Create custom devices through the REST API

Published: 2019-02-15

You can create custom devices through the REST API that track network traffic across multiple IP addresses and ports. For example, you might want to add a custom device for each branch office. If you create the devices through a script, you can read the list of devices from a CSV file. In this topic, we will demonstrate methods for both the REST API and the ExtraHop REST API Explorer.

Before you begin

- You must log into the ExtraHop appliance with an account that has unlimited privileges to generate an API key.
- You must have a valid API key to make changes through the REST API and complete the procedures below. (See Generate an API key.)
- Familiarize yourself with the ExtraHop REST API Guide to learn how to navigate the ExtraHop REST API Explorer.

Create a custom device

You can create a custom device and associate the custom device with a list of IP addresses or CIDR blocks through the **POST /customdevices** operation.

1. In the REST API Explorer, click **Custom Device**, and then click **POST /customdevices**.
2. In the body field, specify properties for the custom device that you want to create.

   For example, the following body matches the custom device to the CIDR blocks 192.168.0.0/26, 192.168.0.64/27, 192.168.0.96/30, and 192.168.0.100/32:

```
{
  "description": "The location of our office in Washington",
  "name": "Seattle",
  "criteria": [
    {
      "ipaddr": "192.168.0.0/26"
    },
    {
      "ipaddr": "192.168.0.64/27"
    },
    {
      "ipaddr": "192.168.0.96/30"
    },
    {
      "ipaddr": "192.168.0.100/32"
    }
  ]
}
```

Python script example

This example python script creates custom devices by reading criteria from a CSV file. Each row of the CSV file must contain the following columns in the specified order:

Name	ID	Description	IP address or CIDR block

Note: The script does not accept a header row in the CSV file. There is no limit to the number of columns in the table; each column after the first four specifies an additional IP address for the device. The first four columns are required for each row.

For example, the following CSV list contains criteria for offices in France, Holland, and California:

```
France,francehq,The location of our office in
 France,192.168.0.103,192.168.0.105,192.168.0.101
Holland,hollandhq,The location of our office in Holland,192.168.0.102
California,californiahq,The location of our office in
 California,192.168.0.104,192.168.0.103
```

The script includes the following configuration variables:

- **HOST:** The IP address or hostname of the Discover appliance
- **APIKEY:** The API key
- **CSV_FILE:** The path of the CSV file relative to the location of the script file

```
#!/usr/bin/env python2

import json
import httplib
import csv
import os.path

HOST = 'example.extrahop.com'
APIKEY = 'f6876657888a7c1f24ac77827'
CSV_FILE = 'device_list.csv'

headers = {'Content-Type': 'application/json',
           'Accept': 'application/json',
           'Authorization': 'ExtraHop apikey=%s' % APIKEY}

def readCSV():
    devices = []
    with open(CSV_FILE, 'rb') as f:
        reader = csv.reader(f)
        for row in reader:
            device = {}
            ips = []
            device['name'] = row.pop(0)
            device['extrahop_id'] = row.pop(0)
            device['description'] = row.pop(0)
            for ip in row:
                ips.append({"ipaddr": ip})
            device['criteria'] = ips
            devices.append(device)
    return devices

def createDevice(device):
    conn = httplib.HTTPSConnection(HOST)
    conn.request('POST', '/api/v1/customdevices', body=json.dumps(device),
 headers=headers)
    resp = conn.getresponse()
    if resp.status != 201:
        print "Could not create device: " + device['name']
        print "    " + json.loads(resp.read())['error_message']
    else:
```

```
        print "Created custom device: " + device['name']
        device_id = os.path.basename(resp.getheader('location'))

devices = readCSV()
for device in devices:
    createDevice(device)
```

Note: If the script returns an error message that the SSL certificate verification failed, make sure that a trusted certificate has been added to your appliance. Alternatively, you can add the context option to send the request over TLSv1.2 and bypass certificate verification; however, this method is not secure and is not recommended. The following code creates an HTTP connection over TLSv1.2:

```
conn = httplib.HTTPSConnection(HOST,
 context=ssl.SSLContext(ssl.PROTOCOL_TLSv1_2))
```