Configure AMP for Endpoints Event Stream Feature

Contents

Introduction

This document describes how to configure and consume the Event Stream feature for Advanced Malware Protection (AMP) for Endpoints.

Prerequisites

Requirements

Cisco recommends that you have knowledge of the following topics:

- AMP for Endpoints
- Basic knowledge of Python programming

Components Used

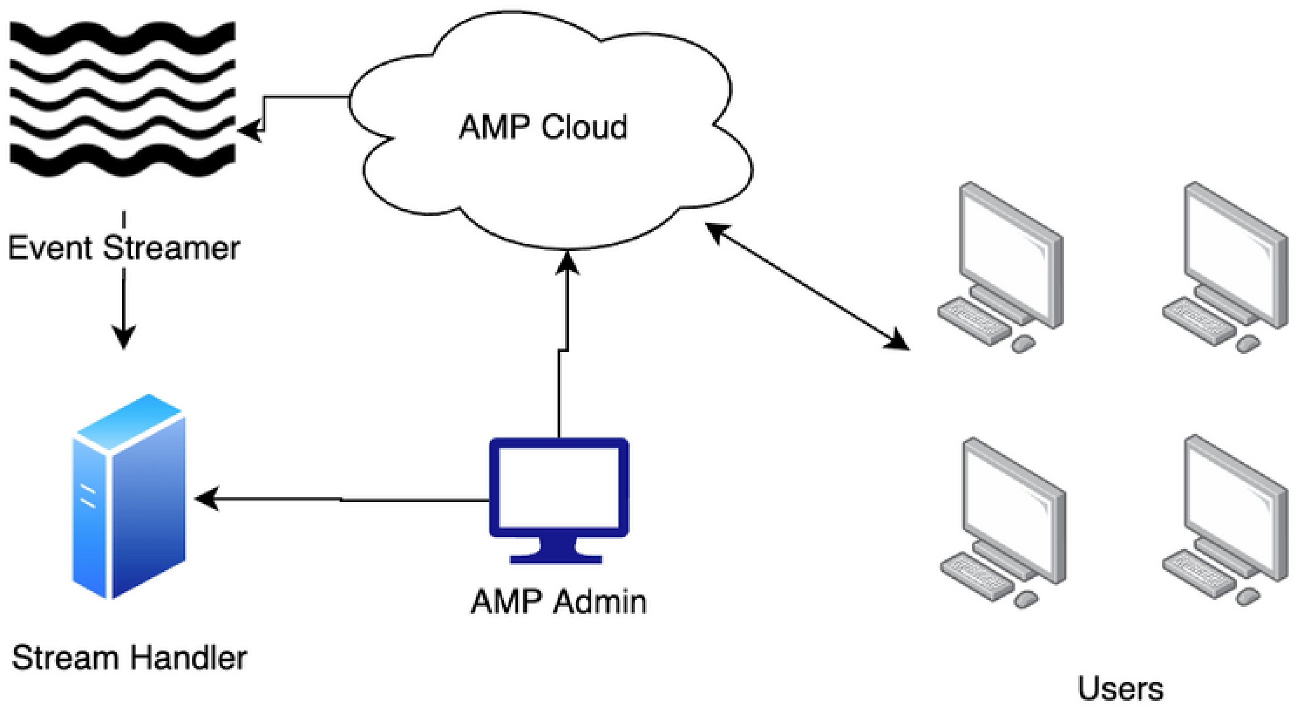
The information in this document is based on Python 3.7 with the **pika** (version 1.1.0) and **requests** (version 2.22.0) external libraries.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Configure

Network Diagram

This image provides an example of Event Stream sequencing:



Configurations

Create API credentials

1. Navigate to your AMP for Endpoints portal and login
2. Under **Accounts,** choose **API Credentials**
3. Click **New API Credentials**
4. Enter a value in the **Application name** field
5. Select **Read & Write** for **Scope**
6. Click **Create**
7. Store these credentials in a password manager or encrypted file

Create Event Stream

1. Open a Python shell and import the **json**, **ssl**, **pika** and **requests** libraries.

```
import json
import pika
import requests
import ssl
```

2. Store the values for the url, client_id, and api_key. Your URL can vary if you are not using the North American cloud. Also, your client_id and api_key is unique to your environment.

```
url = "https://api.amp.cisco.com/v1/event_streams"
client_id = "d16aff14860af496e848"
```

```
api_key = "d01ed435-b00d-4a4d-a299-1806ac117e72"
```

3. Create the data object to pass to the request. This must include name, and can include event_type and group_guid to restrict the events and groups included in the stream. If no group_guid or event_type is passed, the event stream will include all groups and event types.

```
data = {
    "name": "Event Stream for ACME Inc",
    "group_guid": ["5cdf70dd-1b14-46a0-be90-e08da14172d8"],
    "event_type": [1090519054]
}
```

4. Make the POST request call, and store the value in a variable.

```
r = requests.post(
    url = url,
    data = data,
    auth = (client_id, api_key)
)
```

5. Print the status code. Confirm that the code is 201.

```
print(r.status_code)
```

6. Load the content of the response into a json object, and store that object in a variable.

```
j = json.loads(r.content)
```

7. Review the contents of the response data.

```
for k, v in j.items():
    print(f"{k}: {v}")
```

8. The Advanced Message Queuing Protocol (AMQP) data is inside the response. Extract the data into respective variables.

```
user_name = j["data"]["amqp_credentials"]["user_name"]
queue_name = j["data"]["amqp_credentials"]["queue_name"]
password = j["data"]["amqp_credentials"]["password"]
host = j["data"]["amqp_credentials"]["host"]
port = j["data"]["amqp_credentials"]["port"]
proto = j["data"]["amqp_credentials"]["proto"]
```

9. Define a callback function with these parameters. In this setup, you print the body of the event to the screen. However, you can change this content of this function to suit your objectives.

```
def callback(channel, method, properties, body):
    print(body)
```

10. Prepare the AMQP URL from the variables you created.

```
amqp_url = f"amqps://{user_name}:{password}@{host}:{port}"
```

11. Prepare the SSL context

```
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
amqp_ssl = pika.SSLOptions(context)
```

12. Prepare the AMQP stream with the pika library methods.

```
params = pika.URLParameters(amqp_url)
params.ssl_options = amqp_ssl

connection = pika.BlockingConnection(params)
channel = connection.channel()

channel.basic_consume(
    queue_name,
    callback,
    auto_ack = False
)
```

13. Initiate the stream.

```
channel.start_consuming()
```

14. The stream is now live and awaiting events.

Verify

Trigger an event on an endpoint in your environment. For example, initiate a flash scan. Notice the stream prints the event data to the screen.

Press **Ctrl+C** (Windows) or **Command-C** (Mac) to interrupt the stream.

Troubleshoot

Status Codes

- A status code of 401 indicates there is an issue with authorization. Check your **client_id** and **api_key**, or generate new keys.
- A status code of 400 indicates there is a Bad Request issue. Check that you don't have an Event Stream created with that name, or that you don't have more than 5 Event Streams created.