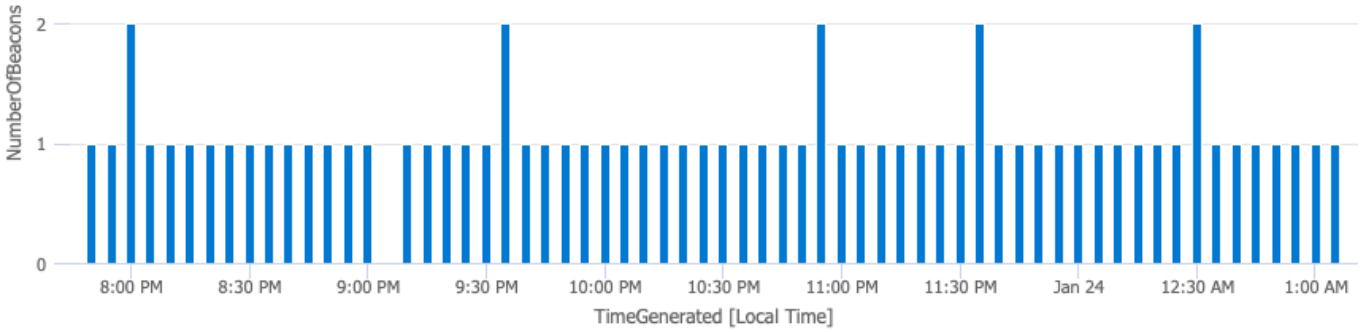
HUNTING MALWARE BEACONS

WITH JUPYTER NOTEBOOKS
AND
AZURE SENTINEL

BINARY DEFENSE™

THREAT HUNT HYPOTHESIS: REPEATING CONNECTIONS

Baza Backdoor connections over time:



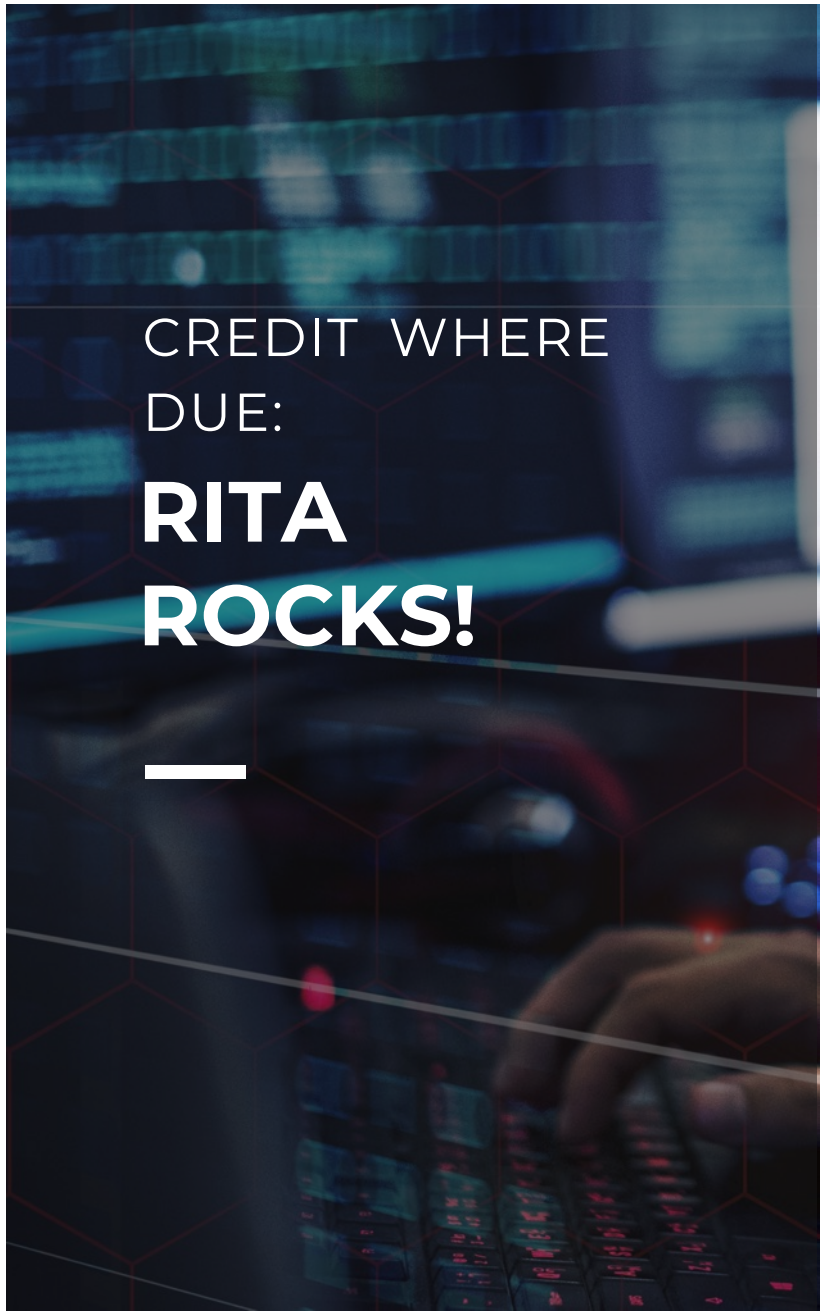
Connection pattern searches work even if:

- Remote IP is not suspicious (e.g. Google)
- Port is not unusual (e.g. TLS over 443)
- Malware is hiding in another process (injection)
- Traffic is encrypted

Need to use other methods if:

- C2 over DNS using expected DNS server
- Malware keeps a long-lived connection open
- Check-ins are infrequent or very irregular

BINARY DEFENSE™



REAL INTELLIGENCE THREAT ANALYTICS

RITA is an open source framework for network traffic analysis.

ACTIVE COUNTERMEASURES

zeek

https://www.activecountermeasures.com/free-tools/rita/

- Inspiration came from RITA, which uses Zeek logs
- Recommend checking out Zeek and RITA

But what if you don't have Zeek data?

INGREDIENT #1

All we need is the time and IP addresses of network traffic, but related processes would be nice!

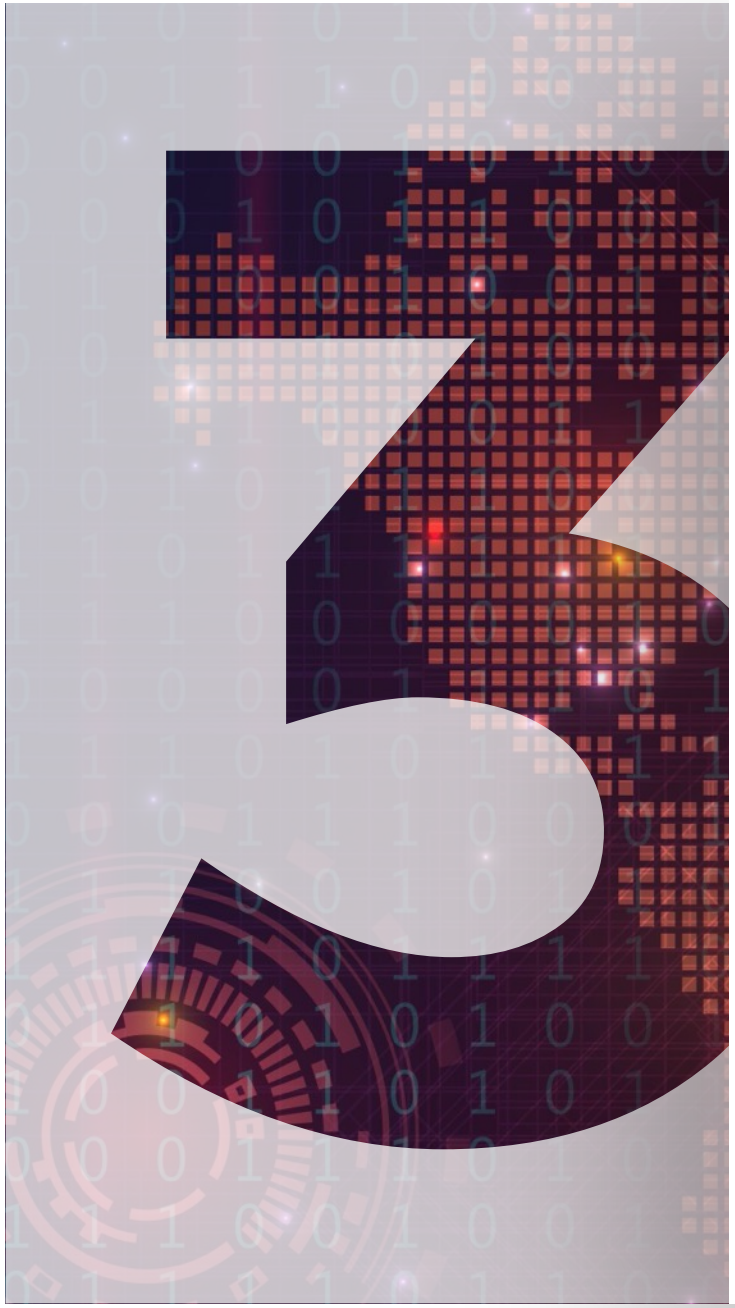
Several source data options:

- Microsoft Defender for Endpoint
  - DeviceNetworkEvents
- Sysmon
  - Event ID 3
- PacketBeat
  - Requires Logstash or Elasticsearch
- Suricata
  - Flow records (EVE Log)
  - Requires network visibility

INGREDIENT #2

We need to store the events in a database somewhere:

- **Splunk**
- **Elasticsearch**
- **Azure Sentinel**
- **MongoDB (RITA)**
- **Many other options**



INGREDIENT #3

We need a system to query the data, run some custom algorithms against it, and show the results visually, allowing exploration of the data

BINARY DEFENSE™

TOOLS USED:

Network Event Source:

- Option A: Microsoft Defender for Endpoint (MDE)
- Option B: Sysmon version 13.0.1

Storage & Query:

- Microsoft Azure Sentinel
- (Elasticsearch or Splunk would also work – we'll try them later)

Data Processing & Exploration:

- Jupyter-Hunt Server with MSTICPy modules and matplotlib

Malware Beacons Hunt.ipyr

Markdown

Python 3

BINARY DEFENSE

Threat Hunting Notebook: Malware Beacons by Network Pattern Analysis

This notebook connects to Microsoft Azure Sentinel, queries records of network connections from Sysmon (Event ID 3), or Microsoft Defender for Endpoint (DeviceNetworkEvents) and analyzes the pattern of time between network connections to find communication that looks like a regularly repeating beacon used for Command and Control (C2). These are likely to be either an endpoint agent checking in (which should be well known and documented) or malware checking in with its C2 server.

Step 1: Run the code block below to import all the Python modules we need to start. If there are any errors, stop and fix them now before going on.

[7]:
```
import sys, math,requests, json, datetime

MIN_REQ_PYTHON = (3,6)
if sys.version_info < MIN_REQ_PYTHON:
    print('Check the Kernel->Change Kernel menu and ensure that Python 3.6')
    print('or later is selected as the active kernel.')
    sys.exit("Python %s.%s or later is required.\n" % MIN_REQ_PYTHON)
```

BINARY DEFENSE

Malware Beacons Hunt.ipyr

Markdown

Python 3

Connect to Workspace for data

The next step is to connect this Notebook to your data source. All of the Workspace connection information should be set up in the msticpyconfig.yaml and config.json files, but you still need to authenticate with your user credentials that have read access to the Workspace you wish to connect to.

Unless you speecify otherwise, you'll be connected to the default workspace which contains data from Binary Defense testing clients used to experiment with attacks and detections.

Instructions:

1. Run the code block below
2. Wait for the KQL Magic output to appear
3. Click the button at the bottom of the output to copy the app code and open the authentication window
4. Authenticate with your user account, then continue running this Notebook

[3]:

```
## Create a QueryProvider object for running queries in our LogAnalytics workspace
qry_prov = QueryProvider(data_environment='LogAnalytics')
## Use the workspace configuration we've set up in msticpyconfig.yaml (you can als
#ws_config = WorkspaceConfig(workspace="Default")
ws_config = WorkspaceConfig(workspace="Workspace2")
qry_prov.connect(connection_str=ws_config.code_connect_str)
```

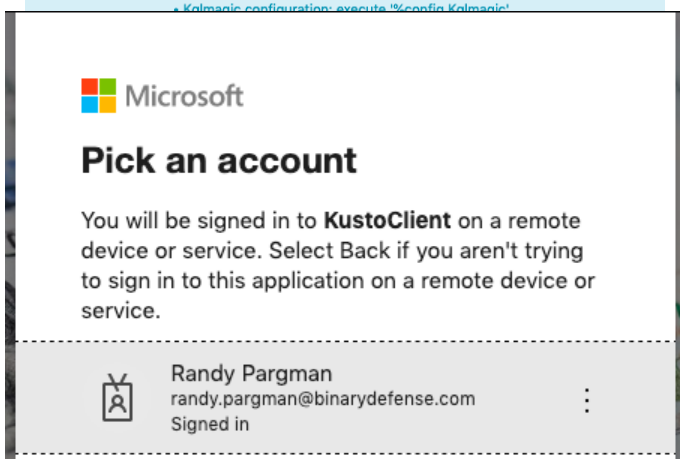
```
Please wait. Loading Kqlmagic extension...
```

kql

Kql Query Language, aka kql, is the query language for advanced analytics on Azure Monitor resources. The current supported data sources are Azure Data Explorer (Kusto), Log Analytics and Application Insights. To get more information execute '%kql --help "kql"'

• kql reference: Click on 'Help' tab > and Select 'kql reference' or execute '%kql --help "kql"'

• Kqlmagic configuration: execute '%config Kqlmagic'



Uses normal Microsoft account authentication, including MFA and PIM to authenticate to Azure Sentinel.

No API key or shared secret is required (whew!)

BINARY DEFENSE™

Gather Network Event Data

Before we can analyze network event data patterns, we need to get the raw information about timing of connection events. This notebook has KQL queries to get the data from Microsoft Defender for Endpoint if you have that, or Sysmon if you prefer to use that. All you really need from each event is the time, source IP and destination IP, so you could consume logs from Packetbeat or many other sources. If you have Zeek logs, you don't need this and should just use RITA to process those logs directly.

Query for Microsoft Defender for Endpoint

Run the code block below if you use Microsoft Defender for Endpoint

[4]:
```
# Microsoft Defender for Endpoint query that uses DeviceNetworkEvents
network_conn_query = '''DeviceNetworkEvents
| where TimeGenerated between (datetime(%s) .. datetime(%s))
| where ActionType !in ("InboundConnectionAccepted", "ListeningConnectionCreated")
| where isnotempty(RemoteIP)
| project TimeGenerated, LocalIP, RemoteIP
'''
process_query = '''DeviceNetworkEvents
| where TimeGenerated > ago(3d)
| where LocalIP=="%s"
| where RemoteIP=="%s"
| extend Hostname = tostring(split(DeviceName, ".")[0])
| summarize count() by Hostname,
UserName=InitiatingProcessAccountName,
FileName=InitiatingProcessFileName,
CommandLine=InitiatingProcessCommandLine,
ProcessId=InitiatingProcessId,
LocalIP, RemoteIP, RemotePort
'''
print("Defender for Endpoint query loaded.")
print("Do NOT run the next code block for Sysmon if you are using Defender for Endpoint. Skip ahead to next block.")
```

```
Defender for Endpoint query loaded.
Do NOT run the next code block for Sysmon if you are using Defender for Endpoint. Skip ahead to next block.
```

[5]:
```
## First get raw timing of network connections
## This is such a massive amount of data that we can only query it
## for about one hour's worth of results at a time
num_hours_to_search = 24
current_time = datetime.now(tz=pytz.utc)
connection_timing = {}
total_connections = 0

for hour in trange(num_hours_to_search):
    ##print("Querying network connections between %d and %d hours ago..." % (hour+1, hour))
    date1 = current_time - timedelta(hours=hour+1)
    date2 = current_time - timedelta(hours=hour)
    hour_query = network_conn_query % (date1, date2)
    #print(hour_query)
    df_hour_network_conns = qry_prov.exec_query(query=hour_query)
    for index, row in df_hour_network_conns.iterrows():
        key = tuple([row['LocalIP'], row['RemoteIP']])
        if not key in connection_timing:
            connection_timing[key] = []
        connection_timing[key].append(row['TimeGenerated']) # add connection time to list for this IP pair
        total_connections += 1

print("There were %d unique host pairs with %d total connections" % (len(connection_timing), total_connections))
```

```
100% |████████████████████████████████| 24/24 [19:16<00:00, 48.19s/it]
```

About 200k host pairs and 1.2 to 1.3 million connection events

Hunting for Network Beacons like RITA

This code is inspired by and adapted from the algorithm in the open source project RITA from Active Countermeasures: https://github.com/activecm/rita
The algorithm that analyzes network traffic to compute a beacon score can be found in this file: https://github.com/activecm/rita/blob/master/pkg/beacon/analyzer.go

This works by looking at each pair of communicating IP addresses and the timing of all the network connections between them. It computes the difference in time between each connection and then scores them based on the following factors:

- How regularly spaced apart are the connections?
- What much dispersion is there between median timing and outliers?
- How many connections are there over time?

Run the code block below to do all of the calculations for you. It ignores any pairs of hosts that had fewer than six total connections.

[24]:
```
beacon_scores = []
ip_pair_score_list = [] # for building a selection drop down later
current_time = datetime.now(tz=pytz.utc)
earliest_time = current_time
# compute the beacon score for each pair of communicating IPs
for ip_pair in connection_timing.keys():
    try:
        local_ip = ipaddress.ip_address(ip_pair[0])
        remote_ip = ipaddress.ip_address(ip_pair[1])
    except:
        continue
    if local_ip.is_private and remote_ip.is_private:
        continue
    if local_ip.is_loopback and remote_ip.is_loopback:
        continue
    timing_list = connection_timing[ip_pair]
```

Ignores private IP to private IP connections (but beware of port proxies!)

Top Potential Beacons by Overall Score

Now it is time to review the results. We can look at the data in different ways, but the most useful way to sort is by the beacon scores computed above, with the highest scores on top. Run the code block below to view the dataframe sorted by score. Adjust the number in the dataframe head() function below to get the top 10, top 25 or whatever number you want to review.

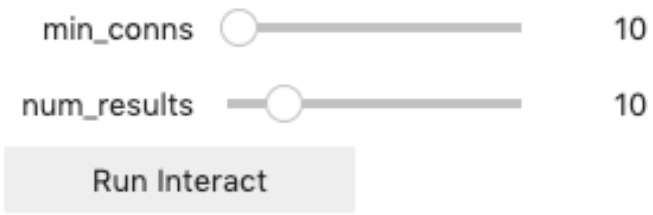
[25]:
```
beacons_df.set_index("score")
beacons_df.sort_values("score", ascending=False, inplace=True)

num_conns_slider = widgets.IntSlider(min=5, max=200, step=5, value=10)
num_results_slider = widgets.IntSlider(min=1, max=50, step=1, value=10)

def filter_beacons(min_conns=10,num_results=10):
    filtered = beacons_df[beacons_df['num_conns']>=min_conns].head(num_results)
    filtered.set_index("score")
    return filtered

widgets.interact_manual(filter_beacons,min_conns=num_conns_slider,num_results=num_results_slider)

# If you don't want the interactive sliders, comment out above and replace with two lines below:
#beacons_df.sort_values("score", ascending=False, inplace=True)
#beacons_df.head(25)
```



If there are a lot of results, it can be helpful to filter out the ones with few connections to help focus. Once every 10 minutes = ~144 in 24 hrs

min_conns 10

num_results 10

Run Interact

skew score: 1.0 - Bowley Number/Bowley Density (25/50/75 percentiles)
madm score: Median Absolute Dispersion about the Median
count score: Number of beacons >= expected from first conn to now

	score	ip_pair	skew_score	madm_score	count_score	beacon_interval_high	beacon_interval_low	earliest_conn	latest_conn	num_conns
151	0.984	(10[redacted]60, 72.[redacted]53)	0.990208	0.96170	1.000000	3602.2144	3596.9634	2021-01-23 07:23:35	2021-01-24 06:23:25	23
765	0.696	[redacted]2061:fe80::, [redacted]afca:b0...	0.989938	0.87285	0.225129	328.1731	314.5573	2021-01-23 13:10:56	2021-01-23 15:08:16	22
817	0.675	[redacted]0:961c::, [redacted]345a::)	0.956522	0.99960	0.068222	40.3454	40.2870	2021-01-23 09:20:09	2021-01-23 10:49:30	133
829	0.670	[redacted]20:8d1::, [redacted]29e1::)	1.000000	0.99980	0.007289	843.0170	9.9892	2021-01-23 06:56:33	2021-01-23 08:58:54	63
429	0.665	(10[redacted]7, 72.[redacted]240)	0.992855	0.00000	1.000000	14585.2690	33.5100	2021-01-23 09:00:15	2021-01-24 04:22:53	11
286	0.660	(10[redacted]246, 72.[redacted]240)	0.978596	0.00000	1.000000	11043.9411	3608.3541	2021-01-23 09:25:18	2021-01-24 06:39:53	12
270	0.660	(10[redacted]29, 72.[redacted]240)	0.999085	0.00000	0.979925	13236.6060	27.1882	2021-01-23 12:20:13	2021-01-24 06:38:14	13
311	0.658	(10[redacted]40, 72.[redacted]240)	0.972312	0.00000	1.000000	9395.9012	115.5832	2021-01-23 07:48:39	2021-01-24 04:57:12	17
421	0.657	(10[redacted]8, 205.[redacted]42)	0.970307	0.00000	1.000000	13975.1920	3.7830	2021-01-23 13:44:53	2021-01-24 03:56:27	11
170	0.656	(10.[redacted]66, 204[redacted]9)	0.989241	0.94520	0.031394	8.5808	1.7526	2021-01-24 06:36:06	2021-01-24 06:37:07	13

BINARY DEFENSE™

Let's get Visual

> "Those who do not learn from histograms are destined to keep repeating their analyses"

Select an IP pair from the list below and click the "Run interact" button to generate a histogram of the connection times. The graph shows how many connections were detected in each 10 minute bucket. So, if connections happened once a minute, you should see vertical bars closely spaced, all at the 10 count level.

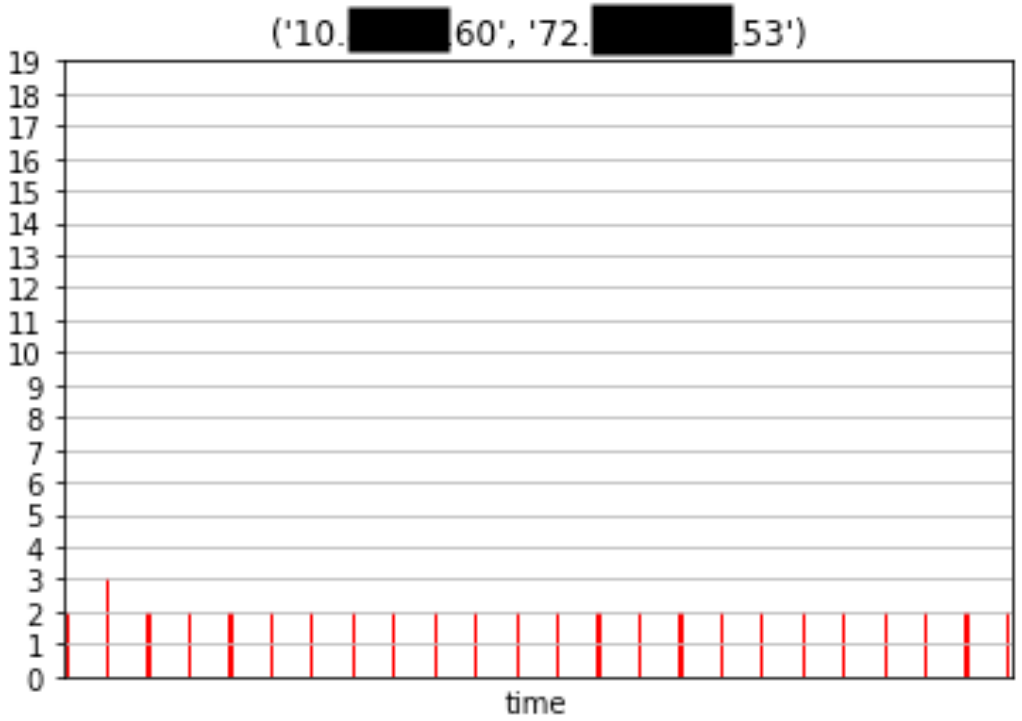
[32]:
```
current_ip_pair = None
def viewplot(ip_info):
    # "linking function with output"
    score, ip_pair = ip_info.split(":", 1)
    global current_ip_pair
    current_ip_pair = tuple(ip_pair.split("-", 1))
    plot = pd.DataFrame(connection_timing[current_ip_pair],  columns=['time'])
    # Setting the date as the index since the Grouper works on Index,
    # the date column is not dropped to be able to count
    plot.set_index('time', drop=False, inplace=True)
    # Get the histogram
    mid_time = earliest_time + (current_time-earliest_time)/2
    plot.groupby(pd.Grouper(freq='10Min')).count().plot(kind='bar',
                                                        xticks=(),
                                                       yticks=range(0,20),
                                                       grid=True, legend=False,
                                                       title=str(current_ip_pair),
                                                       color='red',
                                                       xlim=(earliest_time, current_time))
    return plot

# Create an interactive selector
ip_select =  widgets.Select(options=ip_pair_score_list[:num_results_slider.value+1],
                            width='400px', height='800px')
widgets.interact_manual(viewplot, ip_info=ip_select)
```

ip_info

0.98:10.[redacted]60-72.[redacted]53
0.70:b5f[redacted]8:9b[redacted]0
0.68:0:ff[redacted]:961c::-0:ff[redacted]
0.67:10.[redacted]-72.[redacted].240
0.67:10.[redacted]-72.[redacted].240

Run Interact



	time
2021-01-23 07:23:35.120000+00:00	2021-01-23 07:23:35.120000+00:00
2021-01-23 07:23:35.331000+00:00	2021-01-23 07:23:35.331000+00:00
2021-01-23 08:24:06.132000+00:00	2021-01-23 08:24:06.132000+00:00
2021-01-23 08:24:06.391000+00:00	2021-01-23 08:24:06.391000+00:00
2021-01-23 08:24:06.603000+00:00	2021-01-23 08:24:06.603000+00:00
2021-01-23 09:23:35.916000+00:00	2021-01-23 09:23:35.916000+00:00
2021-01-23 09:23:36.132000+00:00	2021-01-23 09:23:36.132000+00:00
2021-01-23 10:23:28.172000+00:00	2021-01-23 10:23:28.172000+00:00
2021-01-23 10:23:28.392000+00:00	2021-01-23 10:23:28.392000+00:00
2021-01-23 11:23:27.797000+00:00	2021-01-23 11:23:27.797000+00:00
2021-01-23 11:23:28.001000+00:00	2021-01-23 11:23:28.001000+00:00
2021-01-23 12:23:27.913000+00:00	2021-01-23 12:23:27.913000+00:00
2021-01-23 12:23:28.129000+00:00	2021-01-23 12:23:28.129000+00:00
2021-01-23 13:23:27.735000+00:00	2021-01-23 13:23:27.735000+00:00
2021-01-23 13:23:27.961000+00:00	2021-01-23 13:23:27.961000+00:00
2021-01-23 14:23:28.038000+00:00	2021-01-23 14:23:28.038000+00:00
2021-01-23 14:23:28.258000+00:00	2021-01-23 14:23:28.258000+00:00

Investigate Suspicious Processes

Now that you have suspicious connections to check into, we need to see which processes were responsible for those network connections.

Run the code block below to view the results for the IP pair that you selected for the histogram view above.

Note that these results are for processes from the last three days that had communications between the two IP addresses

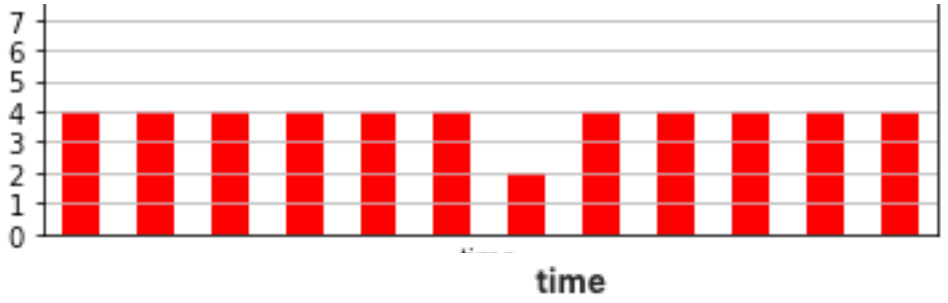
[40]:
```
ip_process_query = process_query % (current_ip_pair[0], current_ip_pair[1])
df_matching_processes = qry_prov.exec_query(query=ip_process_query)
df_matching_processes
```

[40]:

	Hostname	UserName	FileName	CommandLine	ProcessId	LocalIP	RemoteIP	RemotePort	count_
0	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	3716	10.[redacted].160	72.[redacted].53	47142	1
1	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	3716	10.[redacted].160	72.[redacted].53	57867	1
2	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	5516	10.[redacted].160	72.[redacted].53	16254	1
3	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	5516	10.[redacted].160	72.[redacted].53	32200	1
4	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	296	10.[redacted].160	72.[redacted].53	35740	1
...	...	...	...	...	...	...	...	...	...
141	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	5888	10.[redacted].160	72.[redacted].53	50984	1
142	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	2992	10.[redacted].160	72.[redacted].53	37749	1
143	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	2992	10.[redacted].160	72.[redacted].53	36215	1
144	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	2892	10.[redacted].160	72.[redacted].53	50810	1
145	hq-p[redacted]prd	svcsqlserveragent	DTExec.exe	FILE "\"\\H[redacted]xIntegrati...	2892	10.[redacted].160	72.[redacted].53	63048	1

146 rows × 9 columns

Score: 0.668, skew: 1.0, madm:0.99, count: 0.003

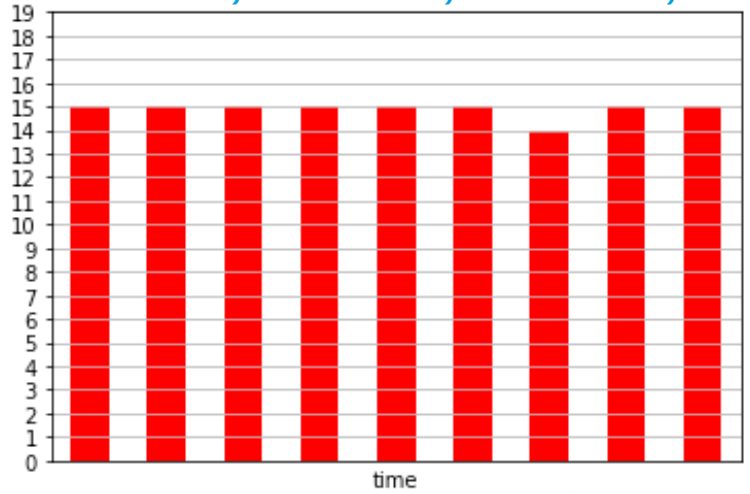


2021-01-23 13:10:56.767000+00:00	2021-01-23 13:10:56.
2021-01-23 13:10:56.767000+00:00	2021-01-23 13:10:56.
2021-01-23 13:16:11.286000+00:00	2021-01-23 13:16:11.
2021-01-23 13:16:11.286000+00:00	2021-01-23 13:16:11.
2021-01-23 13:21:27.107000+00:00	2021-01-23 13:21:27
2021-01-23 13:21:27.107000+00:00	2021-01-23 13:21:27
2021-01-23 13:26:44.578000+00:00	2021-01-23 13:26:44.
2021-01-23 13:26:44.578000+00:00	2021-01-23 13:26:44.

... (skipping to the end)

2021-01-23 14:57:24.865000+00:00	2021-01-23 14:57:24.8
2021-01-23 15:02:53.115000+00:00	2021-01-23 15:02:53.
2021-01-23 15:02:53.115000+00:00	2021-01-23 15:02:53.
2021-01-23 15:08:16.645000+00:00	2021-01-23 15:08:16.6
2021-01-23 15:08:16.645000+00:00	2021-01-23 15:08:16.6

Score: 0.662, skew: 0.96, madm:0.99, count: 0.028

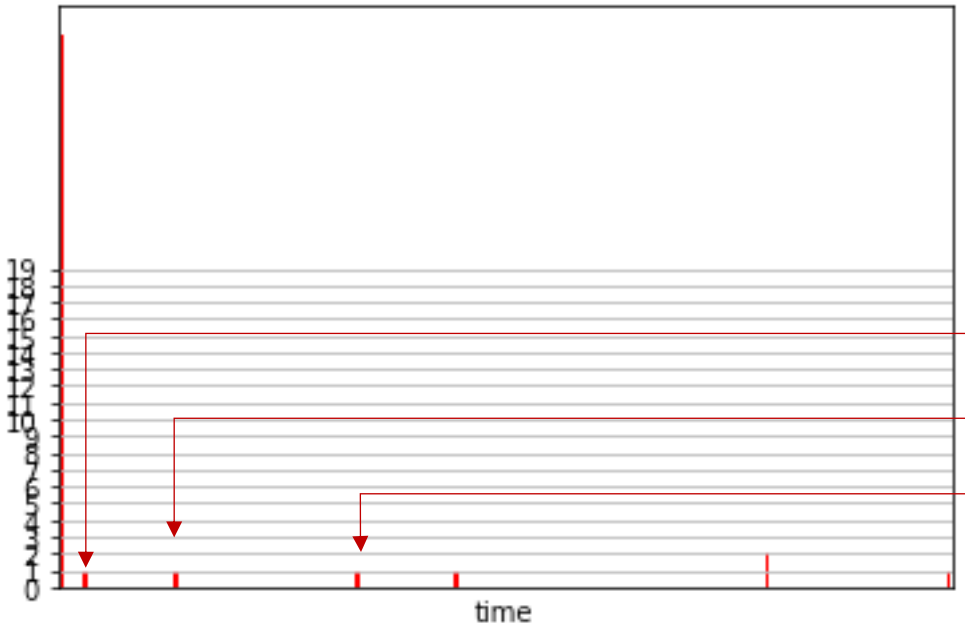


2021-01-23 09:20:09.237000+00:00	2021-01-23 09:20:09.
2021-01-23 09:20:49.565000+00:00	2021-01-23 09:20:49.5
2021-01-23 09:21:29.886000+00:00	2021-01-23 09:21:29.8
2021-01-23 09:22:10.199000+00:00	2021-01-23 09:22:10.
2021-01-23 09:22:50.503000+00:00	2021-01-23 09:22:50.5
...	
2021-01-23 10:46:49.640000+00:00	2021-01-23 10:46:49.6
2021-01-23 10:47:29.974000+00:00	2021-01-23 10:47:29.
2021-01-23 10:48:10.289000+00:00	2021-01-23 10:48:10.2
2021-01-23 10:48:50.598000+00:00	2021-01-23 10:48:50.5
2021-01-23 10:49:30.897000+00:00	2021-01-23 10:49:30.

134 rows × 1 columns

BINARY DEFENSE™

Score: 0.665, skew: 0.99, madm:0, count: 1.0



2021-01-23 08:51:24.323000+00:00	2021-01-23 08:51:24.323000
2021-01-23 08:51:24.356000+00:00	2021-01-23 08:51:24.356000
2021-01-23 08:51:24.397000+00:00	2021-01-23 08:51:24.397000
2021-01-23 08:51:24.430000+00:00	2021-01-23 08:51:24.430000
2021-01-23 08:51:24.464000+00:00	2021-01-23 08:51:24.464000
2021-01-23 08:51:24.497000+00:00	2021-01-23 08:51:24.497000
2021-01-23 08:51:24.531000+00:00	2021-01-23 08:51:24.531000
2021-01-23 08:51:24.564000+00:00	2021-01-23 08:51:24.564000
2021-01-23 08:51:24.599000+00:00	2021-01-23 08:51:24.599000
2021-01-23 08:51:24.640000+00:00	2021-01-23 08:51:24.640000

All of these are the big spike at the start

... (skipping to the end)

2021-01-23 08:51:27.200000+00:00	2021-01-23 08:51:27.200000
2021-01-23 08:51:27.240000+00:00	2021-01-23 08:51:27.240000
2021-01-23 08:51:27.273000+00:00	2021-01-23 08:51:27.273000
2021-01-23 08:51:27.309000+00:00	2021-01-23 08:51:27.309000
2021-01-23 09:24:48.796000+00:00	2021-01-23 09:24:48.796000
2021-01-23 11:25:32.471000+00:00	2021-01-23 11:25:32.471000
2021-01-23 15:28:12.806000+00:00	2021-01-23 15:28:12.806000
2021-01-23 17:30:41.769000+00:00	2021-01-23 17:30:41.769000
2021-01-24 00:25:34.233000+00:00	2021-01-24 00:25:34.233000
2021-01-24 00:25:34.270000+00:00	2021-01-24 00:25:34.270000
2021-01-24 04:27:31.531000+00:00	2021-01-24 04:27:31.531000

BINARY DEFENSE

INGREDIENT #4: MALWARE!

We have to have something to hunt! Let's try:

1. BazaLoader -> BazaBackdoor
2. Cobalt Strike Beacon
3. Custom VBS RAT with Google Scripts C2

BINARY DEFENSE

Query for Sysmon

If you use Sysmon to collect Event ID 3 (Network Connections) then uncomment the code block below and run it:

```
# Sysmon Event ID 3 -- if you already have a custom function
# to parse Sysmon events, you may simplify this query by using
# that. This notebook makes no assumptions or requirements for
# custom queries that you might have set up and parses the raw event.
# If you're looking for a good Sysmon parser custom function, see this:
# https://github.com/Azure/Azure-Sentinel/tree/master/Parsers/Sysmon
# or this:
# https://github.com/BlueTeamLabs/sentinel-attack/tree/master/parser
network_conn_query = '''Event
| where Source == "Microsoft-Windows-Sysmon"
| where EventID == 3
| where TimeGenerated between (datetime(%s) .. datetime(%s))
| extend EvData = parse_xml(EventData)
| extend EventDetail = EvData.DataItem.EventData.Data
| project-away EventData, EvData
| extend NetworkConnectionInitiated = tobool(EventDetail.[7].["#text"])
| extend LocalIP = tostring(EventDetail.[9].["#text"])
| extend RemoteIP = tostring(EventDetail.[14].["#text"])
| where isnotempty(RemoteIP)
| where NetworkConnectionInitiated == true
| project TimeGenerated, LocalIP, RemoteIP
'''
process_query = '''
Event
```

MALWARE BEACONS IN SMALL TEST ENVIRONMENT

min_conns 10

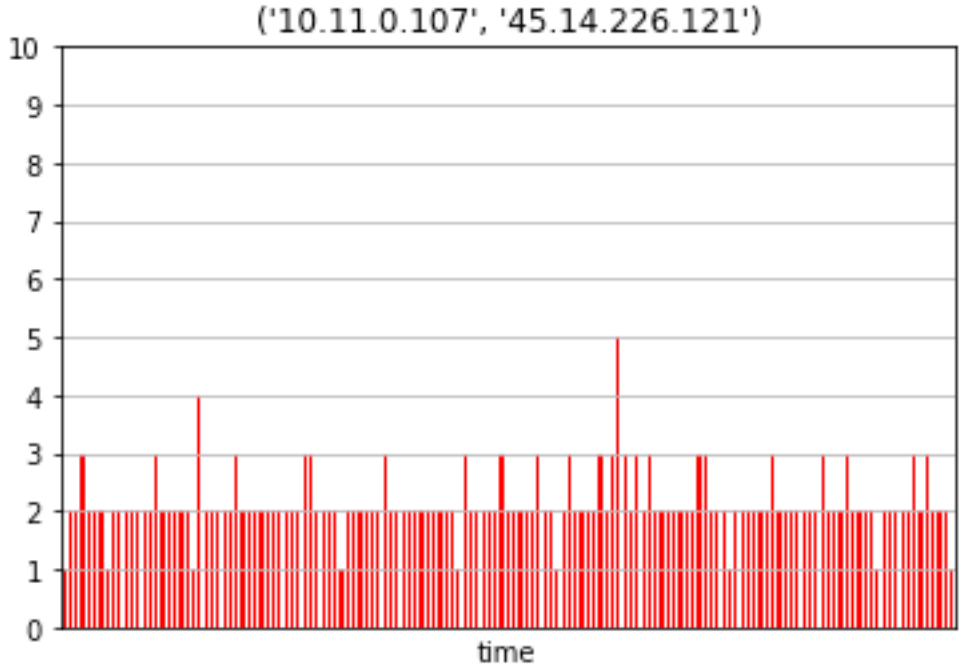
num_results 10

Run Interact

	score	ip_pair	skew_score	madm_score	count_score	beacon_interval_high	beacon_interval_low	earliest_conn	latest_conn	num_conns
2	0.982	(10.11.0.107, 45.14.226.121)	0.994547	0.951100	1.000000	311.2516	101.9286	2021-01-22 18:49:05	2021-01-23 18:42:27	307
0	0.981	(10.10.0.2, 172.217.2.110)	0.948571	0.994300	1.000000	61.3040	60.4400	2021-01-22 21:41:55	2021-01-23 18:46:02	1246
1	0.805	(10.10.0.2, 172.217.7.161)	0.865598	0.993000	0.556134	61.4327	60.3143	2021-01-22 21:41:56	2021-01-23 18:46:02	692
5	0.782	(10.10.0.2, 172.217.7.129)	0.964174	0.994650	0.385421	61.3763	60.4852	2021-01-22 23:04:07	2021-01-23 15:34:18	448
4	0.547	(10.10.0.2, 172.217.12.225)	0.564692	0.992233	0.082582	61.8090	60.3183	2021-01-23 02:10:47	2021-01-23 15:46:28	80
3	0.488	(10.10.0.2, 52.113.194.132)	0.773419	0.000000	0.688671	2130.2350	8.7700	2021-01-22 19:10:04	2021-01-23 18:41:01	112

BAZA BACKDOOR

Score: 0.98, skew: 0.99, madm:0.95, count: 1.0



	time
2021-01-22 18:07:44.790000+00:00	2021-01-22 18:07:44.790000+00:00
2021-01-22 18:12:52.530000+00:00	2021-01-22 18:12:52.530000+00:00
2021-01-22 18:18:01.293000+00:00	2021-01-22 18:18:01.293000+00:00
2021-01-22 18:23:14.913000+00:00	2021-01-22 18:23:14.913000+00:00
2021-01-22 18:28:22.943000+00:00	2021-01-22 18:28:22.943000+00:00
...	...
2021-01-23 17:43:56.967000+00:00	2021-01-23 17:43:56.967000+00:00
2021-01-23 17:49:05.573000+00:00	2021-01-23 17:49:05.573000+00:00
2021-01-23 17:54:13.993000+00:00	2021-01-23 17:54:13.993000+00:00
2021-01-23 17:59:23.780000+00:00	2021-01-23 17:59:23.780000+00:00
2021-01-23 18:04:31.920000+00:00	2021-01-23 18:04:31.920000+00:00

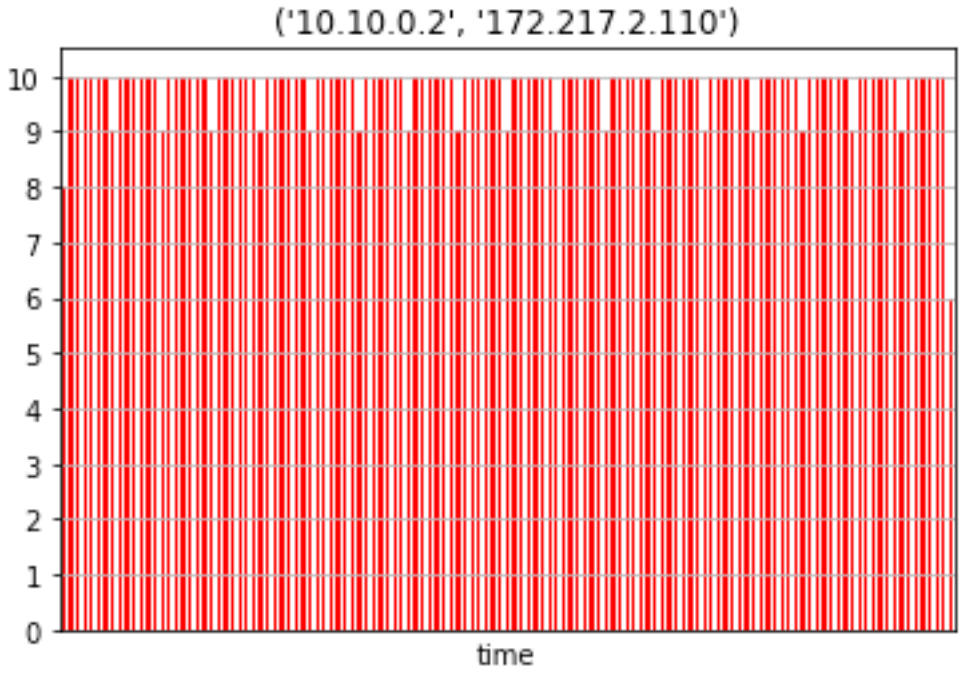
308 rows × 1 columns

- **Malware check-in approx. every 5 minutes**
- **Continuous connections all day and night**
- **All from one process: cmd.exe, one PID**
- **Hypothesis: process tampering**

	Hostname	UserName	ProcessPath	FileName	ProcessId	LocalIP	RemoteIP	RemotePort	count_
0	Work-0011-DM	ACCOUNTSPAYABLE\dmichaels	C:\Windows\System32\cmd.exe	cmd.exe	5000	10.11.0.107	45.14.226.121	443	789

GOOGLE SCRIPT VBS BACKDOOR

Score: 0.98, skew: 0.95, madm:0.99, count: 1.0



time	
2021-01-22 21:41:55.357000+00:00	2021-01-22 21:41:55.357000+00:00
2021-01-22 21:42:56.670000+00:00	2021-01-22 21:42:56.670000+00:00
2021-01-22 21:43:57.563000+00:00	2021-01-22 21:43:57.563000+00:00
2021-01-22 21:44:58.890000+00:00	2021-01-22 21:44:58.890000+00:00
2021-01-22 21:45:59.773000+00:00	2021-01-22 21:45:59.773000+00:00
...	...
2021-01-23 18:41:57.327000+00:00	2021-01-23 18:41:57.327000+00:00
2021-01-23 18:42:58.343000+00:00	2021-01-23 18:42:58.343000+00:00
2021-01-23 18:43:59.220000+00:00	2021-01-23 18:43:59.220000+00:00
2021-01-23 18:45:00.030000+00:00	2021-01-23 18:45:00.030000+00:00
2021-01-23 18:46:02.613000+00:00	2021-01-23 18:46:02.613000+00:00

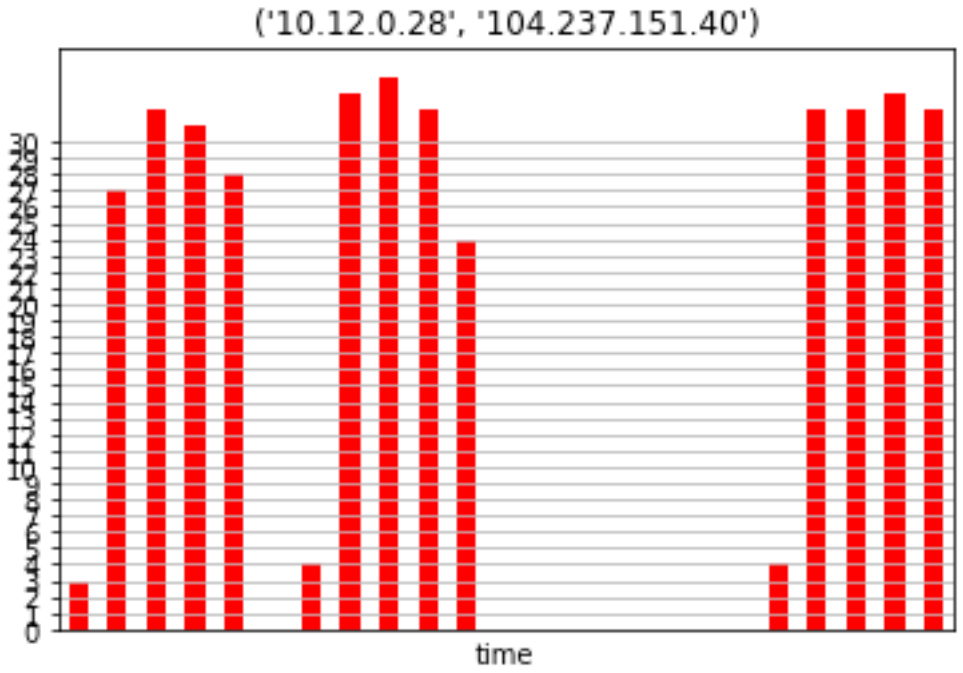
- **Malware check-in approx. every 60 sec.**
- **Mostly from cscript.exe, but also Teams**
- **More than one IP for script.google.com**

1247 rows × 1 columns

	Hostname	UserName	ProcessPath	FileName	ProcessId	LocalIP	RemoteIP	RemotePort	count_
0	Work-0017-RP	ACCOUNTSPAYABLE\rparker	C:\Windows\System32\cscript.exe	cscript.exe	3732	10.10.0.2	172.217.2.110	443	1637
1	Work-0017-RP	ACCOUNTSPAYABLE\rparker	C:\WINDOWS\system32\cscript.exe	cscript.exe	3732	10.10.0.2	172.217.2.110	443	1
2	Work-0017-RP	ACCOUNTSPAYABLE\Administrator	C:\Users\Administrator\AppData\Local\Microsoft...	Teams.exe	6808	10.10.0.2	172.217.2.110	443	1

COBALT STRIKE BEACON

Score: 0.57, skew: 0.93, madm:0.68, count: 0.094



2021-01-18 22:18:08.177000+00:00	2021-01-18 22:18:08.177000+00:00
2021-01-18 22:18:43.670000+00:00	2021-01-18 22:18:43.670000+00:00
2021-01-18 22:19:21.720000+00:00	2021-01-18 22:19:21.720000+00:00
2021-01-18 22:20:05.197000+00:00	2021-01-18 22:20:05.197000+00:00
2021-01-18 22:20:41.490000+00:00	2021-01-18 22:20:41.490000+00:00
...	...
2021-01-19 01:58:12.567000+00:00	2021-01-19 01:58:12.567000+00:00
2021-01-19 01:58:46.650000+00:00	2021-01-19 01:58:46.650000+00:00
2021-01-19 01:58:52.230000+00:00	2021-01-19 01:58:52.230000+00:00
2021-01-19 01:59:20.230000+00:00	2021-01-19 01:59:20.230000+00:00
2021-01-19 01:59:23.003000+00:00	2021-01-19 01:59:23.003000+00:00

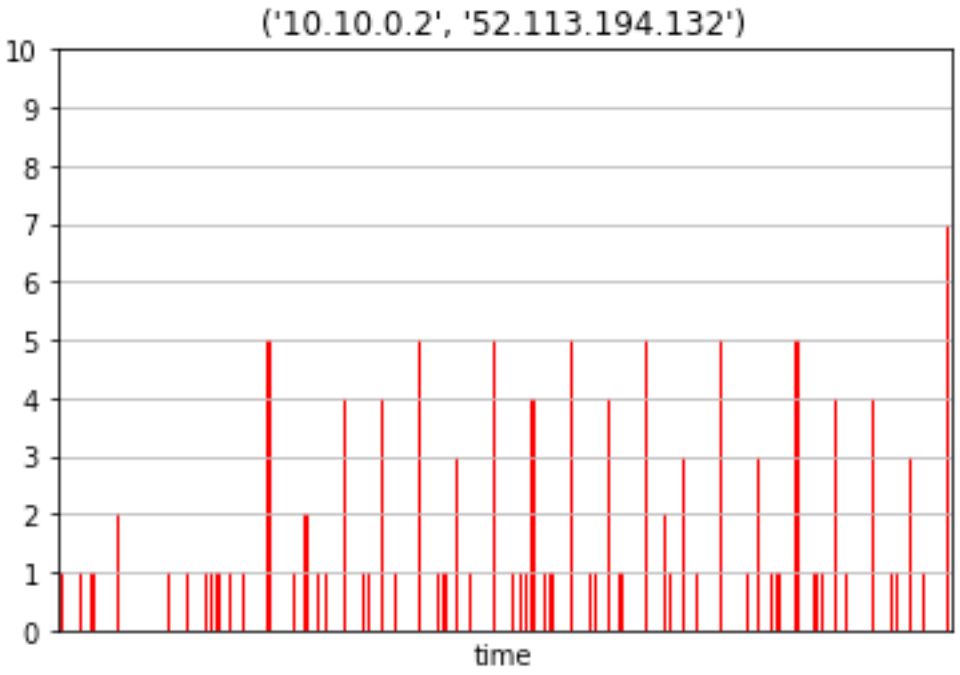
381 rows × 1 columns

- **Beacon interval: 45 sec. with 37% jitter**
- **Stopped and started several times in 24hrs**
- **Helpful to filter list for high # of connections**

	Hostname	UserName	ProcessPath	FileName	ProcessId	LocalIP	RemoteIP	RemotePort	count_
0	Work-0013-RS	WORK-0013-RS\Rheese Spiess	C:\Users\Rheese Spiess\Desktop\Stager\beacon_n...	beacon_nostage.exe	4504	10.12.0.28	104.237.151.40	443	662
1	Work-0013-RS	WORK-0013-RS\Rheese Spiess	C:\Users\Rheese Spiess\Desktop\Stager\beacon_n...	beacon_nostage.exe	3360	10.12.0.28	104.237.151.40	443	646

MICROSOFT TEAMS

Score: 0.49, skew: 0.77, madm:0.0, count: 0.68



time	
2021-01-22 19:10:04.453000+00:00	2021-01-22 19:10:04.453000+00:00
2021-01-22 19:40:13.853000+00:00	2021-01-22 19:40:13.853000+00:00
2021-01-22 20:00:04.770000+00:00	2021-01-22 20:00:04.770000+00:00
2021-01-22 20:40:03.953000+00:00	2021-01-22 20:40:03.953000+00:00
2021-01-22 20:40:52.543000+00:00	2021-01-22 20:40:52.543000+00:00
...	...
2021-01-23 18:40:12.770000+00:00	2021-01-23 18:40:12.770000+00:00
2021-01-23 18:40:22.663000+00:00	2021-01-23 18:40:22.663000+00:00
2021-01-23 18:40:32.547000+00:00	2021-01-23 18:40:32.547000+00:00
2021-01-23 18:40:36.963000+00:00	2021-01-23 18:40:36.963000+00:00
2021-01-23 18:41:01.923000+00:00	2021-01-23 18:41:01.923000+00:00

- **This is fine**

ProcessPath	FileName	ProcessId	LocalIP	RemoteIP	RemotePort	count_
C:\Users\rparker\AppData\Local\Microsoft\Teams...	Teams.exe	7884	10.10.0.2	52.113.194.132	443	17
C:\Users\Administrator\AppData\Local\Microsoft...	OneDriveSetup.exe	8956	10.10.0.2	52.113.194.132	443	1

BINARY DEFENSE™

QUESTIONS?