Dynamic Correlation, ML and Hunting

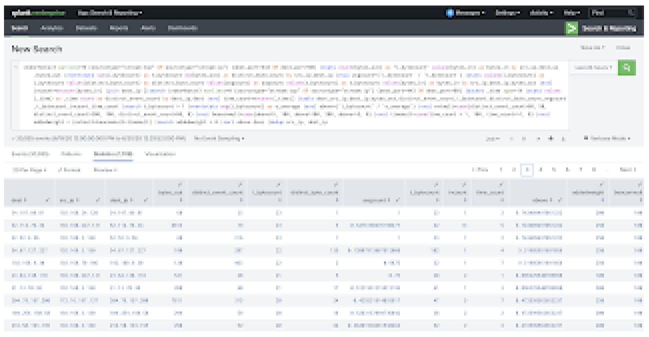
findingbad.blogspot.com/2020/06/dynamic-correlation-ml-and-hunting.html

Hunting has been my primary responsibility for the last several years. Over this time I've done a lot of experimentation around different processes and methods of finding malicious activity in log data. What has always stayed true though is the need for a solid understanding of the hypothesis you're using, familiarity with all the data you can take advantage of and a method to produce/analyze the results. For this post I'd like to share one of the ideas I've been working on lately.

I've previously written a number of blog posts on beaconing. Over time I've refined much of how I go about looking for these anomalous connections. My rule of thumb for hunting beacons (or other types of malicious activity) is to ignore static IOC's as those are best suited for detection. Instead, focus on behaviors or clusters of behaviors that will return higher confidence output. Here's how I'm accomplishing this in a single Splunk search.

1. Describe what you are looking for in numbers. This will allow you to have much more control over your conditional statements which impacts the quality of your output.
2. Define those attributes that you are interested in and assign number values to them. These attributes will be your points of correlation.
3. Reduce your output to those connections that exhibit any of what you are looking for. This is the correlation piece where we can use the total score of all attributes identified within a src/dest pair. Higher sums translate to greater numbers of attributes identified.

Below is a screenshot of the search I came up with. This is again using the botsv3 data set from Splunk's Boss of the SOC competition. Thanks Splunk!

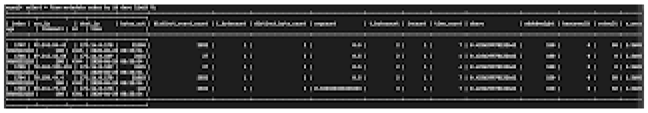


The following is a description of the fields in the output.

-dest: Based on the data source, this field may include the ip address or domain name.

-src_ip: Source of request
-dest_ip: Destination of request
-bytes_out: Data sent from src_ip.
-distinct_event_count: The total number of connections per destination.
-i_bytecount: The total count of bytes_out by src/dest/bytes_out. Large numbers may indicate beaconing.
-t_bytecount: The total count of connections between src/dest.
-avgcount: i_bytecount / t_bytecount. Beacon percentage. Values closer to 1 are more indicative of a beaocn.
-distinct_byte_count: Total count of bytes_out (used in determining percentages for beaconing).
-incount: The count of unique bytes_in values. When compared with t_bytecount you may see the responses to beacons.
-time_count: The number of hours the src/dest have been communicating. Large numbers may indicate persistent beaconing.
-o_average: The average beacon count between all src/dests.
-above: The percentage above o_average.
-beaconmult: weight multiplier given to higher above averages.
-evtmult: weight multiplier given to destinations with higher volume connections.
-timemult: weight multiplier given to connections that last multiple hours.
-addedweight: The sum of all multipliers.

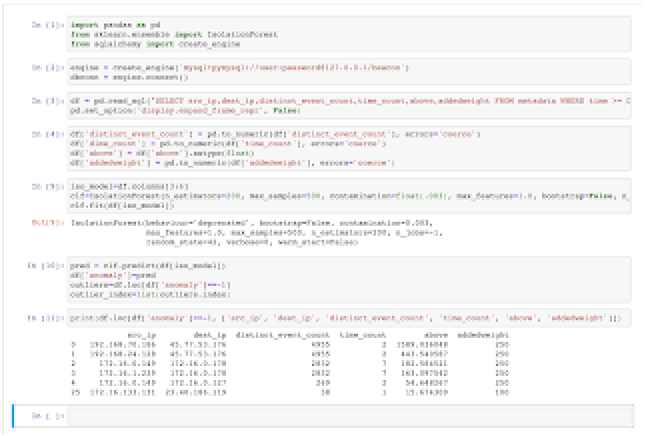
You can see from the search results that we reduced 30k+ events down to 1700 that exhibit some type of behavior that we're interested in. This is good, but still not feasible to analyze every event individually. I have a couple of choices to reduce my output at this point. I can adjust my weighted condition to something like "|where weighted > 100" which would have the effect of requiring multiple characteristics being correlated. My other choice is to use some type of anomaly detection to surface those odd connections. You can probably tell from the "ML" portion of the title which direction I'm going to go. So from here we need a method to pick out the anomalies as the vast majority of this data is likely legitimate traffic. For this I'll be inserting our results into a MySQL database. I don't necessarily need to for this analysis, but it's a way for me to keep the metadata of the connections for greater periods of time. This will allow me to do longer term analysis based on the data that is being stored.


```
[illegible]
```

Once it's in the database we can use python and various ML algorithms to surface anomalous traffic. For this I'll be using an Isolation Forest. I'll also be choosing fields that I think best represents what a beacon looks like as I don't want to feed every field through this process.

distinct_event_count: Overall activity.
time_count: How persistent is the traffic?
above: How does the beacon frequency compare to all other traffic?
addedweight: How many beacon characteristics does this traffic exhibit?

The following screenshot contains the code as well as the output.



```
In [1]: import pandas as pd
        from sklearn.ensemble import IsolationForest
        from sqlalchemy import create_engine

In [2]: engine = create_engine('mysql+pymysql://user:password@127.0.0.1/beacon')
        dbconn = engine.connect()

In [3]: df = pd.read_sql('SELECT src_ip,dest_ip,distinct_event_count,time_count,above,addedweight FROM metadata WHERE time >= C
        pd.set_option('display.expand_frame_repr', False)

In [4]: df['distinct_event_count'] = pd.to_numeric(df['distinct_event_count'], errors='coerce')
        df['time_count'] = pd.to_numeric(df['time_count'], errors='coerce')
        df['above'] = df['above'].astype(float)
        df['addedweight'] = pd.to_numeric(df['addedweight'], errors='coerce')

In [5]: iso_model=df.columns[2:6]
        clf=IsolationForest(n_estimators=100, max_samples=500, contamination=float(.003), max_features=1.0, bootstrap=False, n_
        clf.fit(df[iso_model])

Out[5]: IsolationForest(behaviour='deprecated', bootstrap=False, contamination=0.003,
                        max_features=1.0, max_samples=500, n_estimators=100, n_jobs=-1,
                        random_state=42, verbose=0, warm_start=False)

In [10]: pred = clf.predict(df[iso_model])
         df['anomaly']=pred
         outliers=df.loc[df['anomaly']==-1]
         outlier_index=list(outliers.index)

In [11]: print(df.loc[df['anomaly']==-1, ['src_ip', 'dest_ip', 'distinct_event_count', 'time_count', 'above', 'addedweight']]);

            src_ip        dest_ip  distinct_event_count  time_count        above  addedweight
0   192.168.70.186   45.77.53.176                  6955           2  1589.816848          250
1   192.168.24.118   45.77.53.176                  6955           2   441.540587          250
2     172.16.4.149   172.16.0.178                  2812           7   161.586511          250
3     172.16.1.219   172.16.0.178                  2812           7   161.097942          250
4     172.16.8.149   172.16.0.127                   249           2    54.648247          250
25  172.16.133.131  23.48.186.119                    18           1    15.614309          100

In [ ]:
```

Looking at the top 3 tenths of 1 percent of the most anomalous src/dest pairs you can see that there are 4 destination ip addresses that may need investigating. If you've read my last 2 posts on beaconing the 45.77.53.176 ip should look familiar. This ip was definitely used for C2. The 172.16.0.178 ip is also interesting. Taking a quick look at the destination in the botsv3 data, you can see memchached injection that appears to be successful. Additional investigation of the src ip's in this output would definitely be justified.

I will say that this method is very good at identifying beacons, but beacons are not always malicious. Greater work may be needed to surface those types malicious connections. Some additional ideas may be first seen ip's or incorporating proxy data where even more characteristics can be defined, scored and correlated.

A large portion of hunting is experimentation so experiment with the data and see what you can come up with!