Smart Cities Hackathon Qomodo IoT Threat Analysis Challenge

Qomodo IoT Threat Analysis Challenge

Summary

In this challenge, you will create a Node.js script to generate a selection of sensor packets that simulate a good and bad actor. The Qomodo threat analysis will then analyse the packets and report these within the Qomodo visualisation engine. The readings will be created using Node.JS and submitted over MQTT. You will then be able to use the Qomodo platform to analyse the results.

This challenge can be completed by creating a Docker image first or, if your Node.js is installed locally, you can run the script locally. In this challenge, you will learn how to install Docker, create a new image, and create a Node.JS file for submitting readings.

Step 1

Go to the official Docker website: https://www.Docker.com/products/Docker-desktop
Download Docker Desktop for your operating system (Windows, macOS, or Linux).

Follow the installation instructions provided for your operating system. Make sure to install Docker app repository - this contains the program engine required to run commands.

Open a terminal or command prompt and start up the Docker engine.
You can ask your AI Assistant how to do this for your operating system, as well as how to configure your system to start Docker on boot.

Step 2

Choose a location on your computer for the project files for example c:\code

Create a new folder for your challenge, for example with `mkdir Qomodo`.

Step 3

Create a **Dockerfile**

```
# Use the official Node.js image as the base image
FROM node:latest

# Set the working directory inside the container
WORKDIR /app

# Copy the local files into the container
COPY . .

# Install dependencies and start the application
RUN npm install

# Start the application
CMD ["node", "qomodo.js"]
```

Step 4

Create a file called **compose.yml**

```
services:
 node:
   build: .
   container_name: qomodo-challenge
   restart: always
```

Step 5

To run the scripts over HYYP we require a package for this so we next need to create a file called **package.json**

```
{
 "name": "qomodo",
 "version": "1.0.0",
 "description": "",
 "main": "qomodo.js",
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
   "mqtt": "^5.10.1",
   "chance": "^1.1.12"
 }
}
```

Step 6

Next, we need to create a new **qomodo.js** file. The first part of the script defines the constants for the packages used to create and transfer the readings through MQTT

```
const mqtt = require('mqtt');
const Chance = require('chance');
const chance = new Chance();
```

Step 7

Now, we need to provide the MQTT Broker URL and topics that the script will be using. If you want to learn more about these, ask Marvin for more details. Enter the password and username as printed on the password sheet provided.

```
// Configuration
const brokerUrl = 'mqtt://gw-0.qomodo.io:8883';
const topic = 'events';

// User Details
const options = {
    username: "REPLACE_WITH_USERNMAE",
    password: "REPLACE_WITH_PASSWORD",
    rejectUnauthorized: true
};
const teamNumber = 0;// REPLACE WITH YOUR_TEAM NUMBER
```

Step 8

Now, we need to define the bad actor observables, which will be generated by the script

```
// Required Observables
const observables = [
   { src_ip: '152.32.138.247', instances: 12 },
   { src_ip: '45.159.209.228', instances: 1, dst_ip: '152.32.138.247' },
   { src_ip: '155.138.146.162', instances: 1, dst_port: '5555', filename:
'kv-mipsel', file_hash:
'48299c2c568ce5f0d4f801b4aee0a6109b68613d2948ce4948334bbd7adc49eb' }
];
```

Step 9

Next, we define the constant for the random readings event

```
const generateRandomEvent = () => ({
   device_id: chance.guid(),
   event_id: chance.guid(),
   event_timestamp: new Date().toISOString(),
   event_type: 'NetworkConnection',
   src_ip: chance.ip(),
   src_port: chance.integer({ min: 1024, max: 65535 }).toString(),
   dst_ip: chance.ip(),
   dst_port: chance.integer({ min: 1024, max: 65535 }).toString(),
   src_bytes: chance.integer({ min: 1, max: 10000 }).toString(),
   dst_bytes: chance.integer({ min: 1, max: 10000 }).toString(),
   domain_name: chance.domain(),
   team_number:teamNumber

});
```

Step 10

Next, we define the constant for the observables events

```
// Generate specific observables
const generateObservableEvent = (observable, isContinuedHost = false) => ({
   device_id: isContinuedHost ? continuedHost : chance.guid(),
   event_id: chance.guid(),
   event_timestamp: new Date().toISOString(),
   event_type: 'NetworkConnection',
   src_ip: observable.src_ip,
   src_port: observable.src_port || chance.integer({ min: 1024, max: 65535
}).toString(),
   dst_ip: observable.dst_ip || chance.ip(),
   dst_port: observable.dst_port || chance.integer({ min: 1024, max: 65535
}).toString(),
   src_bytes: chance.integer({ min: 1, max: 10000 }).toString(),
   dst_bytes: chance.integer({ min: 1, max: 10000 }).toString(),
   domain_name: chance.domain(),
   filename: observable.filename,
   file_hash: observable.file_hash,
   team_number: teamNumber

});

let continuedHost = chance.guid(); // This will link events for the same
sensor
```

Step 11

Next, we need to define the logic to publish the events

```
// Function to publish events
const publishEvents = () => {
   const client = mqtt.connect(brokerUrl, options);

   client.on('connect', function () {
       console.log('Connected to MQTT broker');

       // Publish random events
       for (let i = 0; i < 100; i++) {
           let message;
           if (i < observables[0].instances) {
               message = generateObservableEvent(observables[0]);
           } else if (i === observables[0].instances) {
               message = generateObservableEvent(observables[1]);
           } else if (i === observables[0].instances + 1) {
               message = generateObservableEvent(observables[2], true);
           } else if (i === observables[0].instances + 2) {
               message = generateObservableEvent({
                   src_ip: '155.138.146.162',
                   filename: 'kv-mipsel',
                   file_hash:
'48299c2c568ce5f0d4f801b4aee0a6109b68613d2948ce4948334bbd7adc49eb'
               }, true);
```

```
            } else {
                message = generateRandomEvent();
            }

            const messageStr = JSON.stringify(message);
            client.publish(topic, messageStr, { qos: 1 }, function (err) {
                if (err) {
                    console.error('Failed to publish event:', err);
                } else {
                    console.log('Published event:', message);
                }
            });
        }

        // Disconnect the client after publishing
        setTimeout(() => client.end(), 5000);
    });

    client.on('error', function (err) {
        console.error('Connection error:', err);
    });
};
```

Step 12

Add the logic to submit the events and then do a series of new readings every 2 minutes.

```
// Publish events immediately
publishEvents();

// Set an interval to publish events every 2 minutes (120,000 milliseconds)
setInterval(publishEvents, 120000);
```

Step 13

Next, run the container by calling Docker compose up --build

You should see a response like this:

```
Attaching to hackathon-1
Published event: {
hackathon-1  |   device_id: 'b3b208d5-d18c-50bd-8db2-b0803983e382',
hackathon-1  |   event_id: 'eb33fb44-553b-5243-bab0-80d9c1217330',
hackathon-1  |   event_timestamp: '2024-09-13T08:04:48.103Z',
hackathon-1  |   event_type: 'NetworkConnection',
hackathon-1  |   src_ip: '111.179.17.123',
hackathon-1  |   src_port: '33638',
hackathon-1  |   dst_ip: '50.223.58.34',
hackathon-1  |   dst_port: '17188',
hackathon-1  |   src_bytes: '2811',
hackathon-1  |   dst_bytes: '4406',
hackathon-1  |   domain_name: 'esi.io'
hackathon-1  | }
hackathon-1  | Published event: {
hackathon-1  |   device_id: '282afb0b-4722-5eef-b0ca-8e1e104c92ad',
hackathon-1  |   event_id: '07dd4e77-2a56-5afe-a20f-2629182e0022',
hackathon-1  |   event_timestamp: '2024-09-13T08:04:48.103Z',
hackathon-1  |   event_type: 'NetworkConnection',
hackathon-1  |   src_ip: '112.151.248.189',
hackathon-1  |   src_port: '31815',
hackathon-1  |   dst_ip: '15.223.59.133',
hackathon-1  |   dst_port: '3429',
hackathon-1  |   src_bytes: '1502',
hackathon-1  |   dst_bytes: '6108',
hackathon-1  |   domain_name: 'egiiz.ug'
hackathon-1  | }
```

Step 14

Log into https://hackathon.qomodo.io and confirm you can see the readings being displayed.

The login details will be provided on your team password sheet. You should see the observable events displayed as alerts.

Step 15

There are frequent alerts that have occurred due to the traffic from the sensors you have set up. Let's investigate - what is the malicious IP address in question? What nation-state or country could be behind this attack on us? When a device is compromised, control is typically gained through a malicious payload. Search the platform for this payload and

confirm the device-ID. What vulnerability was used to exploit this device? We want a CVE number.
You have now completed this challenge.