Enterprise Scale Threat Hunting: C2 Beacon Detection with Unsupervised ML and KQL — Part 2

By Mehmet Ergene

Published: 2023-12-12 · Archived: 2026-04-05 23:45:57 UTC

Continuing with the same example, CS beacon with 15 minutes sleep and 25% jitter, we can calculate the below values for the beacon(these values can be calculated by analyzing the time deltas):
MinBeaconSleep = 675s
MaxBeaconSleep = 900s
AvgBeaconSleep = 787.5s
MinStdev(BeaconSleep) = 0s
MaxStdev(BeaconSleep)= MaxBeaconSleep — AvgBeaconSleep =112.5s

Using the information above, we can calculate the approximate/exact jitter ratio. How? Well, it's basic mathematics: we have an equation with several variables.

CalculatedJitter = (MaxStdev(BeaconSleep) / AvgBeaconSleep) * 100 = **14.2%**

The reason for the calculated jitter being smaller than the configured jitter is the behavior of the Cobalt Strike Beacon. The jitter in Cobalt Strike shifts the average beacon sleep to the left of the configured sleep value. If this were an Empire beacon, the calculated jitter would be 25%.

To make things more clear, we have some values that are configured in the beacon. On the other hand, we have the data that is a result of the configuration. What we are doing here is that, since we don't know the beacon configuration, we are analyzing the resulting data to verify it's from a beacon or not(kind of reverse engineering). Since we are trying to perform verification, we can define what kind of beacon configuration we are trying to verify.

Developing the KQL query

In order to detect beaconing, we can use firewall logs, proxy logs, process network connection logs, etc. The logs must have at least below information:

- Source Username/Source HostName
- Destination IP/Destination Hostname
- Destination Port
- Timestamp

We can use requestURL or URLHostname information for proxy logs if we want. Using Source IP information is not recommended because IP assignments during VPN connections are not cached in DHCP. Therefore, you can see one device with several different IPs assigned to it, which will break the detection logic.

Logic

```
For each Source-Destination-Port pair:
    sort the Timestamp ascending
    calculate time difference between each Timestamp
    calculate the stdev, avg, min, and max of the time deltas
    calcualte the jitter by using the stddev and avg time delta
    If jitter < [threshold]
        display the details/generate an alert
```

- Min and max time delta values can be used to increase the fidelity if there are too many results(beacon time delta must be between MinBeaconSleep and MaxBeaconSleep, but there might be some spikes).
- If there are more than X users/computers connecting to the same destination, it's more likely a nonmalicious beacon (for example, windows update)
- If the logs have only the IP address of the destination, the results can be enriched in several ways, like joining other events that have IP and hostname info.

With KQL, we can put TimeGenerated, SentBytes, and ReceivedBytes into lists using **make_set/make_list**. Then, we can sort the timestamp values by using **array_sort_asc** (Using **serialize** and **sort** doesn't work when the data is big). Since we have an array, we can use its length to apply some filtering:

Now, we have one row for each source-destination pair, and all the connection timestamps are in the array. This approach makes the data size small, and small data size means faster processing.

Next, we can use **mv-apply** on the array of timestamps, set_TimeGenerated, to run a subquery for each connection pair. This approach also improves the query performance. In the subquery, we can perform the first step of beacon analysis by using the **JitterThreshold** we just defined:

Note that we are storing the query result into the **BeaconCandidates** variable. We will perform further analysis on the stored results for fine-tuning.

Get Mehmet Ergene's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Now, we have all beacon candidates based on the thresholds defined. Next, we can filter out the beacons based on the **CompromisedDeviceCountMax** threshold. If there are more devices/users beaconing to the same destination, the beacons are most likely nonmalicious.

We put the results into a new variable, **PotentialBeacons,** for further analysis.

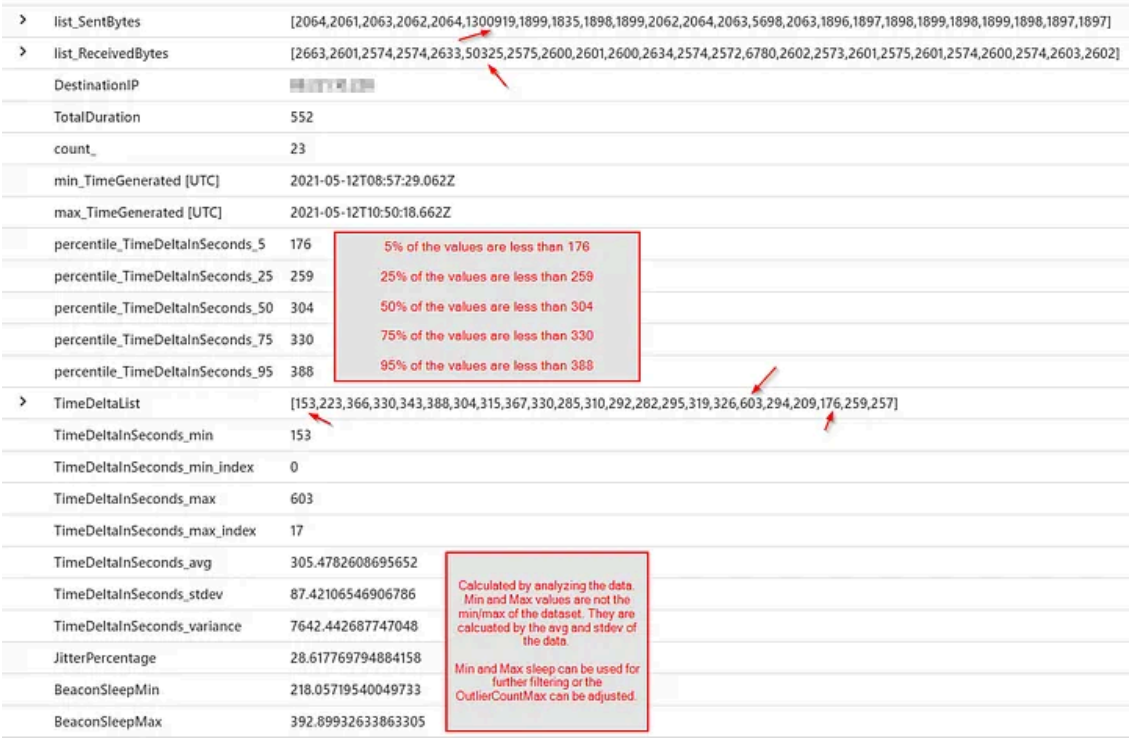
Next, we will find connections that can't be a beacon. We will use TimeDeltaList and list_SentBytes:

We used **series_outliers** to analyze the outliers. The function performs custom Tukey's analysis that accepts some outliers to exist in the data. In addition to that, if there are too many outliers, the connection can't be a beacon. Since the device can be turned off or not connected to a network for a while, time deltas can have spikes, but these spikes shouldn't happen a lot. That's why we put the **OutlierCountMax** condition; to accept more spikes.

As we now have the **PotentialBeacons**, **ImpossibleBeaconsByTimeDelta** and **ImpossibleBeaconsBySentBytes**, we can finally get all real beacons by removing the **ImpossibleBeaconsByTimeDelta** and **ImpossibleBeaconsBySentBytes** from **PotentialBeacons**:

Sample result(redacted) and how to read the data:

Press enter or click to view image in full size



I've developed queries for Palo Alto FW (Azure Sentinel), Sysmon (Azure Sentinel), and Microsoft Defender for Endpoint/Microsoft 365 Defender. You can find the queries in my GitHub repo. They run super-fast (90 million events are analyzed in 20 seconds) and are able to detect beacons with high jitter, like 90%.

How to use the queries

We first need to define boundaries for the beacons you want to detect. Defining the boundaries based on the Empire beacon behavior covers Cobalt Strike and others.

Hunting with the jitter only

In this scenario, we want to detect all beacons without filtering them based on the sleep interval. Just change the **JitterThreshold** and run the query.

Hunting with the jitter and sleep interval

In this scenario, we want to filter beacons based on the jitter and sleep interval thresholds.

Example: Beacons that have at least 15-minute(900s) sleep with %25 jitter

- **JitterThreshold = 25**
- **TimeDeltaThresholdMin** = 900 - (900*25/100) = 675 = 11 minutes, 15 seconds

Optionally, we want to set an upper boundary for the sleep interval:

- **TimeDeltaThresholdMax** = 900 + (900*25/100) = 1125= 18 minutes, 45 seconds

Based on these values, we can filter the results.

> **P.S.: If you want to learn KQL, especially for Microsoft Sentinel or Microsoft 365 Defender, do check out my [training website](). Hope to see you there!"**

Conclusion

Detecting C2 beacons is hard but not impossible(just requires some statistics knowledge). Beacons with high jitter configuration like 100% are harder to detect, but still possible if you have time to analyze the results.

You can automate the analysis of the results in several ways like using Logic Apps, enriching the data with VT score, using Jupyter Notebook, etc.

Although the series ends here, I'll cover a specific C2 scenario using the method I've explained. If you see a mistake or want to ask something about the method, send me a message on Twitter.

Happy hunting!
