SNMP Trap Integration: Splunk to CA Spectrum

# 1. Overview

This document outlines the implementation details for integrating Splunk with CA Spectrum using SNMP traps. Events from Splunk trigger alerts that are sent via a webhook to a FastAPI-based application, which formats and sends SNMP traps to CA Spectrum.

# 2. Architecture

Splunk triggers alerts based on predefined conditions. These alerts are sent to the FastAPI application via webhooks. The FastAPI application processes the incoming payload, formats SNMP traps, and sends them to CA Spectrum.

Workflow:

1. Splunk Event Trigger  
2. Webhook sends alert to FastAPI App  
3. App formats the alert and generates SNMP traps  
4. SNMP trap is sent to CA Spectrum

# 3. Prerequisites

## Splunk Configuration

Splunk instance with necessary event-based searches.  
Admin access to configure alert webhooks.

## FastAPI Application

Python 3.8 or later  
Dependencies:  
pip install fastapi uvicorn pysnmp pydantic

## CA Spectrum

Ensure SNMP trap receiver is configured.  
Open firewall ports for incoming SNMP traps (default port: 162).

## Credentials File

credentials.json should store SNMPv3 authentication information:  
{  
 "snmpv3\_user": "your\_snmp\_user",  
 "auth\_password": "your\_auth\_password",  
 "priv\_password": "your\_priv\_password",  
 "auth\_protocol": "SHA",  
 "priv\_protocol": "AES"  
}

# 4. FastAPI Application

Below is the Python FastAPI code used to send SNMP traps to CA Spectrum:  
  
import json  
import threading  
import queue  
from fastapi import FastAPI  
from pydantic import BaseModel  
from pysnmp.hlapi import \*  
from pysnmp.proto.rfc1902 import OctetString  
  
# FastAPI instance  
app = FastAPI()  
  
# Queue to handle SNMP trap requests  
trap\_queue = queue.Queue()  
  
# Function to load SNMPv3 credentials from a JSON file  
def load\_snmpv3\_credentials(file\_path='credentials.json'):  
 try:  
 with open(file\_path, 'r') as f:  
 return json.load(f)  
 except Exception as e:  
 print(f'Error loading SNMP credentials: {e}')  
 return None  
  
# Function to send SNMPv3 traps  
def send\_snmp\_trap(oids, snmp\_credentials):  
 try:  
 user = snmp\_credentials['snmpv3\_user']  
 auth\_password = snmp\_credentials['auth\_password']  
 priv\_password = snmp\_credentials['priv\_password']  
 auth\_protocol = usmHMACSHAAuthProtocol if snmp\_credentials['auth\_protocol'] == 'SHA' else usmHMACMD5AuthProtocol  
 priv\_protocol = usmAesCfb128Protocol if snmp\_credentials['priv\_protocol'] == 'AES' else usmDESPrivProtocol  
  
 errorIndication, \_, \_, \_ = next(  
 sendNotification(  
 SnmpEngine(),  
 UsmUserData(user, auth\_password, priv\_password, authProtocol=auth\_protocol, privProtocol=priv\_protocol),  
 UdpTransportTarget(('192.168.1.100', 162)), # Replace with your SNMP receiver IP  
 ContextData(),  
 'trap',  
 NotificationType(  
 ObjectIdentity('1.3.6.1.4.1.12345.1.1.1')  
 .addVarBinds(\*oids)  
 )  
 )  
 )  
  
 if errorIndication:  
 print(f'Error sending SNMP trap: {errorIndication}')  
 return {"status": "error", "message": str(errorIndication)}  
  
 print('SNMP trap sent successfully!')  
 return {"status": "success", "message": "SNMP trap sent successfully"}  
  
# Worker function to process SNMP traps from the queue  
def trap\_worker(snmp\_credentials):  
 while True:  
 try:  
 oids = trap\_queue.get()  
 if oids is None:  
 break  
 send\_snmp\_trap(oids, snmp\_credentials)  
 trap\_queue.task\_done()  
 except Exception as e:  
 print(f'Worker error: {e}')  
  
# Start worker threads  
num\_workers = 5  
threads = []  
snmp\_credentials = load\_snmpv3\_credentials()  
  
if snmp\_credentials:  
 for \_ in range(num\_workers):  
 t = threading.Thread(target=trap\_worker, args=(snmp\_credentials,))  
 t.daemon = True  
 t.start()  
 threads.append(t)  
  
# Pydantic model for API request  
class SNMPTrapRequest(BaseModel):  
 source: str  
 severity: str  
 timestamp: str  
 message: str  
 application: str  
 region: str  
  
# API Endpoint to send SNMP trap  
@app.post("/send\_snmp\_trap/")  
def api\_send\_snmp\_trap(request: SNMPTrapRequest):  
 oids = [  
 ('1.3.6.1.4.1.12345.1.2.1', OctetString(request.source)),  
 ('1.3.6.1.4.1.12345.1.2.2', OctetString(request.severity)),  
 ('1.3.6.1.4.1.12345.1.2.3', OctetString(request.timestamp)),  
 ('1.3.6.1.4.1.12345.1.2.4', OctetString(request.message)),  
 ('1.3.6.1.4.1.12345.1.2.5', OctetString(request.application)),  
 ('1.3.6.1.4.1.12345.1.2.6', OctetString(request.region))  
 ]  
  
 trap\_queue.put(oids)  
 return {"status": "queued", "message": "SNMP trap request added to queue"}  
  
# Run FastAPI server  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
 uvicorn.run(app, host="0.0.0.0", port=8002)

# 5. Deployment

Save the above code as snmp\_trap\_app.py.  
Create the credentials.json file.  
Start the application:  
python snmp\_trap\_app.py  
Configure Splunk webhook as mentioned earlier.

# 6. Testing

Using Curl:  
curl -X POST "http://localhost:8002/send\_snmp\_trap/" -H "Content-Type: application/json" -d '{"source": "TestSource", "severity": "High", "timestamp": "2025-02-05T10:00:00Z", "message": "Sample SNMP Trap Message", "application": "TestApp", "region": "NA"}'  
Expected Response:  
{"status":"queued","message":"SNMP trap request added to queue"}  
  
Monitoring CA Spectrum:  
Check the Spectrum Event Manager to ensure SNMP traps are received.  
Use tools like Wireshark to monitor trap transmissions.

# 7. Conclusion

This integration provides a robust solution for forwarding Splunk events to CA Spectrum using SNMP traps. By leveraging FastAPI and SNMPv3, secure and reliable alerting can be maintained.