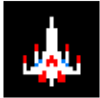
Detecting network beacons via KQL using simple spread stats functions

ateixei.medium.com/detecting-network-beacons-via-kql-using-simple-spread-stats-functions-c2f031b0736b

Alex Teixeira

April 30, 2021



Alex Teixeira

Apr 30, 2021

·

8 min read

What’s a network beacon? Why is that important? Well, let’s start with a quick definition before jumping into detection design and KQL code.

beacon

/ˈbiːk(ə)n/

noun

a fire or light set up in a high or prominent position as a warning, signal, or celebration.
"a chain of beacons carried the news"

- BRITISH
  a hill suitable for a beacon.
  "Ivinghoe Beacon"
- a light or other visible object serving as a signal, warning, or guide at sea, on an airfield, etc.

Similar: warning light/fire | signal light/fire | bonfire | smoke signal | beam | signal

As Google suggests, beacon is a visible object serving as a **signal** or **warning**. That’s what we, as detection engineers, are looking for when deploying a new detection or analytic rule.

In our context, *beacon* is referred to as traffic leaving the network at somewhat regular intervals with the purpose of communicating with a command-and-control server (C2).

This method can be used in a variety of ways: to ‘heartbeat’, to request new commands, or to download updates by interacting with the C2 server. It also works over any protocol, mostly leveraging outbound allowed network connections via the web proxy or firewall (HTTP/S, DNS, etc).

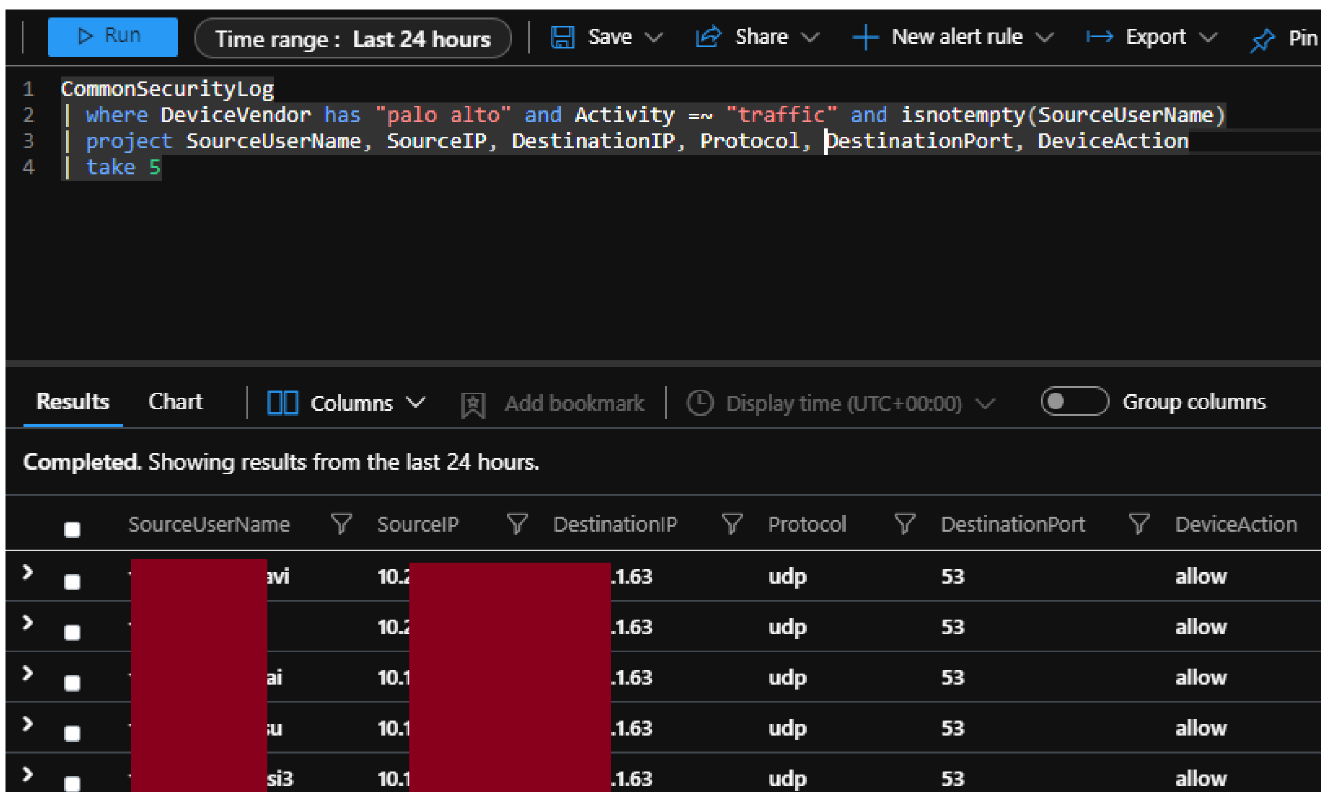
What data or log telemetry to use?

While there are many methods to tackle the challenge of detecting beaconing traffic, among the target data sources to use, web proxy, firewall and DNS log sources are figuring as top candidates.

Moreover, any data source providing *time series* telemetry including the *origin* and *target* of the network connection is also a potential candidate. Examples:

- Host FW such as Windows Firewall (/ eventlogs)
- EDR (some provide all external network connections)

In this example, we are going to pick **Palo Alto** events that are logged to the **CommonSecurityLog** table in Log Analytics/Azure Sentinel. The events look like the following (base query):



As it applies to many other NG FWs or UTMs, the user account is also logged and that makes a big difference here.

Regardless of the (dynamic) IP address assigned to an affected host, tracking the origin via the user account eases the process of doing Hostname lookups while also making it faster to track the affected user.

Crafting a fully parameterized query

KQL (Kusto Query Language) allows us to define constants and variables to be used throughout the code, just like a procedural programming language does.

That’s done via the statement:

```
1  // Excluded networks, all private/RFC 1918 are excluded via ipv4_is_private()
2  let ExcludedNets = datatable(ExcludedNet: string)
3      [
4      '200.1.0.0/16',    // My public DMZ 1
5      '200.2.0.0/16',    // My public DMZ 2
6      ];
7  // Query schedule parameters
8  let time_offset = 2h;           // how far back the needle should end?
9  let time_window = 8h;           // how much time to look back from there?
10 let end = ago(time_offset);     // latest (TimeGenerated)
11 let start = end - time_window;  // earliest (TimeGenerated)
12 // Beacon detection parameters
13 let threat_match_only = 0;             // set to 1 to alert only when there's also a match from ThreatIntelligenceIndicator
14 let bin_size = 1h;                     // what's the size of each window for collecting deltas of consecutive similar flows
15 let min_bins = 8;                      // how many distinct hours (within time_window) needed to consider it a consistent beacon flow
16 let min_events_per_bin = 60;           // how many events (minimum) needed within each bin (bin_size) to consider it a beacon
17 let max_delta_deviation = 15;          // delta = diff, in secs, between consecutive similar requests, this = max stdev(delta)
18 let max_victims = 2;                   // how many max distinct SourceUserName values should be seen linked to an external IP address
19 let max_receive_bytes_deviation = 1024; // max deviation on stdev(ReceivedBytes), assuming the bytes received should not change much
```

Click to view

Here’s where we define all the parameters, from the query’s time period, to beacon’s attributes, including which IP address ranges to ignore.

Some of those key parameters are explained further below and you can find all descriptions in the query as comments.

The Beacons table

This dynamic table holds a list of beacon candidates. The *CommonSecurityLog* table is used to store events from multiple products and vendors, therefore we need to narrow the query down to our scope.

In this case, we look for related events coming from vendor which also provides a valid value:

```
let Beacons = materialize(CommonSecurityLog| where TimeGenerated between
(start..end)| where DeviceVendor has "palo alto" and Activity =~ "traffic" and
isnotempty(SourceUserName)| where not(ipv4_is_private(DestinationIP))| evaluate
ipv4_lookup(ExcludedNets, DestinationIP, ExcludedNet, return_unmatched = true)| where
isempty(ExcludedNet)| project TimeGenerated, SourceUserName, DestinationIP,
DestinationPort, DeviceAction, ReceivedBytes, Protocol, SentBytes
```

After excluding unwanted traffic, we simply project the relevant fields, speeding up the process. Another best practice we are leveraging here is the use of special function, which enables results caching.

Calculating stats per beacon instance

First of all, there are probably many other ways to do that. Happy to discuss other ideas and how we could improve it.

I’ve tried using the operator which enables sub-query over split-by results but unfortunately given the high number of potential distinct tuples (cardinality), the max partitions limit (64) is easily reached.

In Splunk (SPL), many commands do support that approach, including *eventstats* and *streamstats*, which is what I leverage here.

In Kusto (KQL), there's a **super awesome** list of resources provided by Ashwin Patil (Senior Program Manager @Microsoft MSTIC) that covers an example of Network Beaconing detection but using a different, much simpler approach:

ashwin-patil/blue-teaming-with-kql

Repository with Sample KQL Query examples for Threat Hunting This folder has various KQL examples related to Threat…

github.com

The way to calculate those stats is basically by sorting the events by time and 'tuple' and then using Windows Functions to reference fields from previous records matching the current tuple at hand.

What's in a tuple?

In this context, it's basically what identifies a **distinct instance** of a beacon candidate. That's defined in this example by the following line:

```
| extend tuple = strcat(SourceUserName, '->', Protocol, ' ', DestinationIP, ':',
DestinationPort, ' (', DeviceAction, ')')
```

That means *tuple* holds both the origin (username) and target (destination IP + Port) related to a potential beacon. To make it even stricter, we also add the traffic outcome (allow/deny) to the tuple identifier.

The delta and variance

The key detection design here, besides scoping on the important fields and other enrichments (covered later) is on how to calculate the difference in time or *delta* between two consecutive requests matching the same tuple, and later measuring **how dispersed those values are**.

That's done by referencing the *TimeGenerated* from previous/current record and later calculating the standard deviation of those values split by tuple.

And here's how we've done it with function:

```
| extend TimeGenerated_prev=prev(TimeGenerated)| extend diff=iff(tuple ==
prev(tuple), datetime_diff('second', TimeGenerated, TimeGenerated_prev), 9999)|
summarize EventCount=count(), stdevif(diff, diff != 9999), stdev(ReceivedBytes),
arg_max(TimeGenerated, *)  by tuple, Bin=bin(TimeGenerated, bin_size)
```

Instead of standard deviation, we could have used *variance* but with the former, it’s easier to *describe* what happened, more below.

As you can see, the first parameter is consumed here: . That is defined earlier in the query via *let* statement and represents the size of each window for collecting deltas from consecutive similar records.

```
let bin_size = 1h;
```

The idea is of course to enable easier code customization.

The last bit on beacon candidates is to filter on relevant flows only, and that’s done by using the operator in combination with other parameters which behave as **thresholds**.

```
| where EventCount >= min_events_per_bin and stdevif_diff <= max_delta_deviation and
stdev_ReceivedBytes <= max_receive_bytes_deviation);
```

The most important here is the following:

```
let max_delta_deviation = 15; // delta = diff, in secs, between consecutive similar
requests, this = max stdev(delta)
```

Roughly speaking, setting that value to 15 and assuming the times are in seconds, we are selecting network flows in which delta values observed are within 15s from their mean.

That enables us to tackle some **jitter** componentsand other simple evasion techniques available from some C2 tools and attack frameworks.

The final part o the *candidates* table is shown below:

```
| sort by tuple asc, TimeGenerated asc
| extend TimeGenerated_prev=prev(TimeGenerated)
| extend diff=iff(tuple == prev(tuple), datetime_diff('second', TimeGenerated, TimeGenerated_prev), 9999)
| summarize EventCount=count(), stdevif(diff, diff != 9999), stdev(ReceivedBytes), StartTime=min(TimeGenerated), EndTime=arg_max(TimeGenerated, *)
  by tuple, Bin=bin(TimeGenerated, bin_size)
| where EventCount >= min_events_per_bin and stdevif_diff <= max_delta_deviation and stdev_ReceivedBytes <= max_receive_bytes_deviation);
```

Click to enlarge

Increasing Fidelity

Here’s a list of additional enrichments and behavioral checks done in order to increase detection efficacy, all made possible via ops:

- “Are there multiple accounts matching the same destination IP address?” If there are too many, that’s a Set this via ;
- “Is the destination IP matching a Threat Intel feed?” In this example, we leverage the table. In case you want to alert only when there are matches, set to 1;
- “Is the beaconing traffic consistent?” Despite suggesting to detect beacons within 1h, over how many hours is the traffic observed? Check the parameter for details on that important setting;

- In a common C2 scenario, data inflow (received bytes) tends to be fairly stable and consistent, check how is used.

The output

Below is an expanded record, including a *Description* of the alert:

Field	Value
StartTime [UTC]	2021-04-27T12:01:04Z
EndTime [UTC]	2021-04-27T12:59:52Z
stdevif_diff	3.6601955603355405
VictimCount	1
EventCount	99
stdev_ReceivedBytes	98.1319395050378
BinCount	8
Protocol	udp
DestinationIP	95.71
DestinationPort	27019
DeviceAction	allow
TotalEventCount	780
TotalBytesSent	1534
TotalBytesReceived	3102
Description	Between 2021-04-27T12:01:04.0000000Z and 2021-04-27T12:59:52.0000000Z, potential beaconing traffic matching [f c->udp 195.71:27019 (allow)] has generated 99 events in 01:00:00 with ~3.7s deviation among beacons and 780 total events observed in the entire window checked

tuple	DestinationIP	Description	StartTime [UTC]	EndTime [UTC]	stdevif_diff	VictimCount	EventCount	stdev_ReceivedBytes	BinCount	Protocol
\ ->tcp 3.8:80 (allow)	.43.8	Between 2021-04-27T12:00:06.0000000Z and 2021-04-27T12:59:...	4/27/2021, 12:00:06.000 ...	4/27/2021, 12:59:50.000 ...	4.007	1	300	105.175	8	tcp

The query (rule template)

The full KQL code or rule template is available below and can easily be adapted to any other data source — providing it contains the necessary fields.

You can also use this template to create a **baseline**, which enables detection of new beacons as they happen by comparing their signature (tuple) to previously tracked ones.

Please feel free to reach out to exchange ideas on how to use and improve this method and happy hunting!

```
// Excluded networks, RFC 1918 are excluded via ipv4_is_private()let ExcludedNets =
datatable(ExcludedNet: string)    [    '200.1.0.0/16',   // Pub range 1
'200.2.0.0/16'    // Pub range 2    ];// Query schedule parameterslet time_offset =
2h;           // how far back the needle should end?let time_window = 8h;
// how much time to look back from there?let end = ago(time_offset);     // latest
(TimeGenerated)let start = end - time_window;  // earliest (TimeGenerated)// Beacon
detection parameters  let threat_match_only = 0;              // set to 1 to alert
only when there's also a match from ThreatIntelligenceIndicatorlet bin_size = 1h;
// what's the size of each window for collecting deltas of consecutive similar
flowslet min_bins = 8;                       // how many distinct hours (within
time_window) needed to consider it a consistent beacon flowlet min_events_per_bin =
60;            // how many events (minimum) needed within each bin (bin_size) to
consider it a beaconlet max_delta_deviation = 15;          // delta = diff, in secs,
between consecutive similar requests, this = max stdev(delta)let max_victims = 2;
// how many max distinct SourceUserName values should be seen linked to an external
IP addresslet max_receive_bytes_deviation = 1024; // max deviation on
stdev(ReceivedBytes), assuming the bytes received should not change muchlet Beacons =
materialize(CommonSecurityLog    | where TimeGenerated between (start..end)    |
where DeviceVendor has "palo alto" and Activity =~ "traffic" and
isnotempty(SourceUserName)    | where not(ipv4_is_private(DestinationIP))    |
evaluate ipv4_lookup(ExcludedNets, DestinationIP, ExcludedNet, return_unmatched =
true)    | where isempty(ExcludedNet)    | project TimeGenerated, SourceUserName,
DestinationIP, DestinationPort, DeviceAction, ReceivedBytes, Protocol, SentBytes    |
extend tuple = strcat(SourceUserName, '->', Protocol, ' ', DestinationIP, ':',
DestinationPort, ' (', DeviceAction, ')')    | extend
TimeGenerated=bin(TimeGenerated, 1s) // this and next keep only one event per tuple -
in case the request is made in the same second    | distinct TimeGenerated,
SourceUserName, DestinationIP, DestinationPort, DeviceAction, ReceivedBytes,
Protocol, SentBytes, tuple    | sort by tuple asc, TimeGenerated asc    | extend
TimeGenerated_prev=prev(TimeGenerated)    | extend diff=iff(tuple == prev(tuple),
datetime_diff('second', TimeGenerated, TimeGenerated_prev), 9999)    | summarize
EventCount=count(), stdevif(diff, diff != 9999), stdev(ReceivedBytes),
StartTime=min(TimeGenerated), EndTime=arg_max(TimeGenerated, *) by tuple,
Bin=bin(TimeGenerated, bin_size)    | where EventCount >= min_events_per_bin and
stdevif_diff <= max_delta_deviation and stdev_ReceivedBytes <=
max_receive_bytes_deviation);let InfrequentDestinationIPs = Beacons     | summarize
VictimCount=dcount(SourceUserName) by DestinationIP    | where VictimCount <=
max_victims;let ConsistentBeacons = Beacons    | summarize BinCount=dcount(Bin) by
tuple    | where BinCount >= min_bins;Beacons| lookup kind=inner
(InfrequentDestinationIPs) on DestinationIP| lookup kind=inner (ConsistentBeacons) on
tuple| join kind=leftouter (ThreatIntelligenceIndicator    | where TimeGenerated >
ago(90d) and ConfidenceScore >= 75    | where isnull(KillChainReconnaissance) and
ConfidenceScore >= 30 and not(Description matches regex "(?i)(brute.*force)")    |
extend DestinationIP = coalesce(NetworkDestinationIP, NetworkIP, NetworkSourceIP)
| where DestinationIP matches regex '[0-9]'    | summarize
ConfidenceScore=max(ConfidenceScore), LastSeen=max(TimeGenerated) by DestinationIP,
ThreatDescription=Description, ThreatType    | sort by LastSeen desc, ConfidenceScore
desc)    on DestinationIP| where ConfidenceScore > 0 or 0 == threat_match_only|
summarize arg_min(StartTime, EndTime, stdevif_diff, VictimCount, EventCount,
stdev_ReceivedBytes, BinCount, Protocol, DestinationIP, DestinationPort,
DeviceAction, ThreatType, ConfidenceScore, ThreatDescription),
TotalEventCount=sum(EventCount), TotalBytesSent=sum(SentBytes),
TotalBytesReceived=sum(ReceivedBytes) by tuple| sort by stdevif_diff asc,
stdev_ReceivedBytes asc, EventCount desc| extend Description=strcat('Between ',
StartTime, ' and ', EndTime, ', potential beaconing traffic matching [', tuple, ']
```

```
has generated ', EventCount, ' events in ', bin_size, ' with ~',
round(stdevif_diff,1), 's deviation among beacons and ', TotalEventCount, ' total
events observed in the entire window checked.')
```