
- /usr/cids/idsRoot/bin/mainApp—Contains the main application.
- /usr/cids/idsRoot/bin/sensorApp—Contains the sensor application.
- /usr/cids/idsRoot/bin/collaborationApp—Contains the collaboration application.
- /usr/cids/idsRoot/bin/switchApp—Contains the switch application.
- /usr/cids/idsRoot/etc—Stores sensor configuration files.
- /usr/cids/idsRoot/htdocs—Contains the IDM files for the web server.
- /usr/cids/idsRoot/lib—Contains the library files for the sensor applications.
- /usr/cids/idsRoot/log—Contains the log files for debugging.
- /usr/cids/idsRoot/tmp—Stores the temporary files created during run time of the sensor.

Summary of Cisco IPS Applications

Table A-2 gives a summary of the applications that make up the IPS.

Table A-2 Summary of Applications

Application	Description
AuthenticationApp	Authorizes and authenticates users based on IP address, password, and digital certificates.
Attack Response Controller	An ARC is run on every sensor. Each ARC subscribes to network access events from its local Event Store. The ARC configuration contains a list of sensors and the network access devices that its local ARC controls. If an ARC is configured to send network access events to a master blocking sensor, it initiates a network access control transaction to the remote ARC that controls the device. These network access action control transactions are also used by IPS managers to issue occasional network access actions.
CLI	Accepts command line input and modifies the local configuration using the IDAPI.
CollaborationApp	Shares information with other devices through a global correlation database to improve the combined efficacy of all the devices.
Control Transaction Server ¹	Accepts control transactions from a remote client, initiates a local control transaction, and returns the response to the remote client.
Control Transaction Source ²	Waits for control transactions directed to remote applications, forwards the control transactions to the remote node, and returns the response to the initiator.

Table A-2 Summary of Applications (continued)

Application	Description
IDM	The Java applet that provides an HTML IPS management interface.
IME	The Java applet that provides an interface for viewing and archiving events.
InterfaceApp	Handles bypass and physical settings and defines paired interfaces. Physical settings are speed, duplex, and administrative state.
Logger	Writes all the log messages of the application to the log file and the error messages of the application to the Event Store.
MainApp	Reads the configuration and starts applications, handles starting and stopping of applications and node reboots, handles software upgrades.
NotificationApp	Sends SNMP traps when triggered by alert, status, and error events. NotificationApp uses the public domain SNMP agent. SNMP GETs provide information about the general health of the sensor.
SDEE Server ³	Accepts requests for events from remote clients.
SensorApp	Captures and analyzes traffic on the monitored network and generates intrusion and network access events. Responds to IP logging control transactions that turn logging on and off and that send and delete IP log files.
SwitchApp	Part of the IPS 4500 series design that enables the InterfaceApp and sensor initialization scripts to communicate with and control the built-in switch. Any application that needs to get or set information on the switch must communicate with the SwitchApp.
Web Server	Waits for remote HTTP client requests and calls the appropriate servlet application.

1. This is a web server servlet.
2. This is a remote control transaction proxy.
3. This is a web server servlet.