Import external data to your Discover appliance

Published: 2020-02-23

The ExtraHop Open Data Context API enables you to import data from an external host into the session table on your Discover appliance. That data can then be accessed to create custom metrics that you can add to ExtraHop charts, store in records on an Explore appliance, or export to a external analysis tool.

After you enable the Open Data Context API on your Discover appliance, you can import data by running a Python script from a memcached client on an external host. That external data is stored in key-value pairs, and can be accessed by writing a trigger.

For example, you might run a memcached client script on an external host to import CPU load data into the session table on your Discover appliance. Then, you can write a trigger that accesses the session table and commits the data as custom metrics.

Warning: The connection between the external host and the ExtraHop appliance is not encrypted and should not transmit sensitive information.

Enable the Open Data Context API

You must enable the Open Data Context API on your Discover appliance before it can receive data from an external host.

Before you begin

- You must have unlimited privileges to access the Admin UI on your Discover appliance.
- If you have a firewall, your firewall rules must allow external hosts to access the specified TCP and UDP ports. The default port number is 11211.

1. Log into the Admin UI on the Discover appliance.
2. In the System Configuration section, click **Capture**.
3. Click **Open Data Context API**.
4. Click **Enable Open Data Context API**.
5. Configure each protocol that you want to allow external data transmissions through:

Option	Description
TCP	1. Select the **TCP Port enabled** checkbox. 2. In the **TCP Port** field, type the port number that will receive external data.
UDP	1. Select the **UDP Port enabled** checkbox. 2. In the **UDP Port** field, type the port number that will receive external data.

6. Click **Save and Restart Capture**.

   Important: The appliance will not collect metrics while it is restarting.

7. Click **Done**.

© 2020 ExtraHop Networks, Inc. All rights reserved.

Write a Python script to import external data

Before you can import external data into the session table on your Discover appliance, you must write a Python script that identifies your Discover appliance and contains the data you want to import into the session table. The script is then run from a memcached client on the external host.

This topic provides syntax guidance and best practices for writing the Python script. A complete script example is available at the end of this guide.

Before you begin

Ensure that you have a memcached client on the external host machine. You can install any standard memcached client library, such as http://libmemcached.org/ or https://pypi.python.org/pypi/pymemcache. The Discover appliance acts as a memcached version 1.4 server.

Here are some important considerations about the Open Data Context API:

- The Open Data Context API supports most memcached commands, such as `get`, `set`, and `increment`.
- All data must be inserted as strings that are readable by the Discover appliance. Some memcached clients attempt to store type information in the values. For example, the Python memcache library stores floats as pickled values, which cause invalid results when calling `Session.lookup` in triggers. The following Python syntax correctly inserts a float as a string:

```
mc.set("my_float", str(1.5))
```

- Although session table values can be almost unlimited in size, committing large values to the session table might cause performance degradation. In addition, metrics committed to the datastore must be 4096 bytes or fewer, and oversized table values might result in truncated or imprecise metrics.
- Basic statistics reporting is supported, but detailed statistics reporting by item size or key prefix is not supported.
- Setting item expiration when adding or updating items is supported, but bulk expiration through the `flush` command is not supported.
- Keys expire at 30-second intervals. For example, if a key is set to expire in 50 seconds, it can take from 50 to 79 seconds to expire.
- All keys set with the Open Data Context API are exposed through the `SESSION_EXPIRE` trigger event as they expire. This behavior is in contrast to the Trigger API, which does not expose expiring keys through the `SESSION_EXPIRE` event.

1. In a Python editor, open a new file.
2. Add the IP address of your Discover appliance and the port number where the memcached client will send data, similar to the following syntax:

```
client = memcache.Client(["eda_ip_address:eda_port"])
```

3. Add the data you want to import to the session table through the memcached `set` command, formatted in key-value pairs, similar to the following syntax:

```
client.set("some_key", "some_value")
```

4. Save the file.
5. Run the Python script from the memcached client on the external host.

Write a trigger to access imported data

You must write a trigger before you can access the data in the session table.

Before you begin

This topic assumes experience with writing triggers. If you are unfamiliar with triggers, check out the following topics:

- Triggers
- Build a trigger
- Learn how to build a trigger to collect custom metrics

1. Log into the Web UI on the ExtraHop Discover or Command appliance.
2. Click the System Settings icon and then click **Triggers**.
3. Click **New**, and then click the Configuration tab.
4. In the **Name** field, type a unique name for the trigger.
5. In the **Events** field, begin typing an event name and then select an event from the filtered list.
6. Click the **Editor** tab.
7. In the Trigger Script textbox, write a trigger script that accesses and applies the session table data. A complete script example is available at the end of this guide.

   The script must include the `Session.lookup` method to locate a specified key in the session table and return the corresponding value.

   For example, the following code looks up a specific key in the session table to return the corresponding value, and then commits the value to an application as a custom metric:

   ```
   var key_lookup = Session.lookup("some_key");
                    Application("My
    App").metricAddDataset("my_custom_metric",
                    key_lookup);
   ```

   Tip: You can also add, modify, or delete key-value pairs in the session table through methods described in the Session class of the ExtraHop Trigger API Reference.
8. Click **Save and Close**.

Next steps
You must assign the trigger to a device or device group. The trigger will not run until it has been assigned.

Open Data Context API example

In this example, you will learn how to check the reputation score and potential risk of domains that are communicating with devices on your network. First, the example Python script shows you how to import domain reputation data into the session table on your Discover appliance. Then, the example trigger script shows you how to check IP addresses on DNS events against that imported domain reputation data and how to create a custom metric from the results.

Example Python script

This Python script contains a list of 20 popular domain names and can reference domain reputation scores obtained from a source such as DomainTools.

This script is a REST API that accepts a POST operation where the body is the domain name. Upon a POST operation, the memcached client updates the session table with the domain information.

```
#!/usr/bin/python
import flask
import flask_restful
import memcache
import sqlite3

top20 = { "google.com", "facebook.com", "youtube.com", "twitter.com",
          "microsoft.com", "wikipedia.org", "linkedin.com",
```

```
          "apple.com","adobe.com", "wordpress.org", "instagram.com",
          "wordpress.com", "vimeo.com", "blogspot.com", "youtu.be",
          "pinterest.com", "yahoo.com", "goo.gl", "amazon.com", "bit.ly}

dnsnames = {}

mc = memcache.Client(['10.0.0.115:11211'])

for dnsname in top20:
    dnsnames[dnsname] = 0.0

dbc = sqlite3.Connection('./dnsreputation.db')
cur = dbc.cursor()
cur.execute('select dnsname, score from dnsreputation;')
for row in cur:
    dnsnames[row[0]] = row[1]
dbc.close()

app = flask.Flask(__name__)
api = flask_restful.Api(app)

class DnsReputation(flask_restful.Resource):
    def post(self):
        dnsname = flask.request.get_data()
        #print dnsname
        mc.set(dnsname, str(dnsnames.get(dnsname, 50.0)), 120)
        return 'added to session table'

api.add_resource(DnsReputation, '/dnsreputation')

if __name__ == '__main__':
     app.run(debug=True,host='0.0.0.0')
```

Example trigger script

This example trigger script canonicalizes (or converts) IP addresses that are returned on DNS events into domain names, and then checks for the domain and its reputation score in the session table. If the score value is greater than 75, the trigger adds the domain to an application container called "DNSReputation" as a detail metric called "Bad DNS reputation".

```
//Configure the following trigger settings:
//Name: DNSReputation
//Debugging: Enabled
//Events: DNS_REQUEST, DNS_RESPONSE

if (DNS.errorNum != 0 || DNS.qname == null
    || DNS.qname.endsWith("in-addr.arpa") || DNS.qname.endsWith("local")
    || DNS.qname.indexOf('.') == -1 ) {
    // error or null or reverse lookup, or lookup of local namereturn
    return;
}

//var canonicalname = DNS.qname.split('.').slice(-2).join('.');
var canonicalname = DNS.qname.substring(DNS.qname.lastIndexOf('.',
 DNS.qname.lastIndexOf('.')-1)+1)

//debug(canonicalname);


//Look for this DNS name in the session table
var score = Session.lookup(canonicalname)
if (score === null) {
```

```
    // Send to the service for lookup
    Remote.HTTP("dnsrep").post({path: "/dnsreputation", payload:
 canonicalname});
} else {
    debug(canonicalname + ':' +score);
    if (parseFloat(score) > 75) {
  //Create an application in the Web UI and add custom metrics
        //Note: The application is not displayed in the Web UI after the
  //initial request, but is displayed after subsequent requests.
        Application('DNSReputation').metricAddDetailCount('Bad DNS
 reputation', canonicalname + ':' + score, 1);
    }
}
```