Creating Models

Date of Publish: 2019-04-09

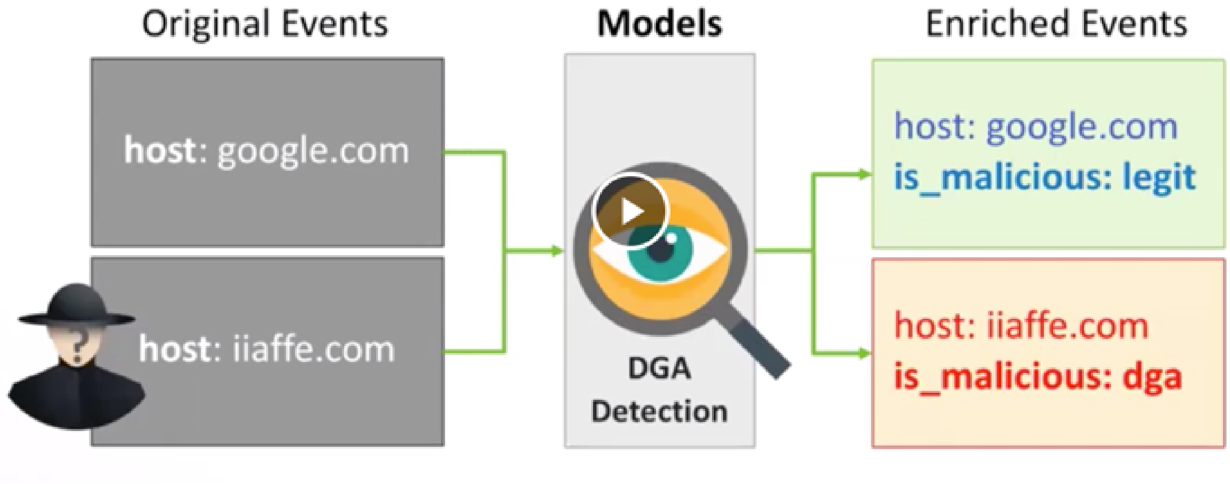
https://docs.hortonworks.com

Contents

Creating Models

One of the enhancements to cybersecurity most frequently requested is the ability to augment the threat intelligence and enrichment processes with insights derived from machine learning and statistical models. While valuable, this model management infrastructure has significant challenges.

You can add a model to an enrichment stream to identify malware data. For example, you might want to use a Domain Generation Algorithm (DGA) detection model to detect generated as opposed to authentic domains. The following figure illustrates the data flow and output if you are using a DGA detection model:



While valuable, the model management infrastructure has significant challenges.

- Applying the model management infrastructure might be both computationally and resource intensive and could require load balancing and multiple versions of models.
- Models require frequent training or updating to react to growing threats and new patterns that emerge.
- Models should be language and environment agnostic as much as possible. So, models should include small-data and big-data libraries and languages.

To support these requirements, Hortonworks Cybersecurity Platform (HCP) powered by Metron provides the following components:

- A YARN application that listens for model deployment requests and upon execution, registers their endpoints in ZooKeeper.
- A command line deployment client that localizes the model payload onto HDFS and submits a model request.
- A Java client that interacts with ZooKeeper and receives updates about model state changes (for example, new deployments and removals).
- A series of Stellar functions for interacting with models deployed by the Model as a Service infrastructure.

Set up Model as a Service

The YARN application listens for model deployment requests. Models are exposed as REST microservices that expose your model application as an endpoint. The YARN application takes the submitted request that specifies the model payload that includes a shell script and other model collateral which will start the microservice. Upon execution of the shell script that starts the model, the YARN application registers the endpoints in ZooKeeper.

About this task

If you are using or depending upon an API library in your model such as Flask and Jinja2, the library must be installed on every data node. This is because the model is executed by a shell script which must be able to run successfully on every node.

In order to know on which port that the REST service is listening, the model must create a file in the current working directory which indicates the URL for the model. Because you might have more than one copy of the model, it is a good idea to find an open port and bind to that. An example of how to do that in Python is as follows:

```
  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost', 0))
port = sock.getsockname()[1]
sock.close()
with open("endpoint.dat", "w") as text_file:
  text_file.write("{\"url\" : \"http://0.0.0.0:%d\"}" % port)
```

Procedure

1. As root, log into the host from which you run Metron.
2. Create a directory for the model.

   For this example, create a "sample" directory in the root user's home directory.
3. Now, you can create a simple shell script that will expose a REST endpoint called "echo" that will echo back the arguments passed to it. Create a file in the "sample" directory named "echo.sh", and copy the following information into the file.

   Note: In this simple REST service, we are always binding to port 1500. In a real REST service which would expose your model, you will use an appropriate port.

```
#!/bin/bash
rm -f out
mkfifo out
trap "rm -f out" EXIT
echo "{ \"url\" : \"http://localhost:1500\", \"functions\" : { \"apply\" :
 \"echo\" } }" > endpoint.dat
while true
do
  cat out | nc -l 0.0.0.0 1500 > >( # parse the netcat output, to build
 the answer redirected to the pipe "out".
    export REQUEST=
    while read line
    do
      line=$(echo "$line" | tr -d '[\r\n]')

      if echo "$line" | grep -qE '^GET /' # if line starts with "GET /"
      then
        REQUEST=$(echo "$line" | cut -d ' ' -f2) # extract the request
      elif [ "x$line" = x ] # empty line / end of request
      then
        HTTP_200="HTTP/1.1 200 OK"
        HTTP_LOCATION="Location:"
        HTTP_404="HTTP/1.1 404 Not Found"
        # call a script here
        # Note: REQUEST is exported, so the script can parse it (to answer
 200/403/404 status code + content)
        if echo $REQUEST | grep -qE '^/echo/'
        then
            printf "%s\n%s %s\n\n%s\n" "$HTTP_200" "$HTTP_LOCATION"
 $REQUEST ${REQUEST#"/echo/"} > out
        else
```

```
            printf "%s\n%s %s\n\n%s\n" "$HTTP_404" "$HTTP_LOCATION"
 $REQUEST "Resource $REQUEST NOT FOUND!" > out
        fi
      fi
    done
  )
done
```

4. Change directories to $METRON_HOME.

```
cd $METRON_HOME
```

5. Start the MaaS service in bin/maas_service.sh -zk $ZOOKEEPER_HOST:2181.

```
bash bin/maas_service.sh -zk $ZOOKEEPER_HOST:2181
```

where

-c, --create	Flag to indicate whether to create the domain specified with -domain.
-d,--domain <arg>	ID of the time line domain where the time line entities will be put
-e,--shell_env <arg>	Environment for shell script. Specified as env_key=env_val pairs.
-h,--help	The help screen
-j,--jar <arg>	Jar file containing the application master
-l,--log4j <arg>	The log4j properties file to load
-ma,--modify_acls <arg>	Users and groups that allowed to modify the time line entities in the given domain
-ma,--master_vcores <arg>	Amount of virtual cores to be requested to run the application master
-mm,--master_memory	Amount of memory in MB to be requested to run the application master
-nle,--node_label_expression <arg>	Node label expression to determine the nodes where all the containers of this application will be allocated, "" means containers can be allocated anywhere, if you don't specify the option, default node_label_expression of queue will be used.
-q,--queue <arg>	RM Queue in which this application is to be submitted
-t,--timeout <arg>	Application timeout in milliseconds
-va,--view_acls <arg>	Users and groups that allowed to view the time line entities in the given domain
-zq,--zk_quorum <arg>	ZooKeeper Quorum

-zr,--zk_root <arg>	ZooKeeper Root

6. Test the configuration to ensure that the MaaS service is running correctly.

   For example, you can enter the following:

   a) Start one instance of a sample echo service (named 'sample' version '1.0') in a container of 500m:

   ```
   bin/maas_deploy.sh -hmp ~/sample -hmp /user/root/maas/sample -m 500 -mo
    ADD -n sample -ni 1 -v 1.0 -zk $ZOOKEEPER_HOST:2181 -lmp /root/sample
   ```

   b) Wait a couple seconds and then ensure that the service started by running the following command:

   ```
   curl -i http://localhost:1500/echo/foobar
   ```

   You should see a response foobar.

   c) List the active models and ensure that you see the sample model in the output.

   ```
   bin/maas_deploy.sh -mo LIST -n sample -zk $ZOOKEEPER_HOST:2181
   ```

   d) Remove one instance of the sample model.

   ```
   bin/maas_deploy.sh -mo REMOVE -n sample -ni 1 -v 1.0 -zk
    $ZOOKEEPER_HOST:2181
   ```

   e) After a couple seconds ensure that you cannot access the sample model any longer:

   ```
   curl -i http://localhost:1500/echo/foobar
   ```

Deploy Models

After creating a model, you need to deploy the model onto HDFS and submit a request for one or more instances of the model.

Procedure

1. Create a simple sample python model.

   Let's say that you have a model, exposed as a REST microservice called "mock_dga" that takes as an input argument "host" which represents an internet domain name and returns a field called "is_malicious" which is either "malicious" if the domain is thought to be malicious or "legit" if the domain is not thought to be malicious. The following is a very simple example service that thinks that the only legitimate domains are "yahoo.com" and "amazon.com":

   ```
   from flask import Flask
   from flask import request,jsonify
   import socket
   app = Flask(__name__)

   @app.route("/apply", methods=['GET'])
   def predict():
           h = request.args.get('host')
           r = {}
           if h == 'yahoo.com' or h == 'amazon.com':
                   r['is_malicious'] = 'legit'
           else:
                   r['is_malicious'] = 'malicious'
           return jsonify(r)

   if __name__ == "__main__":
           sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   ```

```
        sock.bind(('localhost', 0))
        port = sock.getsockname()[1]
        sock.close()
        with open("endpoint.dat", "w") as text_file:
                text_file.write("{\"url\" : \"http://0.0.0.0:%d\"}" %
 port)
        app.run(threaded=True, host="0.0.0.0", port=port)
```

2. Store this python model in a directory called /root/mock_dga as dga.py and an accompanying shell script called rest.sh which starts the model:

```
#!/bin/bash
python dga.py
```

3. If you have not already done so, start MaaS:

```
$METRON_HOME/bin/maas_service.sh -zk $ZOOKEEPER_HOST:2181
```

4. Start one or more instances of the model, calling it "dga" and assigning an amount of memory to each instance:

   Because you have placed the model in the /root/mock_dga directory, enter the following:

```
$METRON_HOME/bin/maas_deploy.sh -zk $ZOOKEEPER_HOST:2181 -lmp /root/
mock_dga -hmp /user/root/models -mo ADD -m 512 -n dga -v 1.0 -ni 1
```

   where

-h, --h	A list of functions for maas_deploy.sh
-hmp, --hdfs_model_path <arg>	Model path (HDFS)
-lmp, --local_model_path <arg>	Model path (local)
-m, --memory <arg>	Memory for container
-mo, --mode <arg>	ADD, LIST, or REMOVE
-n, --name <arg>	Model name
-ni, --num_instances <arg>	Number of model instances
-v, --version <arg>	Model version
-zq, --zk_quorum <arg>	ZooKeeper quorum
-zr, --zk_root <arg>	ZooKeeper root

Add the MaaS Stellar Function to the Sensor Configuration

After deploying a model, you need to add the Stellar function for MaaS to the configuration file for the sensor on which you want to run the model.

Procedure

1. Edit the sensor configuration at $METRON_HOME/config/zookeeper/parsers/$PARSER.json to include a new FieldTransformation to indicate a threat alert based on the model.

```
{
  "parserClassName": "org.apache.metron.parsers.GrokParser",
  "sensorTopic": "squid",
  "parserConfig": {
    "grokPath": "/patterns/squid",
    "patternLabel": "SQUID_DELIMITED",
    "timestampField": "timestamp"
  },
  "fieldTransformations" : [
    {
      "transformation" : "STELLAR"
    ,"output" : [ "full_hostname", "domain_without_subdomains",
 "is_malicious", "is_alert" ]
    ,"config" : {
      "full_hostname" : "URL_TO_HOST(url)"
      ,"domain_without_subdomains" :
 "DOMAIN_REMOVE_SUBDOMAINS(full_hostname)"
      ,"is_malicious" : "MAP_GET('is_malicious',
 MAAS_MODEL_APPLY(MAAS_GET_ENDPOINT('dga'), {'host' :
 domain_without_subdomains}))"
      ,"is_alert" : "if is_malicious == 'malicious' then 'true' else null"
                }
    }
                           ]
}
```

where

transformation

Enter 'STELLAR' to indicate this is a Stellar field transformation.

output

The information the transformation will output. This typically contains full_host, domain_without_subdomains, is_malicious, and is_alert.

full_hostname

The domain component of the "url" field.

domain_without_subdomains

The domain of the "url" field without subdomains.

is_malicious

The output of the "mock_dga" model as deployed earlier. In this case, it will be "malicious" or "legit", because those are the values that our model returns.

is_alert

Set to "true" if and only if the model indicates the hostname is malicious.

2. Edit the sensor enrichment configuration at $METRON_HOME/config/zookeeper/parsers/PARSER.json to adjust the threat triage level of risk based on the model output:

```
{
  "index": "$PARSER_NAME",
  "batchSize": 1,
  "enrichment" : {
    "fieldMap": {}
  },
  "threatIntel" : {
    "fieldMap":{},
    "triageConfig" : {
      "riskLevelRules" : {
        "is_malicious == 'malicious'" : 100
      },
      "aggregator" : "MAX"
    }
  }
}
```

3. Upload the new configurations to $METRON_HOME/bin/zk_load_configs.sh --mode PUSH -i $METRON_HOME/config/zookeeper -$ZOOKEEPER_HOST:2181.
4. If this is a new sensor and it does not have a Kafka topic associated with it, then we must create a new sensor topic in Kafka.

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper
 $ZOOKEEPER_HOST:2181 --create --topic $PARSER_NAME --partitions 1 --
replication-factor 1
```

Start Topologies and Send Data

The final step in setting up Model as a Service, is to start the topologies and send some data to test the model.

Procedure

1. Start the sensor upon which the Model as a Service will run:

```
$METRON_HOME/bin/start_parser_topology.sh -k -$KAFKA_HOST -z
 $ZOOKEEPER_HOST:2181 -s $PARSER_NAME
```

2. Generate some legitimate data and some malicious data on the sensor.

   For example:

```
#Legitimate example:
squidclient http://yahoo.com
#Malicious example:
squidclient http://cnn.com
```

3. Send the data to Kafka:

```
cat /var/log/squid/access.log | /usr/hdp/current/kafka-broker/bin/kafka-
console-producer.sh --broker-list $KAFKA_HOST:6667 --topic squid
```

4. Browse the data in Elasticsearch at http://$ELASTICSEARCH_HOST:9100/_plugin/head to verify that it contains the appropriate documents.

   For the current example, you would see the following:

   - One from yahoo.com which does not have is_alert set and does have is_malicious set to legit.
   - One from cnn.com which does have is_alert set to true, is_malicious set to malicious, and threat:triage:level set to 100.

Modify a Model

You can remove a number of instances of the model by executing maax_deploy.sh with remove as the -mo argument.

Procedure

1. For example, the following removes one instance of the dga model, version 1.0:

```
$METRON_HOME/bin/maas_deploy.sh -zk -$ZOOKEEPER_HOST:2181 -mo REMOVE -m
 512 -n dga -v 1.0 -ni 1
```

2. If you need to modify a model, you need to modify the model itself and deploy a new version, then remove the old version instances afterward.