Table of Contents

The Cisco IPS and CX Sensors

Introduction

In the modern networked world, secure networks cannot be taken for granted. Comprehensive network security is part of the solution to this problem, and network intrusion detection and prevention systems are an integral part of the security solution. The ability to provide timely notification of critical alerts is a valuable addition to the sensors, and this document introduces a Java application that allows the user to view sensor alerts.

How the sensor works

The Cisco IPS and CX sensors are intrusion detection and prevention system that monitor network traffic, generate alerts when attacks are detected, and block potential attacks. An attack occurs when someone tries to gain access or deny services to a host on the network.

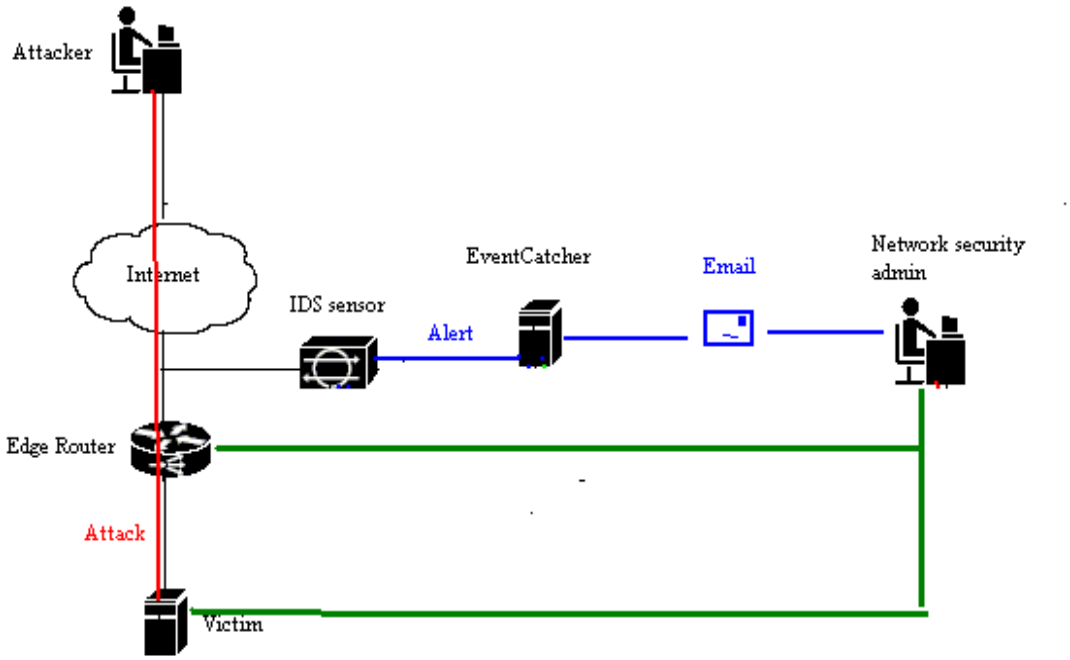

Figure 1 A Typical Attack

Figure 1 shows how a typical attack might develop. A hacker somewhere on the Internet transmits a packet containing an attack that exploits a SQL server vulnerability to a host on the protected network. When the packet enters the network edge router, a promiscuous sniffing interface of the IPS sensor or CX sensor analyzes the packet payload and determines that it matches the signature of a known attack. The sensor generates an internal high severity alert to signal the attack. However, unless someone requests it, the alert will exist merely as a record in the sensor event store. In this example, the EventCatcher application has opened a subscription on the sensor for high severity alerts and is waiting for the next event, so the sensor sends the alert in a SDEE event message. The alert is parsed and identified.

The sensors support open standards network protocols that allow users to write their own applications to retrieve and process alerts; the EventCatcher application is one of these. Basic configuration and monitoring of the sensor is performed in Command Prompt.

Communicating with the IPS Sensor

The sensor command and control interface supports a network API for device management and event notification. Because the API uses open standards, developers can write applications that communicate directly with the sensor. Requests are sent to IPS sensors using a subset of HTTP called Remote Data Exchange Protocol (RDEP). CX sensors only accept HTTP requests. Sensors are configured by default to accept HTTPS connections, which provide cryptographic security, although they can also be set to accept HTTP connections.

The sensor responds to event requests with SML response documents. The XML document structure is defined by the Security Device Event Exchange (SDEE) specification and extended by the Cisco Intrusion Device Event Exchange (CIDEE) specification. SDEE dictates that all sensor event responses consist of SOAP version 1.2 messages using Message Exchange Pattern format. Sensors provide communications security through HTTPS. By default, sensors are configured to require TLS/SSL encryption for all requests. Although sensors can be configured to allow plaintext HTTP connections, this is not recommended due to HTTP being less secure than HTTPS. Sensors authenticate individual users by requiring a username and password, or via exchange of X509 certificates. Once the sensor has authenticated a user, it establishes an SDEE session. This is not the same as a TCP connection. In fact, it is not necessary for consecutive requests to use the same TCP socket. The SDEE session is identified by a session ID returned in the open response, or if requested, by a session cookie. The RDEP client may use either one of these values in subsequent requests instead of sending the username and password. For expedience, EventCatcher simply sends the username and passwword with each request. New URL, URLConnection, and HttpUrlConnection objects are created for each request. EventCatcher leaves it to the underlying Java runtime to determine whether to leave the TCP connection open between requests.

The IPS sensor formats all responses in XML, and packages them in a SOAP container. JDOM is a straightforward and widely used means to parse XML; it uses whatever parser is specified by the developer or the default SAX parser. Since Java 1.4.2_06 has a built-in default SAX parser, it is not necessary to specify one. JDOM converts the XML into a Document object, and individual XML elements are stored as Element objects within the Document. SOAP elements are processed by JDOM in the same way as any other XML elements.

The EventCatcher Application

What is EventCatcher

EventCatcher is a command line Java application developed to interface with the Cisco IPS sensor. It opens a subscription for events on a Cisco sensor appliance. When run, it sends a request to the sensor for events, then reports said events in standard output for further use at the user's discretion. A sample alert is shown in “Sample Alert” on page 14.

The EventCatcher application is compiled using the Sun J2SE Java Development Kit (JDK) and executed with the Sun Java Runtime Environment (JRE). To download and install the tools necessary to support the EventCatcher application, go to “Downloading and Installing Java Development Kit (JDK)” on page 6.

How EventCatcher works

Events are retrieved from the sensor using RDEP requests. The RDEP request to get the next available event should set the action parameter to “get”. Several other parameters are available for use as well (see “Additional Parameter Options” on page 10).

EventCatcher receives events one at a time, so the URI will look like this:

```
https://192.168.1.1/cgi-bin/sdee-server?action=get&maxNbrOfE-
vents=1
```

When the sensor responds to a get events request, EventCatcher parses the response looking for children of the “events” element. SDEE defines a pull model for retrieving events from a sensor, meaning EventCatcher sends a request to the IPS sensor each time it wants to retrieve more events. Each time one is found, the signature ID, description, and element tree corresponding to the event is extracted. All alerts will contain a signature ID and description to uniquely identify the alert.

The application is written in Java since Java is platform independent with regards to this application and contains built-in support for HTTP connections, TLS/SSL encryption, and XML parsing. The XML is parsed by the default Java SAX library and converted into Java classes by the third party JSON library for the CX sensor. The application code will only need to format and process message content.

To support HTTPS, EventCatcher includes instances of X509TrustManager and HostnameVerifier. These classes implement Java interfaces to verify the X509 certificate and hostname of the sensor. In EventCatcher, these classes accept all certificates and names. But a real security admin would be advised to verify both in order to prevent man-in-the-middle attacks that could be used to make it seem that the sensor was not generating any alerts.

Because RDEP is implemented over HTTP, an RDEP message is also a valid HTTP message. For example, an open subscription request can be entered into a browser address bar. So a convenient way to debug the correctness of an open subscription request is to simply enter the URL line in your browser and observe what the sensor returns in the main browser window, as illustrated in Figure 2.

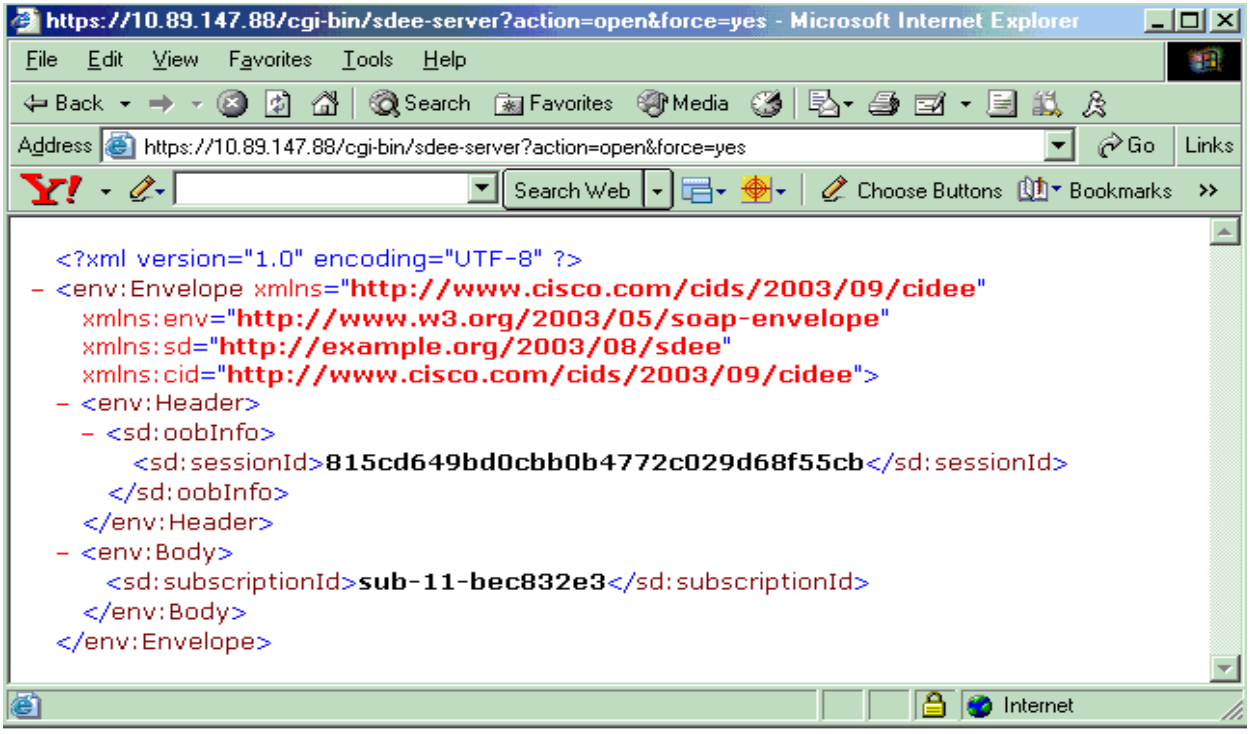

Figure 2 Using a browser to test an RDEP request

The Java code to send an RDEP request and receive the response can all be written using built in Java classes. Initialize a URL object with the RDEP URI. Create an HttpUrlConnection by opening a connection to the sensor. The sensor username and password is converted to base 64 encoding and added to the HttpUrlConnection authorization property. The sensor username and password were entered as EventCatcher command line parameters.

Opening a Subscription

SDEE specifies that the sensor should respond to successful open subscription requests by returning a subscription ID in the response XML. Subscriptions guarantee that all returned events are unique. There are some minor event subscription differences between Cisco IPS sensor versions and platforms, which are detailed in “Differences between Cisco IPS Sensor Implementations” on page 12.

The SDEE specification requests events via a subscription. Subscriptions guarantee that all returned events are unique, and each has a distinct subscription ID. A subscription ID works in a manner similar to a file handle. It is passed on all subsequent requests to the sensor, where it is used to determine the context of the subscription. In a “get” request, the sensor uses it to determine the next available event. In a close or cancel request, it identifies the subscription to be closed. Since this value must be included in future requests, it is saved as a class data member. EventCatcher checks the open subscription response for the subscription ID and saves it if found.

Opening a subscription starts with sending an RDEP open subscription request to the sensor. This is a simple, content-less HTTP GET message which consists of a URI in 3 parts: the sensor IP address, the sensor event servlet path specification, and the event servlet parameters. The sensor IP address and scheme (HTTP or HTTPS) is specified by the user as a command line parameter. The servlet path specification for a version 5 sensor is `/cgi-bin/sdee-server`. Event-

Catcher will send this URI, replacing the sensor IP address with the corresponding command line parameter:

```
https://192.168.1.1/cgi-bin/sdee-
server?action=open&force=yes&events=evIdsAlert&idsAlertSeveri-
ties=high
```

The force parameter ensures that the subscription will be opened even if the maximum number of subscriptions are already open on the sensor. In that case the oldest subscription will be closed in order to free up resources for the new subscription. Most IPS devices allow only three to five open subscriptions, and the subscriptions stay open for up to three days of inactivity, so during development it is advisable to use this parameter.

Closing a Subscription

To close a subscription, just set the action parameter value to “close”. For example,

```
http://192.168.1.1/cgi-bin/sdee-server?action=close
```

Downloading and Installing Java Development Kit (JDK)

Downloading Sun J2SE Java Development Kit and Sun Java Runtime Environment

1. Go to Oracle website at http://www.oracle.com/technetwork/java/index.html
2. Under Top Downloads, select “Java SE”.
3. Select the most recent JDK edition to download. There is no need to also download the JRE, as it will be downloaded alongside the JDK. Be sure to select “Accept License Agreement” to continue.
4. Select JDK from list recommended for your operating system.
5. Once the JDK is installed, click on application to run. Follow the prompts to finish installation.
6. For further installation instructions, see http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html.

Downloading and Compiling EventCatcher Application

1. Go to GitHub site https://github.com/stleary/Events and pull EventCatcher application.
2. Somehow download it?
3. Compile EventCatcher by typing the following in Command Prompt:
   ```
   javac EventCatcher.java
   ```
4. To view usage help page by executing the program with no command line parameters:
   ```
   java EventCatcher

   EventCatcher v1.0 Dec 28, 2005
   Usage:
   EventCatcher <sensorURL><user><passwd><to><from><emailServer>
   [v]
   ```
   This ensures that EventCatcher has downloaded and compiled correctly. The help page will print to standard output and look similar to this:

```
Usage:
   Events sensorURL -u user/passwd [-d deviceType] [-o open
   params] [-g get params] [-v]
   -u sensor username and password, separated by the / char
   -d device type: ipsv4, ips, cx. By default CX will be selected
   -o The URI parameters included in the Open Subscription
   request.
   Open params: force (5.0 only), startTime, events, alertSeveri-
   ties, errorSeverities, ustHaveAlarmTraits, mustNotHaveAlarm-
   Traits.
   Force: yes, no
   Events: evStatus, evShunRqst, evError, evLogTransaction (4.x
   only), evAlert (4.x only), evIdsAlert (5.0 only).
   Concatenate multiple event types with the+ char.
   AlertSeverities: informational, low, medium,high.
   ErrorSeverities: debug, warning, error, fatal.
   -g The URI parameters included in the Get Subscription request.
   Get params: timeout, maxNbrOfEvents, confirm.
   -v Verbose for additional messages.
   EventCatcher establishes an event subscription to a Cisco sen-
   sor and retrieves events.
   Press control-C to exit.
Example: get error events starting at the current time from a 5.0
sensor
   java EventCatcher  https://192.168.1.1 -u cisco/password -d ips
   -o "force=yes&events=evError" -g timeout=10
Example: get all events from a 4.x sensor
   java EventCatcher https://192.168.1.2 -u cisco/password -d
   ipsv4
   -o startTime=0 -g timeout=10
Example: get all events from a CX sensor
```

```
java EventCatcher https://192.168.1.2 -u cisco/password
```

Running EventCatcher From Command Prompt

1. Execute EventCatcher from Command Prompt using the following parameters:
   `java EventCatcher <sensorURL> -u <username>/<password> [-o open params] [-g get params] [-v]`
2. Optional step: Specifying start time.
   If the `startTime` parameter is not specified, EventCatcher is only going to retrieve events that occur after the subscription has started. To get all events in the sensor event store, add parameter:
   `startTime=0`
   To retrieve all events from a specific date, the `startTime` parameter must be defined in nanoseconds from the Unix Epoch (00:00:00 UTC). For example, to retrieve all events from midnight, March 8, 2015, the `startTime` parameter would be defined thusly:
   `startTime=1425772800000000000`
3. Optional step: Specifying event types.
   EventCatcher automatically reports all four event types when run. To select for only certain event types, add parameter
   `events=<event type>+<event type>`
   If selecting for multiple event types, separate types by a plus sign. Event types are listed and described in“Event Type Values and Definitions” on page 13.
4. Optional step: Specifying alert severity.
   EventCatcher automatically reports all evIdsAlert (evAlert for 4.x) events with any alert severity values. To specify only certain alert values, add parameter
   `idsAlertSeverities=<alert severity value>+<alert severity value>`
   As before, separate multiple values with a plus sign. Alert severity values are listed and described in “Additional Parameter Options” on page 10.
5. Several other parameters may be defined as needed. They are shown in “Additional Parameter Options” on page 10.

Additional Parameter Options

Table 1: Get Subscription Parameters

Parameter Name	Description
action	Common to all types of subscription requests. To get more events, set value to “get”.
timeout	If specified, then the sensor will return a response in specified number of seconds, even if no events are available. If not specified, then the sensor may block indefinitely. If events are available, they will be sent immediately; the sensor does not block until the timeout expires if at least one qualifying event is ready to be sent. If no timeout value is specified, the sensor waits for the maximum time possible before returning, unless at least one event is ready to be sent. If the timeout value is 0, then the sensor returns immediately even if no events are available.
maxNbrOfEvents	If specified, then at most this number of events will be returned in each request. However, the sensor will return only as many events are immediately available. It will not wait until the specified number of events are available before responding.
confirm	If specified “no”, the sensor interprets this to mean the previous get events response was not received, and will send these events again. The default value is “yes”, so it does not need to be specified.

Table 2: Open Subscription Parameters

Parameter Name	Description
action	Required for all subscription requests.
force	The subscription will be forced open even if all available subscriptions are already active.
startTime	Events that occur after this time will be returned. Time is defined in nanoseconds from the UNIX epoch.
stopTime	Events that occur after this time are not returned. Time is defined in nanoseconds from the UNIX epoch.

Table 2: Open Subscription Parameters

Parameter Name	Description
events	Specifies alert type. See “Event Type Values and Definitions” on page 13
idsAlertSeverities	Specifies alert severity. SDEE and CIDEE define four idsAlertSeverity values: informational, low, medium, and high. These are defined internal to the sensor when it is configured.
errorSeverities	Specifies error severities between warning, error, and fatal.
mustHaveAlarmTraits	Restricts alerts to those with the specified traits.
mustNotHaveAlarmTraits	Restricts alerts to those without the specified traits.

Differences between Cisco IPS Sensor Implementations

There are minor differences between Cisco implementations of the SDEE event server. There are currently three flavors of IPS sensors: v4 sensors, v5 and up sensors, and IOS-IDS routers. IOS-IDS routers only allow three active subscriptions, but v4 and v5 sensors allow five active subscriptions at a time. The IOS-IDS and v4 sensor event server servlet name is “event-server”, and messages should specify that name on the URI line. The v5 sensor event server servlet name is “sdee-server”.

V5 sensors are backwards compatible with v4 sensors with regards to event subscriptions, so if you write a single v4 implementation it will return events from all types of sensors. However, the v5 implementation is richer and is the recommended solution if you have v5 sensors. Also, this backwards compatibility is not guaranteed in future releases.

Event Type Values and Definitions

The Cisco IPS and CX sensors implement six event types derived from SDEE and CIDEE. EventCatcher automatically reports all event types when run, but inclusion of the `events` parameter allows the user to filter out and select for only certain event types to be printed to standard output.

Table 1: Event Type Parameter Values

Value	Description
evStatus	Reports internal sensor event.
evShunRqst	Sensor instructs external device (e.g., firewall, switch) to not let traffic through from specific IP address.
evError	Reports internal error in sensor.
evLogTransaction	4.x only; no longer in use.
evAlert	4.x only; no longer in use.
evIdsAlert	5.0 only; replaced evAlert. Reports an attack. The alert specifies details about the attack and actions taken by the sensor.

Sample Alert

The following is a sample standard output alert that may be seen when running EventCatcher. Note the event type, severity, and time of attack. This is a high severity evIdsAlert.

```
<sd:evIdsAlert
xmlns:sd="http://example.org/2003/08/sdee"
eventId="11239497526821770 81" vendor="Cisco"
severity="high"
   alarmTraits="16">
   <sd:originator>
   <sd:hostId>Sensor1</sd:hostId>
   <cid:appName

xmlns:cid="http://www.cisco.com/cids/2003/09/cidee">
   sensorApp</cid:appName>
   <cid:appInstanceId

xmlns:cid="http://www.cisco.com/cids/2003/09/cidee">
   8971</cid:appInstanceId>
   </sd:originator>
   <sd:time>1124581173934620000</sd:time>
   <sd:signature description="UDP Bomb" id="4050">
   <cid:subsigId

xmlns:cid="http://www.cisco.com/cids/2003/09/cidee">
   0</cid:subsigId>
   </sd:signature>
</sd:evIdsAlert>
```

Structure of Event Reports

HttpUrlConnection objects return an InputStream object that contains data sent by the sensor. The code to convert an InputStream to a JDOM Document is straightforward:

```
import java.io.*;
import org.jdom.*;

InputStream response = httpUrlConnection.getInputStream();
InputStreamReader in = new InputStreamReader(response);
BufferedReader reader = new BufferedReader(in);
StringBuffer str = new StringBuffer();
String line = null;
while((line = reader.readLine()) != null){
   str.append(line);
   str.append('\n');
SAXBuilder builder = new SAXBuilder();
StringReader sr = new StringReader(str.toString());
Document doc = builder.build(sr);
```

Glossary of Terms

alert

An event the sensor recognizes as an attack.

API

Application Program Interface. The set of routines, protocols, and tools for building software applications.

CIDEE

Cisco Intrusion Detection Event Exchange. Cisco extension for SDEE.

Cisco CX Sensor

Includes the capabilities of the IPS sensor and add on deep packet inspection and reputation.

Cisco IPS Sensor

Cisco Intrusion Prevention System Sensor. Monitors network traffic and generates alerts when attacks are detected.

CLI

Command Line Interface. This project utilizes Command Prompt.

event

Any notification that the sensor sends out, including errors, general updates, and alerts.

EventCatcher

A command line Java application developed to allow users to retrieve events from IPS or CX sensors.

HostnameVerifier

Internal Java class that helps establish a secure connection.

HTTP

Hypertext Transfer Protocol. Functions as network data communication.

HTTPS

Secure communications protocol developed by layering HTTP with TLS/SSL protocols.

IDM

Intrusion Device Manager. A web-based application for configuring and managing sensors.

InputStream

Java abstract class.

IP address

Internet Protocol address. Unique number assigned to every device on a network.

JDK

adfad

JDOM

adf

JRE

adfada

RDEP

Remote Data Exchange Protocol. Designed by Cisco for use in sensor communication and event logging.

SAX

afdasadf

SDEE

Security Device Event Exchange. Sponsored by ISCA Labs. SDEE is a de facto vendor independent standard for intrusion detection alerts.

SML

Standard ML. General-use programming language.

SMTP

Simple Mail Transfer Protocol. Internet standard for email transmission.

SOAP

Simple Object Access Protocol. For exchanging structured information. Used in conjunction with HTTP and SMTP for message transmission.

SQL

Structured Query Language. Special purpose programming language designed for managing data.

TCP

Transmission Control Protocol. Delivers stream of octets between applications on a network.

TLS/SSL

Transport Layer Security. A cryptographic protocol used to provide authentication and ensure privacy of network communications. Original version was called Secure Sockets Layer.

URI

Uniform Resource Identifier. Character string used to identify resource.

X509TrustManager

Public interface in Java.

XML

Extensible Markup Language. Rules for encoding a document so that it is readable to machines and people.